# TP1 : Initiation

Nom : ..... Prénom : .....

Nom : ..... Prénom : .....

Groupe : ..... Binôme : .....

L'objectif principal de ce TP est de se familiariser avec les outils logiciels et matériels qui seront utilisés dans les TP suivants. En particulier, il s'agit de prendre en main la carte à microcontrôleur MSP430 et le logiciel *IAR Embedded Workbench* qui nous servira d'environnement de travail. Ce document vous guidera au cours de cette prise en main. Il contient un certain nombre de questions et servira de compte-rendu (merci de répondre aux questions dans l'espace laissé à cet effet [1]).

## Documentations techniques

Bien que cet énoncé apporte quelques explications quant au fonctionnement de la carte et du MSP430, bien souvent il ne s'agit que qu'un rapide résumé. Pour comprendre les détails, il vous faudra souvent aller chercher des informations supplémentaires dans les diverses documentations techniques, disponibles sur la page Moodle des TP d'architecture. Ne les imprimez pas sans réfléchir (certains font plusieurs centaines de pages) mais prenez le temps de les parcourir afin de pouvoir retrouver rapidement toutes les informations nécessaires au cours des TPs.

[Motherboard.pdf] *MSP430FG4618/F2013 Experimenter's Board User's Guide*.

[CPU.pdf] Chapter 3 of *MSP430x4xx Family User's Guide*.

[MSP430.pdf] *MSP430x4xx Family User's Guide*.

[datasheet.pdf] *MSP430xG461x Device Datasheet*.

[LCD.pdf] SoftBaugh *SBLCDA4 Specification*.

[workbench.pdf] *MSP430 IAR Embedded Workbench IDE User Guide*.

[compiler.pdf] *MSP430 IAR C/C++ Compiler Reference Guide*.

[assembler.pdf] *MSP430 IAR Assembler Reference Guide*.

[linker.pdf] *IAR Linker and Library Tools Reference Guide*.

[libc.pdf] *IAR C LIBRARY FUNCTIONS Reference Guide*.

## 1 Découverte de la carte

### 1.1 Présentation de la carte

La carte que nous allons utiliser en TP (voir figure 1) est produite par Texas Instruments, et commercialisée comme plate-forme d'expérimentation et de prototypage à destination d'autres constructeurs souhaitant évaluer les possibilités des microcontrôleurs MSP430. Afin de réaliser les TP, vous aurez donc besoin de comprendre le fonctionnement du microcontrôleur mais aussi celui de la carte mère.

1. Pour celles et ceux qui souhaiteraient répondre directement dans le document LaTeX, le fichier source est sur Moodle.

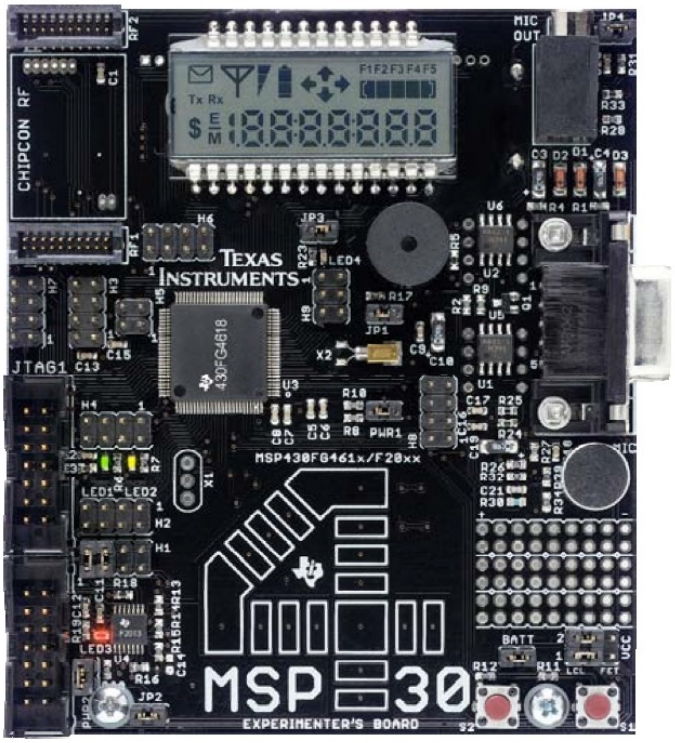


FIGURE 1: La carte MSP430-FG4618/F2013

## 1.2 Périphériques externes

En plus du MSP430 proprement dit, auquel on s'intéressera dans la section suivante, de nombreux périphériques externes sont présents sur la carte mère, ce qui permet de prototyper facilement une assez grande variété d'applications. Entre autres, on trouve :

