TP - Robotique

M. GEORGES-SAINT-MARC - NSI - Lycée Mendès France Rennes

27 novembre 2020

Ce TP est pour partie inspiré des séances proposées lors du DIU de NSI de Rennes par Matthieu Simonin et Guillermo Andrade Barroso, de l'INRIA (Université de Rennes 1).

1 Microcontrôleur

Un **microcontrôleur** (*MCU* en anglais : *microcontroller unit*) est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires, unités périphériques et interfaces d'entrées-sorties.

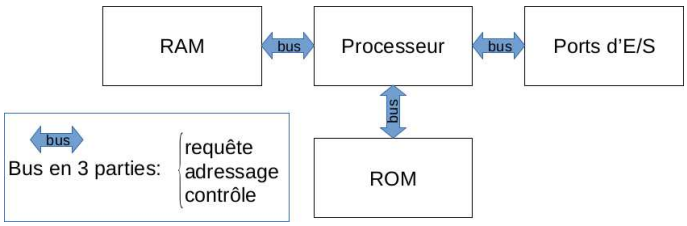
On les retrouve beaucoup dans les **systèmes embarqués** (téléphones portables, horloges, GPS, électroménager, automobile, . . .).

1.1 Microprocesseur et microcontrôleur

Un (micro)processeur est le composant central d'un ordinateur : il interprète les instructions et traite les données d'un programme.

Pour fonctionner, il a besoin d'éléments extérieurs :

- mémoires vive / morte / de masse ;
- horloge (pour le cadencer) ;
- des périphériques (pour interagir avec le monde extérieur).



La connexion entre ces différentes unités est assurée par des **bus** (des dispositifs de transmissions de données).

Un ordinateur de bureau contient ces bus sur une **carte mère**.

L'intérêt d'un microcontrôleur est d'intégrer une partie de ces composants à l'intérieur d'un circuit intégré, afin de gagner en :

1. espace (place occupée par les composants et les connexions : bus sur la carte mère, câbles et nappes de connexion) ;
2. énergie (de consommation électrique) ;
3. dégagement de chaleur ;
4. coût.

Un microcontrôleur est rattaché à une tâche bien particulière, et n'a pas besoin que le ou les programmes qu'il interprète changent très régulièrement. Ainsi, ces programmes ne sont pas installés sur une mémoire **de masse** (disque dur HDD, disque SSD) mais sur une mémoire **morte** (ROM : *Read-Only Memory*, qui n'a vocation qu'à être lue et non modifiée).

Le microcontrôleur réunit donc :

1. un processeur (dont les capacités de calculs sont plus limitées ou spécifiques qu'un microprocesseur d'ordinateur de bureau)
2. des mémoires morte (ROM) et vive (RAM)
3. une horloge (un oscillateur)

4. quelques périphériques permettant certaines tâches précises (calculs, timers, comparateurs, contrôleurs de bus de communication, . . .)

Le choix de ces composants est bien sûr décidé par le fabricant selon les objectifs visés par ce microcontrôleur et les coûts engendrés par sa gravure.

2 Carte Micro :Bit

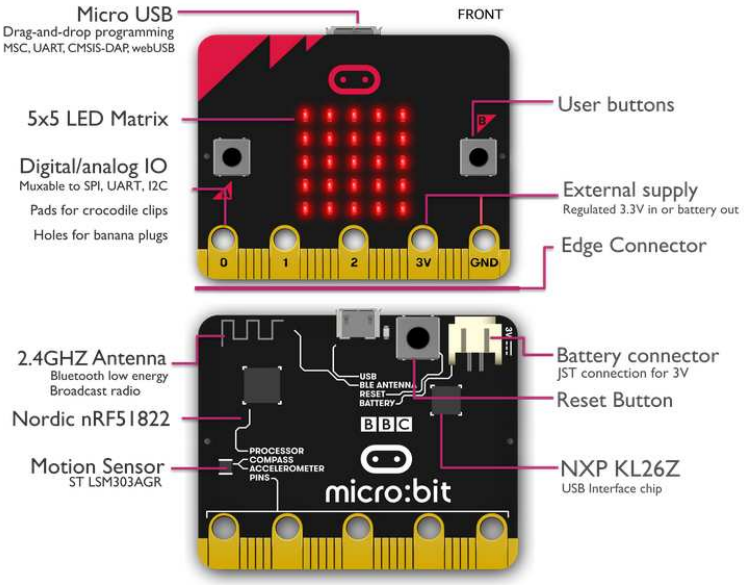
Nous travaillons dans ce TP sur un nano-ordinateur qui comprend deux microcontrôleurs et une carte unique sur laquelle sont montés les composants.
Il s'agit de la carte Micro :Bit, développée depuis 2015 par la BBC à destination des écoliers anglais.

