

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

LINGI1113  
SYSTÈMES INFORMATIQUES 2

---

**Projet 2 : PIC horloge**  
**Rapport**

---



Groupe 43  
PESCHKE Lena 5826 11 00  
SEDDA Mélanie 2246 11 00

*Assistant :* Laurent Lesage

28 octobre 2013

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Documentation pour l'utilisateur</b>	<b>1</b>
2.1	Réglages . . . . .	1
2.2	Le réveil . . . . .	3
<b>3</b>	<b>Documentation pour l'installateur</b>	<b>4</b>
3.1	Compilation . . . . .	4
3.2	Installation . . . . .	4
3.3	Tests . . . . .	5
<b>4</b>	<b>Documentation pour le programmeur</b>	<b>5</b>
4.1	Spécifications . . . . .	5
4.2	Fréquence du PIC . . . . .	5
4.3	Choix d'implémentation . . . . .	6
4.3.1	Mesure du temps . . . . .	6
4.3.2	États du réveil . . . . .	8
4.3.3	Exécution du programme . . . . .	8
4.3.4	Interruptions . . . . .	8
4.3.5	Autres choix . . . . .	9
<b>5</b>	<b>Conclusion</b>	<b>9</b>
	<b>Références</b>	<b>9</b>

# 1 Introduction

Le but de ce mini-projet a été de nous familiariser avec la programmation en C sur une machine dite “nue”, c'est-à-dire sans réel système d'exploitation. Pour cela nous avons programmé un réveil matin sur une carte OLIMEX PIC MaxiWeb. Cette carte inclut un microcontrôleur de la firme Microchip et les programmes y ont un accès direct à la mémoire et aux périphériques. En particulier, nous avons dû utiliser nous-mêmes les interruptions du “timer” pour déterminer l'heure. [Lob13]

Vous trouverez dans ce rapport des explications utiles pour l'utilisateur (c'est-à-dire tout ce qui concerne simplement l'utilisation du réveil lorsqu'il est déjà installé sur le PIC), pour l'installateur (les instructions décrivant comment compiler, installer sur le PIC et tester notre programme) et pour le programmeur (tout ce qui est nécessaire à un programmeur qui désirerait adapter notre programme). Dans cette dernière partie, nous détaillerons notamment les spécifications de notre programme, la façon dont nous mesurons le temps qui passe, ainsi que nos divers choix d'implémentation.

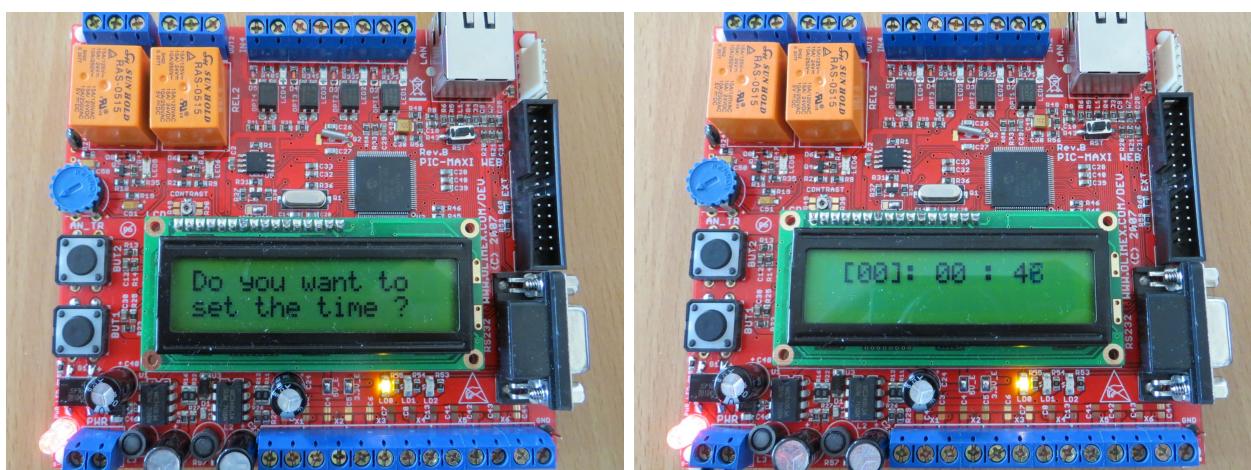
## 2 Documentation pour l'utilisateur

Le réveil se compose de :

- 1 écran LCD
- 1 lampe LED jaune
- 2 lampes LED rouges
- 1 bouton MENU/NEXT/STOP (bouton 1, en bas)
- 1 bouton SELECT/ADD/SNOOZE (bouton 2, en haut)

Il comporte un mode d'affichage, un menu de l'horloge et un menu de l'alarme.