- un écran à cristaux liquides, avec huit caractères sept-segments et plusieurs autres icônes
- un microphone
- une sortie casque analogique
- un buzzer
- un oscillateur à quartz
- deux boutons poussoirs
- une roue tactile capacitive (*touchpad*) en forme de chiffre 4
- un port série (RS-232)
- un emplacement pour puces radio ChipCon (WiFi, Bluetooth, ZigBee).

**Exercice 1 —** En utilisant la documentation de la carte [Motherboard.pdf], identifiez ces composants, et entourez chacun d'entre eux sur la photo ci-dessus.

## 1.3 Microcontrôleurs MSP430-FG4618 et MSP430-F2013

Parler de microcontrôleur MSP430 est un abus de langage. En effet, MSP430 désigne une famille de microcontrôleurs. Ainsi, comme on peut le voir dans la documentation, la carte comporte en réalité *deux* micro-contrôleurs MSP430, de type FG4618 et F2013. Ces composants sont tous deux issus de la famille des microcontrôleurs MSP430, mais ils sont situés aux deux extrémités de la «gamme» : alors que le FG4618 dispose de 8 Ko de mémoire vive (RAM) et de plus de 100 Ko de mémoire morte (flash), le F2013 ne possède que 2 Ko de flash, et seulement 128 octets de RAM ! Le second MSP430 est utilisé ici finalement comme un composant «auxiliaire», seulement destiné à piloter le *touch pad*. Au contraire, le FG4618 est au cœur du système. Il permet de contrôler tous les autres périphériques. C'est à lui que nous allons nous intéresser dans ces TP.

## 1.4 Vous avez dit «microcontrôleur» ?

Jusqu'à présent, vous avez surtout entendu parler des composants matériels constituant les ordinateurs, que sont les processeurs (en anglais CPU pour *Central Processing Unit*), les composants mémoire, les contrôleurs de périphériques, etc. Dans le TP Micromachine en particulier, vous avez appris à concevoir un processeur, constitué d'une UAL, de registres, etc. Le processeur vous a été présenté comme un composant complet, implanté comme une puce électronique, et qui interagit avec d'autres composants, eux-mêmes implantés chacun sur une puce différente, le tout communiquant par le biais d'un bus.

La tendance à la miniaturisation a poussé les fabricants de matériel à implanter plusieurs de ces composants sur la même puce de silicium. C'est le cas des mémoires cache des processeurs d'ordinateur, qui depuis longtemps sont implantées directement sur la même puce que le processeur lui-même. L'avantage principal est l'efficacité accrue des communications, ainsi que la diminution de la taille du système. Cette tendance est actuellement poussée à l'extrême avec l'apparition depuis une dizaine d'années des *Systems-on-a-Chip* ou SoC, qui contiennent sur la même puce à la fois des unités de calcul type CPU, mais aussi des composants de conversion analogique/numérique ou numérique/analogique, des mémoires de masse, des composants de communication radio, etc.

Le terme *microcontrôleur* est couramment utilisé pour désigner une forme d'architecture SoC dont les composants sont fournis et intégrés par le même fabricant. Leur taille est considérablement réduite par rapport à un système équivalent en fonctionnalités mais construit selon les techniques traditionnelles. Leur coût est également réduit (quelques centimes seulement !). On en trouve ainsi partout dans l'informatique embarquée «quotidienne» : téléphonie, adsl-box, montres, etc.

## 1.5 La famille MSP430 et le MSP430-FG4618

Les microcontrôleurs MSP430 sont produits par la société Texas Instruments depuis les années 90. Ils ont connu (et connaissent toujours) un succès important grâce notamment à leur faible consommation d'énergie, et à leur facilité de programmation.
Le microcontrôleur qui va nous intéresser durant les TP d'architecture est le MSP430-FG4618. Son architecture interne est décrite sur la figure 2. Similaire à celle d'un système classique (à plusieurs composants), elle s'organise autour d'un bus principal reliant l'ensemble des périphériques internes du MSP430 : processeur, mémoires, timers...