### 2.1 Réglages



(a) Menu

(b) Modification de l'heure

FIGURE 1 – Modification de l'heure de l'horloge

**Première utilisation** Par défaut, l'horloge commence à 00 : 00 : 00, l'alarme est désactivée et réglée pour 00 : 00. Nous proposerons uniquement à l'utilisateur de choisir l'heure et la

minute du réveil parce que cela n'a pas beaucoup de sens de vouloir qu'il sonne à un nombre de secondes précis et c'est d'ailleurs ce qui se fait sur tous les réveils. Lors de la mise sous tension du PIC, vous entrez dans le menu de l'horloge (Figure 1a).

**Menu de l'horloge** L'accès au menu de l'horloge se fait via le bouton MENU. Une fois sur ce menu, vous pouvez soit appuyer sur le bouton NEXT pour passer au menu de l'alarme (Figure 2a), soit poussez sur SELECT pour pouvoir changer l'heure (Figure 1b). Vous pouvez maintenant incrémenter les heures en appuyant sur ADD. Pour passer aux minutes et ensuite aux secondes, il suffit d'appuyer sur NEXT successivement, et de les augmenter en poussant ADD. Les crochets vous indiquent quelle valeur est en cours de modification. Arrivé aux secondes, le bouton NEXT vous donne accès au menu de l'alarme (Figure 2a).

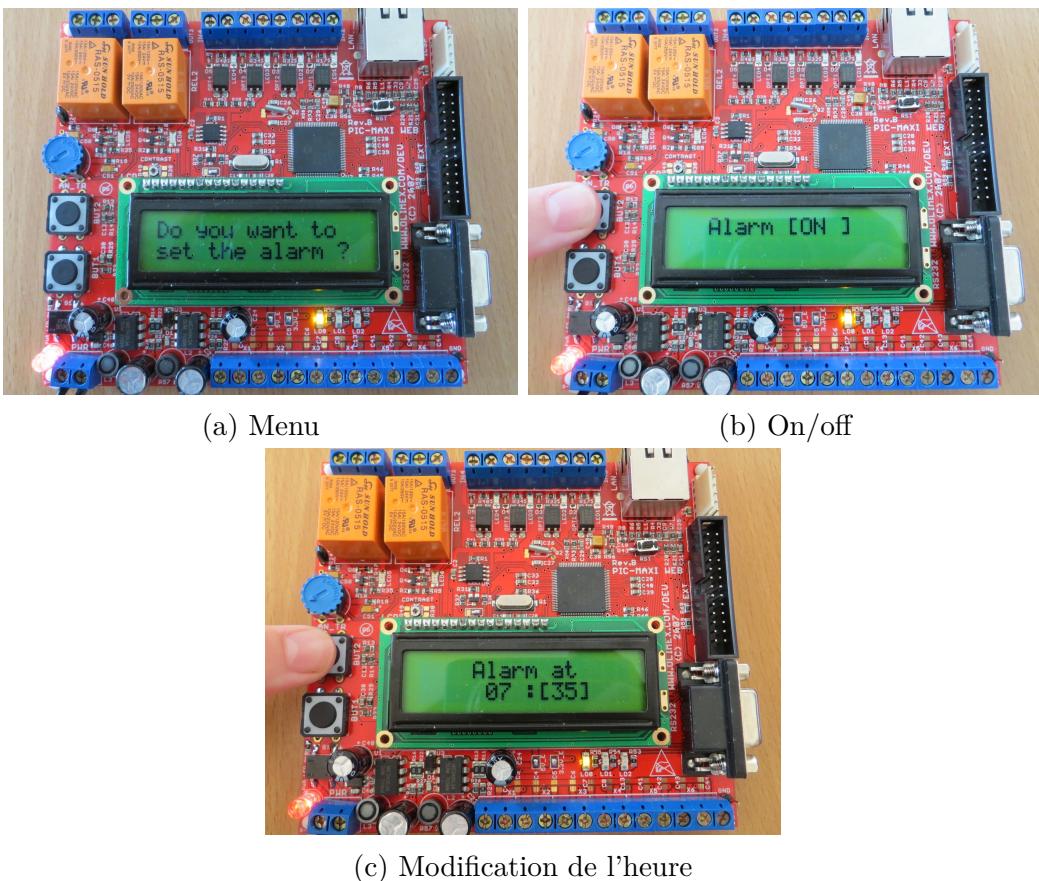


FIGURE 2 – Modification de l'heure de l'alarme

**Menu du l'alarme** Si vous désirez revenir à l'affichage de l'heure (Figure 3), poussez le bouton NEXT; pour régler l'alarme, poussez le bouton SELECT (Figure 2b). En appuyant sur ADD, vous pouvez activer ou désactiver l'alarme. L'écran vous indique l'état actuel par ON ou OFF. Le bouton NEXT vous fait accéder à l'heure de l'alarme (Figure 2c), dont vous pouvez changer la valeur en poussant sur ADD. NEXT vous permet de passer aux minutes, que vous pouvez également modifier avec le bouton ADD. Il n'est pas possible de modifier les secondes de l'alarme étant donné que c'est une fonctionnalité que notre équipe technique a jugée inutile. Enfin, le bouton NEXT vous fait quitter le menu et vous amène à l'affichage de l'heure (Figure 3).

**L'affichage** Le réveil affiche l'heure sous le format hh : mm : ss sur la première ligne de l'écran. La seconde ligne indique si l'alarme est mise (Figure 3b) ou non (Figure 3a) et à quelle heure elle est réglée. Pour accéder au menu (Figure 1a), il suffit d'appuyer sur le bouton MENU et de le parcourir avec les boutons NEXT et SELECT. La LED jaune clignote en continu avec une période de 1 seconde. De cette façon, à n'importe quel endroit du menu, on peut visualiser les secondes qui passent.