## 1.6 Le processeur du MSP430-FG4618

Comme vous pouvez le voir sur la figure 2, le cœur de calcul du MSP430-FG4618 est un CPU pouvant fonctionner jusqu'à à 8 MHz et possédant 16 registres.

**Exercice 2 —** Dans le schéma, il y a deux lignes dénotées respectivement MAB et MDB. Expliquez de quoi il s'agit en quelques phrases.

Réponse :

**Exercice 3 —** Jeu des sept erreurs . . . Identifiez sept points communs entre le MSP430-FG4618 et la micromachine. Identifiez ensuite sept différences. Justifiez rapidement vos choix.

Réponse :

- Similarité 1 :
- Similarité 2 :
- Similarité 3 :
- Similarité 4 :
- Similarité 5 :
- Similarité 6 :
- Similarité 7 :
- Différence 1 :
- Différence 2 :
- Différence 3 :
- Différence 4 :
- Différence 5 :
- Différence 6 :
- Différence 7 :

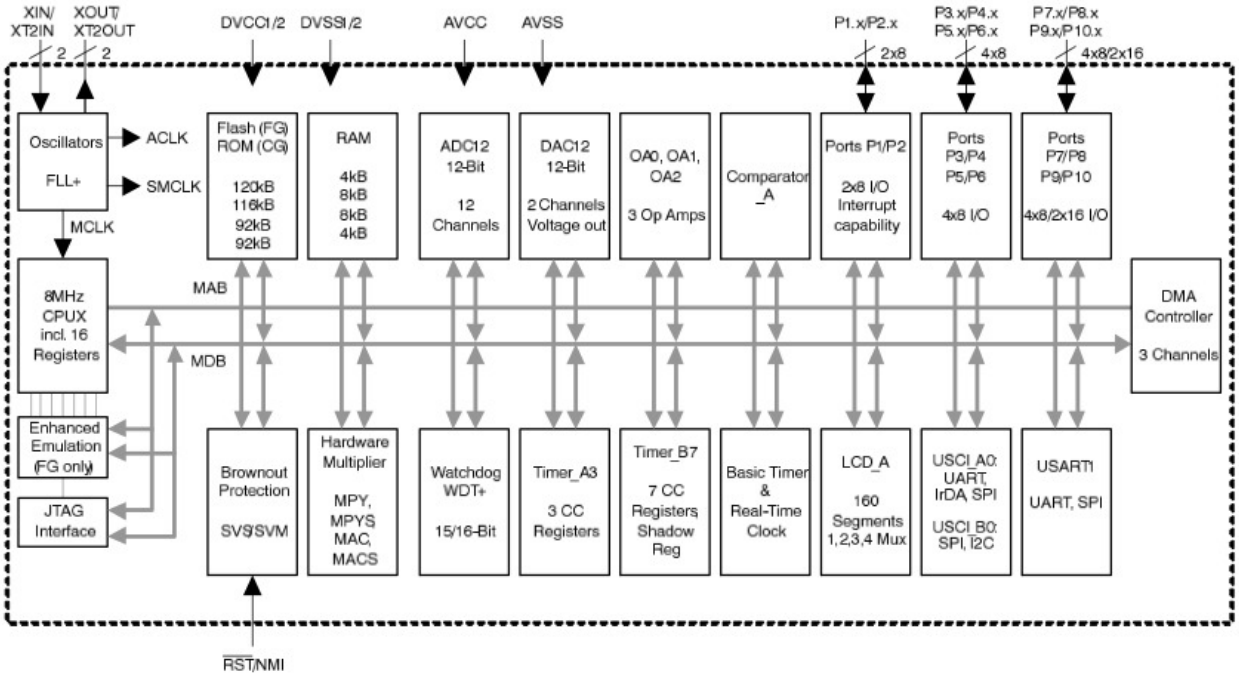


FIGURE 2: Architecture interne du MSP430-FG4618

**16 bits ou 20 bits ?** Il ne vous aura pas échappé que la documentation du microcontrôleur [MSP430.pdf, chap. 1] parle de CPU 16 bits, mais que notre MSP430 dispose d'un espace d'adressage plus vaste que la limite théorique des 64 Ko (16 bits = $2^{16}$ octets adressables). Comment fait-on donc pour accéder aux données situées au-delà de la limite ?