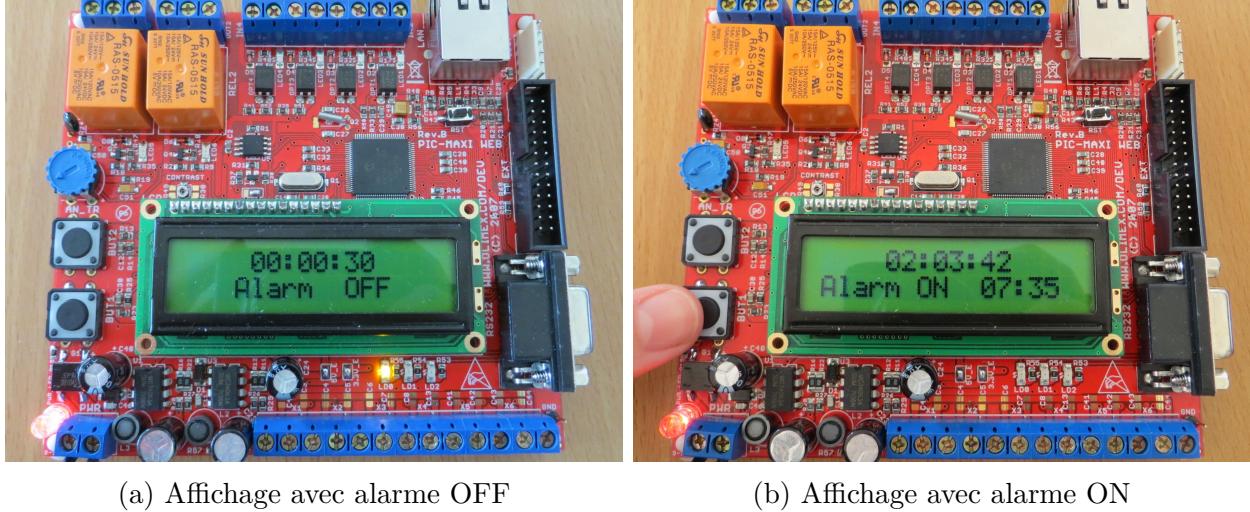


FIGURE 3 – Affichage principal

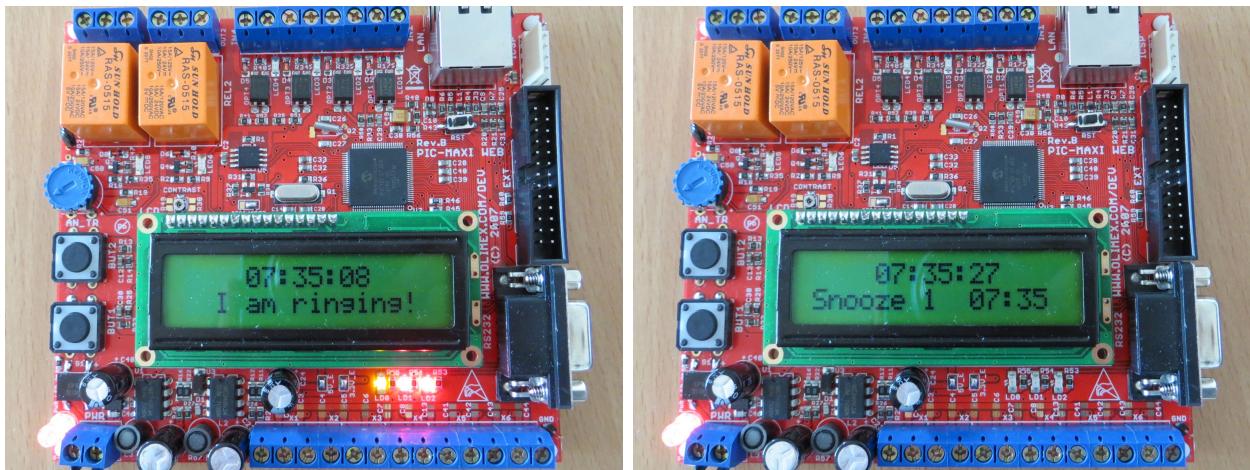


FIGURE 4 – Gestion de la sonnerie de l'alarme

## 2.2 Le réveil

Lorsque le réveil sonne, les deux LED rouges clignotent avec une période de 1 seconde (Figure 4). La sonnerie dure 30 secondes, délai après lequel le réveil s'éteint automatiquement. Il reste toutefois activé pour une prochaine utilisation le lendemain.

Lors de l'alarme, il y a deux options possibles :

**stop** Soit vous souhaitez l'éteindre ; il suffit alors d'appuyer sur STOP. Le réveil sonnera à nouveau dans 24 heures.

**snooze** Soit vous désirez reporter le réveil de 5 minutes ; appuyez alors sur SNOOZE. L'écran vous affiche maintenant l'heure en cours, le nombre de snoozes effectués et l'heure de réveil d'origine (Figure 4b). Pendant le mode snooze, ou à chaque fois que le réveil sonne à nouveau, vous pouvez reporter le réveil de 5 minutes supplémentaires, jusqu'à un maximum de 60 minutes au total. Vous pouvez aussi à tout moment appuyer sur STOP, ce qui aura pour effet d'arrêter le réveil, de revenir à l'affichage normal de l'heure et de remettre l'alarme à l'heure d'origine pour une nouvelle utilisation le lendemain (Figure 3b).

Il n'est pas possible d'accéder au menu tant que le réveil n'a pas été éteint. De même, tant que vous êtes dans le menu, le réveil ne sonne pas, pour ne pas interférer avec d'éventuels changements en cours.

## 3 Documentation pour l'installateur

Le dossier comprend deux programmes :

- `reveil.c`, le réveil
- `findfreq.c`, pour déterminer la fréquence du PIC (vous verrez s'afficher le nombre d'overflows du timer0 après 10 minutes, voir section 4.2 pour plus de détails)

Les instructions suivantes décrivent les étapes à suivre pour installer le réveil sur le PIC. La procédure est la même pour le second programme, il faut juste changer le nom.

### 3.1 Compilation

Pour compiler `reveil.c`, allez dans le dossier `PIChorloge/programme/` et tapez la commande `make reveil` dans le terminal.

### 3.2 Installation

Commencez par brancher le câble d'alimentation du PIC au routeur et mettez-le sous tension (220V AC). Branchez le PIC au port 3 du routeur à l'aide d'un des câbles ethernet ; branchez ensuite votre ordinateur au port 2 du routeur avec l'autre câble ethernet. La LED du port 2 du routeur doit s'allumer.

Dans le terminal, tapez les commandes suivantes pour vous connecter au routeur et transférer le fichier `reveil.hex` au PIC :

```
my_computer $ tftp 192.168.97.60      // press 'enter'  
my_computer $ tftp> put alarm.hex      // wait!
```

Avant d'appuyer sur "enter" pour la seconde commande, resettez le PIC (petit bouton "reset" derrière le port ethernet) et attendez que le LED du port 3 du routeur s'allume. Vous avez à présent 3 secondes pour effectuer le transfert.

Pour quitter le client tftp, tapez

```
my_computer $ tftp> quit                // press 'enter'
```

Le programme se lance automatiquement sur le PIC. Pour le faire redémarrer, appuyez sur le bouton "reset". Vous pouvez à présent débrancher les câbles ethernet.

### 3.3 Tests

Pour évaluer le bon fonctionnement du réveil, nous vous suggérons les tests suivants.

- Si vous avez beaucoup de temps, vous pouvez tester le décalage de l'heure de l'horloge sur plusieurs heures.
- Vous pouvez tester si l'alarme, une fois qu'elle sonne, s'arrête toute seule au bout de 30 secondes.
- Vous pouvez tester si l'alarme est toujours mise et réglée à la même heure une fois qu'elle a été éteinte après une sonnerie.
- Vous pouvez répéter le test précédent après un nombre quelconque de snoozes.
- Vous pouvez vérifier s'il est bien impossible de dépasser la limite de 12 snoozes et si le réveil sonne bien 1 heure après l'heure initialement réglée dans ce cas.
- Vous pouvez tester si l'horloge a bien continué à tourner normalement après un long passage par le menu (pendant lequel l'heure n'a évidemment pas été changée).

## 4 Documentation pour le programmeur

### 4.1 Spécifications

Le réveil satisfait aux fonctionnalités suivantes :

- Il affiche l'heure et la met à jour au moins une fois par seconde. Une LED jaune clignote toutes les secondes.
- Il contient un réveil qui, activé, "sonne" pendant 30 secondes. La sonnerie est remplacée par le clignotement de LED rouges toutes les secondes.
- Il est possible d'arrêter la sonnerie de l'alarme sans désactiver celle-ci.
- Il est possible de régler l'heure sans interrompre l'horloge.
- Il est possible de régler et d'activer/désactiver l'alarme.
- Il est possible de postposer l'alarme de 5 minutes pendant une heure.

### 4.2 Fréquence du PIC

Pour déterminer l'heure, nous avons dû nous baser sur le timer0 de notre PIC. Ce timer est un registre qui est incrémenté automatiquement à chaque coup d'horloge, et qui, lors d'un débordement (overflow), génère une interruption. Nous avons alors effectué quelques tests nous permettant de mesurer à quelle fréquence ce registre est incrémenté ; c'est sur cette valeur que nous avons ensuite pu nous baser pour concevoir notre réveil-matin. Nous allons ici vous présenter notre stratégie.

Outre le timer0, le PIC contient un timer1 qui peut être programmé pour déborder après exactement 2 secondes<sup>1</sup>. Notre stratégie a alors été de mesurer le nombre de fois que le timer0 déborde pendant ce laps de temps. Nous estimerons alors que

$$f_{software} = \frac{overflows \cdot 2^{16}}{2}$$

car le registre fait 16 bits et

$$f_{hardware} = 4 \cdot f_{software}$$

comme expliqué dans la documentation du PIC [Mic06].