Pour les plus curieux d'entre vous, la réponse à cette question se trouve dans le chapitre 4 de la doc [MSP430.pdf]. Pour les autres, et cette réponse nous suffira pour l'ensemble des TP d'architecture, nous allons nous limiter aux premiers 64 Ko de l'espace d'adressage, avec des adresses sur 16 bits(de 0000h à FFFFh) et non pas sur 20 bits (de 00000h à FFFFFh). Nous allons donc adopter comme cartographie mémoire du MSP430 la vue simplifiée présentée sur la figure 3 ci-dessous, au lieu de celle donnée dans la documentation [MSP430.pdf, fig. 1-2].

# 2 Prise en main de *IAR Embedded Workbench*

Le logiciel *IAR Embedded Workbench* est un environnement de développement intégré (IDE), incluant un éditeur de code [workbench.pdf], et une chaîne de compilation complète : compilateur [compiler.pdf, libc.pdf], assembleur [assembler.pdf], éditeur de liens (*linker*) [linker.pdf], et *debugger* [workbench.pdf, chap. 4]. Durant ce TP, nous allons utiliser différents éléments de cet environnement. Vous trouverez les documentations correspondantes sur le Moodle.

Lancer IAR Workbench depuis le menu *Démarrer* > *Tous les programmes* > *Développement* > *IAR Systems* > *IAR Embedded Workbench Kickstart for MSP430 V4* > *IAR Embedded Workbench*.

En vous aidant des différentes documentations mises à disposition sur Moodle, créez un nouveau *worskpace*, puis un projet nommé par exemple *TP1*. Voir notamment la documentation de l'IDE [workbench.pdf, pages 25 et suivantes], et l'annexe A de [Motherboard.pdf]. Veillez en particulier à ce que votre projet soit bien configuré pour compiler du code à destination du MSP430-FG4618 et que le débuggueur soit bien configuré pour utiliser les sondes JTAG (et non en mode simulation).

## 2.1 Un premier programme

Ajouter à ce projet un premier fichier `.c`, avec le contenu suivant. Des explications sur ce que font les différentes parties du programme sont données plus bas.

```
#include "msp430.h"

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
  P5DIR = 0x02; // Configure P5.1

  for (;;) {
    volatile unsigned int i; // volatile to prevent optimization

    P5OUT = P5OUT ^ 0x02; // Toggle P5.1

    for (i=0 ; i<10000 ; i++); // Software Delay
  }
}
```

## 2.2 Quelques explications

**Watchdog** Comme de nombreux processeurs embarqués, le MSP430 possède un *watchdog* matériel [MSP430.pdf, chap. 12]. Le rôle de ce composant est de «surveiller», de l'extérieur, le bon fonctionnement du logiciel, afin d'éviter que le système ne se bloque, par exemple dans une boucle infinie.

Le principe de cette surveillance est assez simpliste : le watchdog est un simple *timer* (i.e. un compteur qui avance à chaque cycle d'horloge) qui redémarre l'ensemble du système lorsqu'il atteint une valeur plafond prédéfinie. Le logiciel doit donc régulièrement signaler qu'il fonctionne correctement, en réinitialisant le watchdog à zéro !

Pour ne pas être gêné par des *reset* intempestifs, et ne pas avoir à se compliquer la vie avec le watchdog, notre programme commence donc par le désactiver.