---

1. Nous n'avons toutefois pas eu l'autorisation de l'utiliser dans le cadre du réveil-matin en lui-même.

Afin d'avoir une erreur relative la plus petite possible, nous n'avons pas utilisé de prescaler<sup>2</sup> sur le timer0 et avons effectué notre mesure sur plus de 2 secondes. En effet, si on considère ici  $f$  comme étant la fréquence hardware, qu'on appelle *value* la valeur qui se trouve dans le timer0 pile au moment de l'overflow du timer1 et si  $N$  est le temps de notre mesure en secondes, nous pouvons aussi écrire

$$\begin{aligned} f_{exacte} &= \frac{4 \cdot (overflows \cdot 2^{16} + value)}{N} \\ f_{mesurée} &= \frac{4 \cdot (overflows \cdot 2^{16})}{N} \\ \text{erreur relative} &= \frac{|f_{exacte} - f_{mesurée}|}{f_{exacte}} \\ &= \frac{value}{overflows \cdot 2^{16} + value} \end{aligned}$$

Notre erreur relative sera donc plus petite lorsque *overflows* est plus grand. C'est pourquoi utiliser un prescaler sur le timer0 serait contreproductif (cela diminuerait le nombre d'overflows) et augmenter le temps de mesure est intéressant (cela augmentera le nombre d'overflows).

Nous avons pris le parti de ne pas mesurer la valeur se trouvant dans le timer0 après l'overflow du timer1 parce que cette valeur est négligeable. L'erreur relative maximale en ne la mesurant pas est en effet  $\leq \frac{1}{overflows+1}$ . De plus, on ne trouverait de toute façon pas précisément la valeur qui se trouvait dans le timer0 pile au moment de l'overflow.

Nous avons donc créé un programme `findfreq.c` qui compte et affiche le nombre d'overflows sur un laps de temps de 10 minutes. Nous avons effectué 3 mesures et avons trouvé qu'en moyenne

$$\begin{aligned} \text{le nombre d'overflows/s} &= 95,366 \\ f_{software} &= 95,366 \cdot 2^{16} = 6249906,176 \\ f_{hardware} &= 6249906,176 \cdot 4 = 24999624 \text{ Hz} \end{aligned}$$

ce qui représente un écart de seulement 0.0015% par rapport à la fréquence de 25 MHz annoncée dans la documentation du PIC [Mic06].

## 4.3 Choix d'implémentation

### 4.3.1 Mesure du temps

Plusieurs choix d'implémentation s'offraient à nous.

Tout d'abord, il a fallu choisir si nous voulions nous servir d'interruptions ou faire de l'attente active, c'est-à-dire examiner régulièrement la valeur du timer dans une boucle jusqu'à ce qu'il atteigne la valeur souhaitée. Nous avons décidé de nous servir des interruptions parce qu'elles prennent le dessus sur le restant du programme. Comme le calcul du temps se base sur le nombre d'overflows, l'incrémentation de ce compteur doit être prioritaire par rapport au reste.

Un deuxième choix, indépendant du premier, est de laisser le temporisateur tourner librement ou non. Ce qu'on entend par ne pas laisser tourner le compteur librement est changer la valeur

---

2. option qui permet de diviser la fréquence d'incrémentation du registre par une puissance de 2

dans le timer à un moment donné. Par exemple, on pourrait calculer le nombre d'incréments qui correspondent à 1 seconde et se dire qu'on va changer la valeur dans le timer0 à  $2^{16}$  moins cette valeur afin qu'il déborde après une seconde. Cette technique a un inconvénient majeur : il y a un délai entre le débordement du compteur et le moment où on le réinitialise ; il faut donc soustraire de la valeur calculée précédemment la valeur du compteur au moment où on le réinitialise. Cette valeur est difficile à calculer de façon précise et pourrait en plus à priori ne pas être constante. Nous avons donc rejeté cette solution et avons décidé de laisser tourner le compteur librement.

Un autre choix à effectuer a été d'utiliser le prescaler ou non. Nous avons décidé de ne pas l'utiliser tout simplement parce que cela diminuerait la fréquence d'incrémentation du timer0 et augmenterait donc l'incertitude sur le temps écoulé. Au contraire, il aurait plutôt été intéressant de l'augmenter. Mais cela n'est pas possible et on prendrait le risque que des interruptions ne soient pas traitées parce qu'elles arrivent en même temps qu'une autre. Notons que, sans prescaler, nous ne pensons pas que ce type d'événement arrive souvent vu la fréquence très proche de 25 MHz que nous avons trouvée.

Il nous a ensuite fallu établir une vraie stratégie permettant de calculer l'heure sur base des interruptions du timer0 s'incrémentant normalement. Une première idée que nous avons eue était d'avoir des variables `heure`, `minute` et `seconde` et un compteur d'overflows “qui n'ont pas encore été traités”. Supposons que nous ayions trouvé qu'il fallait 95,5 overflows pour avoir 1 seconde. Nous aurions alors défini qu'à chaque fois que overflows dépasse un multiple de 95, on incrémenterait nos secondes de 1 (avec une bonne gestion de valeurs maximales des variables et des reports entre elles évidemment) et nous aurions ensuite corrigé l'erreur commise en disant que la fréquence valait 95 en posant qu'une fois que `overflows` =  $2 \cdot 95$ , nous remettions `overflows` à  $-1$ . Cette technique n'est cependant pas du tout applicable dans notre cas car nous avons trouvé 95,336 overflows ; pour garder la même précision, il faudra attendre 500 secondes et mettre `overflows` à  $-168$ , ce qui correspond à plus d'une seconde à retirer. Cela veut dire qu'on aurait soudainement un recul d'une seconde qui se verrait clairement sur le LCD, et ce n'est pas ce que nous désirons.

Ce que nous avons finalement fait pour garder notre précision est de garder un compteur du nombre d'overflows depuis le lancement du programme. Nous pouvons ensuite convertir ce compteur en un délai heure, minute, seconde sur base de simples calculs basés sur le fait que 1 seconde correspond à 95,336 overflows. Il suffit alors de modifier ces variables lorsque nos calculs aboutissent à des valeurs différentes. La modification de l'heure par l'utilisateur modifiera donc directement cette unique valeur aussi. Il faudrait juste ajouter à `overflows` le nombre d'overflows adéquat.

Pour pouvoir compter le temps le plus longtemps possible, nous avons choisi de stocker le compte des overflows dans une variable de type `unsigned long long`. Celle-ci nous permet de compter jusqu'à  $2^{64}$  ticks, ce qui correspond à environ  $5,37 \cdot 10^{13}$  heures. Nous estimons qu'à cette échelle, le réveil fait preuve d'une durée de vie assez longue avant de devoir être resetté, ce qui peut se faire très facilement<sup>3</sup>.

---

3. Il suffit d'appuyer sur le bouton “reset” du PIC ou de débrancher et rebrancher l'alimentation.

#### 4.3.2 États du réveil

Le réveil peut se trouver dans 11 états différents. Ils sont définis en début de programme au moyen de `#define`. Ils permettent de structurer le code de manière claire et de gérer chaque événement dans son contexte.

- TIME\_MENU 1 : Permet d'accéder au menu de l'horloge.
- SET\_HOUR 2 : Permet de changer l'heure de l'horloge.
- SET\_MINUTE 3 : Permet de changer les minutes de l'horloge.
- SET\_SECOND 4 : Permet de changer les secondes de l'horloge.
- ALARM\_MENU 5 : Permet d'accéder au menu de l'alarme.
- SET\_ALARM 6 : Permet d'activer/désactiver l'alarme.
- SET\_A\_HOUR 7 : Permet de changer l'heure du réveil.
- SET\_A\_MIN 8 : Permet de changer les minutes du réveil.
- DISPLAY 9 : Affichage normal.
- ALARM 10 : Le réveil sonne.
- SNOOZE 11 : Le réveil a été reporté.

A chacun de ces états correspond

- un affichage sur le LCD repris dans la méthode `refresh_lcd()` et
- une modification à appliquer si on appuie sur l'un des deux boutons repris dans la méthode `button()`.

#### 4.3.3 Exécution du programme

Après l'initialisation des différents registres nécessaires à son fonctionnement, le programme tourne en boucle pour vérifier

- si l'heure a changé et quelles LED il faut allumer ou éteindre (`time()`)
- s'il faut mettre à jour l'affichage (`refresh_lcd()`)
- s'il faut lancer ou arrêter l'alarme (`alarm()`)
- si l'utilisateur a appuyé sur un bouton (`button()`).