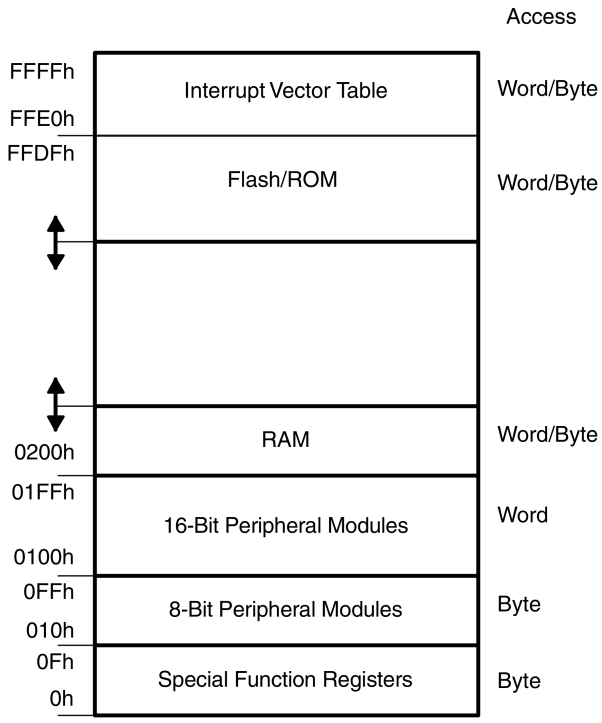


FIGURE 3: Espace d'adressage du MSP430

**Exercice 4 —** Dans notre programme, si on ne prenait pas de précaution particulière vis-à-vis du watchdog, au bout de combien de temps (en secondes) est-ce que celui-ci redémarrerait le système ?

Réponse :

**General-Purpose Input/Output** Notre microcontrôleur possède plusieurs interfaces avec le monde extérieur : des convertisseurs analogique/numérique en entrée, et numérique/analogique en sortie, un port série, plusieurs bus spécialisés ($I^2C$, SPI), etc. La plus simple de ces interfaces est appelée GPIO (pour «General-Purpose Input/Output» [MSP430.pdf, chapitre 11]) et consiste en une série de broches sur lesquelles on peut lire ou écrire un signal booléen depuis le logiciel.

Vus depuis le programme, les ports GPIO se présentent sous la forme de registres projetés en mémoire. Chaque bit de chaque port peut être configuré séparément, comme expliqué dans la documentation [MSP430.pdf, chap. 11].

**Exercice 5 —** Expliquez la signification du terme "projeté" dams le paragraphe ci-dessus. En quoi cette organisation du microcontrôlleur diffère-t-elle de celle de la micromachine ?

Réponse :

**Exercice 6 —** Combien y a-t-il de ports GPIO sur notre microcontrôleur, et combien comportent-ils chacun de bits ?

Réponse :

Pour économiser sur la taille de la puce, la plupart des broches qui connectent le MSP430 au mondre extérieur sont *mutualisées* entre plusieurs périphériques internes.
Ce partage est documenté sur la *datasheet* du processeur [datasheet.pdf, pages 3 et suivantes].

**Exercice 7 —** Notre programme utilise un seul bit d'entrée-sortie, sur le port 5.1 du microcontrôleur. Quelle est la broche correspondante sur la puce, et avec quels autres périphériques internes cette broche est-elle partagée ?

Réponse :

**Exercice 8 —** La ligne `P5DIR = 0x02` de notre programme modifie le registre matériel P5DIR. Dans quelle confirguration se retrouve le port P5 après l'exécution de cette ligne ?

Réponse :

**Exercice 9 —** La ligne `P5OUT = P5OUT ^ 0x02` de notre programme change la valeur du registre matériel P5OUT à chaque tour de boucle. Quel est l'effet obtenu sur le port P5 ? Quel est le résultat que vous observez ? En vous aidant de la la documentation de la carte [Motherboard.pdf], notamment des connexions qui relient le MSP430 à son environnement direct, expliquez ce résultat.

Réponse :

**Volatile** Bien que souvent considéré comme un «langage de bas niveau», le langage C est bien évidemment beaucoup plus abstrait que le langage machine. Le compilateur a donc une marge de manœuvre assez importante dans la façon dont il traduit le programme vers une forme exécutable. En particulier, il va chercher à *optimiser* les performances du programme en supprimant les opérations inutiles.

Par exemple, dans notre cas, la boucle sur `i` n'a pas d'intérêt apparent pour la logique du programme, et donc le compilateur sera tenté de la supprimer complètement. Pour s'assurer que la boucle est bien

conservée, on a utilisé ici le mot-clé `volatile`, qui *interdit* au compilateur d'optimiser les accès à `i`. De cette façon, on peut être certain que notre boucle de temporisation sera bien exécutée, quelles que soient les optimisations appliquées sur le reste du programme.

### 2.3 Exécution du programme

Brancher la sonde JTAG sur la carte (voir la documentation de la carte [Motherboard.pdf, appendice A]) et transférer le programme sur la cible en cliquant sur *Project > Download and Debug*.

Par défaut, l'exécution s'arrête sur un point d'arrêt à l'entrée de la fonction `main()`. Reprendre l'exécution en cliquant sur *Debug > Go*.