#### 4.3.4 Interruptions

Il y a deux routines d'interruptions à des niveaux de priorités différents :

- le niveau 1 génère une interruption dès que le timer0 overflow et ne fait que compter le nombre d'overflows qui se sont produits (la section 4.3.1 explique pourquoi nous avons choisi d'utiliser des interruptions plutôt que de faire de l'attente active) ;
- le niveau 2 génère une interruption lorsqu'on appuie sur un bouton et modifie le flag du bouton correspondant.

Nous avons véritablement voulu réduire nos routines d'interruption au strict minimum pour éviter d'avoir des interruptions qui arrivent lorsqu'une autre s'exécute. C'est la fonction `main()`, et plus particulièrement les fonctions `time()` et `button()`, qui vont traiter les informations modifiées dans nos routines.

Le choix de donner une priorité plus importante à l'incrémentation d'overflows qu'à la pression sur les boutons est assez naturel : nous n'avons pas envie de manquer des overflows tandis qu'il est acceptable d'attendre un peu pour les boutons.

#### 4.3.5 Autres choix

Pour ne pas devoir faire de suppositions sur le fonctionnement du compilateur et notamment sur ce qu'il fait des appels de fonctions imbriqués et pour ne pas risquer de faire grandir la taille de la pile plus que nécessaire, nous avons décidé de réduire au maximum les appels de fonctions. C'est pour cette raison que notre programme ne contient que peu de méthodes.

Nous avons choisi de faire un menu qui comporte relativement beaucoup d'écrans différents. L'idée principale derrière ce choix est que nous n'avions que deux boutons à notre disposition. Pour cependant avoir les fonctionnalités que nous voulions implémenter, il a donc fallu créer des modes différents dans le contexte desquels les boutons ont des fonctions différentes. Par la suite, nous nous sommes rendues compte d'un autre avantage : notre alarme ne sonne pas lorsque l'utilisateur se trouve dans le menu ; étant donné que ce dernier est clairement identifiable et différent du mode "affichage", l'utilisateur ne peut pas rester dans le menu sans s'en rendre compte et s'étonner du non fonctionnement de son réveil.

En ce qui concerne le réveil en tant que tel, nous avons pris le parti de le laisser activé une fois qu'il a sonné, que l'utilisateur l'ait éteint ou non. Nous avons en effet considéré le cas d'une utilisation quotidienne pour laquelle il est plus aisé de ne pas devoir régler et activer le réveil tous les jours. De même, lorsque le réveil a été postposé (snooze) et ensuite éteint, peu importe le nombre de snoozes et le moment de l'arrêt, il est remis à l'heure initiale.

Lors d'un changement manuel de l'heure, l'horloge continue à tourner en temps réel, sans interruption. D'ailleurs, elle ne s'arrête à aucun moment de la manipulation du réveil.

Lorsqu'on change l'heure manuellement, les heures, minutes et secondes sont désolidarisées, c'est-à-dire qu'incrémenter les minutes, par exemple, au-delà de 59 aura pour seul effet de les faire repasser à 0, mais pas d'augmenter les heures. Ainsi, un seul appui de trop sur le bouton ne nécessitera pas de devoir reparcourir tout le menu pour accéder à la valeur précédente.

## 5 Conclusion

Ce projet nous a permis de travailler sur une machine sans réel système d'exploitation pour la première fois. Nous avons été amenées à directement manipuler des registres et à nous poser des questions de priorité des opérations qui sont habituellement gérées par la machine sur laquelle nous travaillons. Pour implémenter une horloge et un réveil, nous avons dû, plus spécifiquement, utiliser des interruptions et coordonner l'ordre des opérations entre elles. Ensuite, nous avons posé un certain nombre de choix d'implémentations. Certains sont liés aux spécifications de la machine, d'autres reflètent plutôt un fonctionnement désiré du programme. Enfin, comme dans tout projet d'informatique, nous avons parcouru les différentes phases de conception, d'implémentation, de tests et de corrections. Le projet final nous paraît répondre aussi bien aux exigences du cahier des charges qu'aux nôtres.

## Références

- [Lob13] Marc Lobelle. Lingi1113 - projet 2 : exercice sur machine nue. [http://foditic.org/LINGI1113\\_14/missions/Projet2.php](http://foditic.org/LINGI1113_14/missions/Projet2.php), October 2013.
- [Mic06] Microchip Technology Inc. *PIC18F97J60 Family Data Sheet*, 2006.