## 3 Prise en main du debugger

On va maintenant s'intéresser un peu plus en détail à l'exécution de notre programme. Pour cela, arrêter le debugger (menu *Debug > Stop Debugging*) puis, dans les options du projet (*Project > Options*), afficher la rubrique «Debugger», et décocher la case «run to main» (voir figure 3).

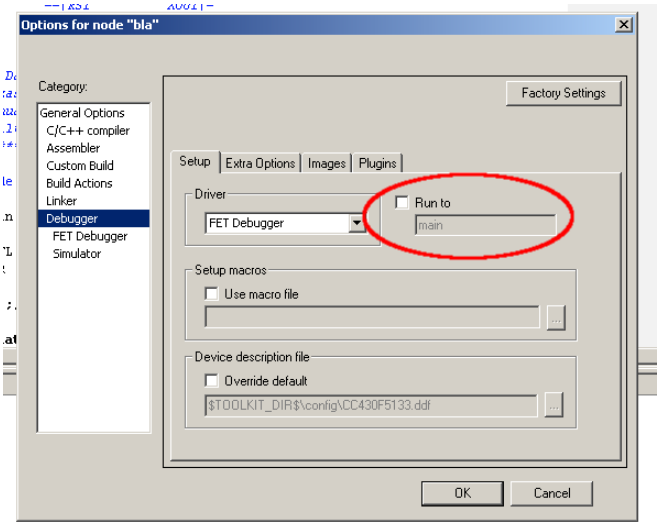


FIGURE 4: Fenêtre d'options du projet

Relancer l'exécution (*Project > Download and debug*). Comme on le lui a demandé, le programme ne démarre plus mais reste suspendu avant la première instruction. On ne peut pas observer à l'aide du debugger ce qui se passe *avant* ce stade, mais il est intéressant de s'y arrêter un instant. Le démarrage du microcontrôleur est décrit dans le chapitre 2 de la spécification [MSP430.pdf]. Lors de la mise sous tension, le microcontrôleur initialise ses différents composants périphériques (horloges...) puis démarre le processeur.

**Exercice 10 —** Quelle est l'adresse de la première instruction du programme ? Comment le MSP430 trouve-t-il cette adresse ?

Réponse :

**Exercice 11 —** Comment est stockée la variable `i` lors de l'exécution ? pourquoi ?

Réponse :

**Exercice 12 —** Quelles instructions assembleur correspondent à la boucle de temporisation ? Calculez à quelle fréquence clignote la LED. Les durées d'exécution des différentes instructions sont indiquées dans la doc du processeur [CPU.pdf, partie 3.4.4].

Réponse :

# 4 Votre premier programme

Allumer ou éteindre une LED permet de communiquer (très sommairement !) avec l'utilisateur, mais on manque encore d'un moyen de communication dans l'autre sens, c'est à dire de l'utilisateur vers programme. Notre carte possède plusieurs composants destinés à cet effet, donc le plus simple est une paire de boutons poussoirs, SW1 et SW2, reliés à d'autres entrées numériques généralistes (GPIO) du microcontrôleur.

**Exercice 13 —** À quelles pattes du MSP430 les deux boutons sont-ils connectés ? (comme d'habitude, c'est moins la réponse proprement dite qui est intéressante, mais l'endroit dans la documentation où vous l'avez trouvée).

Réponse :

**Exercice 14 —** On veut pouvoir lire l'état du bouton SW1 depuis le programme. Que faut-il rajouter dans votre `main()` pour configurer convenablement les ports GPIO ?

Réponse :

**Exercice 15 —** Écrire ensuite une boucle infinie qui réagisse à l'appui sur un bouton :

```
for(;;)
{
  while( /* button is not pressed */ ) ; // do nothing
  // button is pressed

  // do action
}
```

Comme action, faire par exemple clignoter la diode, ou encore mieux, incrémenter une variable que vous observerez grâce au debugger. Vous remarquerez alors qu'une seule pression sur le bouton provoque de nombreux tours de la boucle `for(;;)`. La raison est simple : après l'action, le programme revient au `while`, constate que le bouton est encore appuyé, repart pour un tour, et ainsi de suite.

**Exercice 16 —** Modifier donc le programme pour réagir au *relâchement* du bouton plutôt qu'à l'appui proprement dit :

```
for(;;)
{
  while( /* button is not pressed */ ) ; // do nothing
  while( /* button is pressed */ ) ;     // do nothing
  // button has just been released

  // do action
}
```

Réponse :