2.1 Spécifications et fonctionnement basique

Le microcontrôleur principal est un Nordic Semiconductor nRF51822 cadensé à 16 MHz, en 32 bits, à processeur ARM Cortex-M0, avec 256 ko de mémoire de stockage flash (ROM) et 16 ko de mémoire vive (RAM).

Pour programmer la carte :

1. on écrit du code (ici on le fera en **micro-python**)
2. on compile ce code (conversion en ficher hexa) et on le transfert sur le microcontrôleur : on **flashe** la mémoire (ROM).
3. on lance le composant pour exécution.



2.2 Éditeur de micro-python

Pour le codage et l'écriture dans la ROM, nous passerons par un éditeur spécifique de Python nommé **mu-editor** (mu pour la lettre grecque μ symbole de micro). Pour le lancer, passer par un terminal sous linux et taper `mu-editor`.

Boutons centraux de l'éditeur :

- Flasher : pour écrire le programme dans la ROM de la Micro :Bit
- Fichiers : pour vérifier quels fichiers sont sur la ROM
- REPL : pour lancer des commandes en mode console à la carte sous tension

2.3 Activités simples

Exercice 1 : *Premières manipulations*

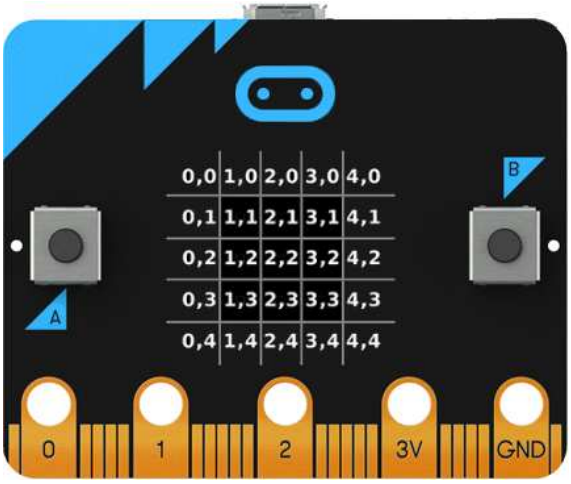
1. Connecter la carte Micro :bit à l'ordinateur via le câble USB, puis lancer `mu-editor`.
2. Pour commencer, nous allons manipuler la carte via la console : cliquer sur le bouton REPL (*Read Evaluate Print Loop*).
   → **Toutes les commandes de cet exercice seront saisies dans la console python de mu-editor.**
3. Afin de charger la bibliothèque adéquate en python, commencer par saisir la commande

```
from microbit import *
```

4. Afficher un message lettre par lettre avec la commande `display.show("Voici un test")`.
5. Tester la différence avec `display.scroll("Ceci est du scrolling")`.
6. **a)** Taper `display.` (`display` suivi d'un **point**), puis appuyer sur la touche Tab pour voir les différentes méthodes liées à `display`.
   b) Sélectionner la méthode permettant d'effacer la lettre à l'écran (cela éteint l'ensemble des LEDs).
7. Les 25 LEDs de la Micro :Bit ont chacune des coordonnées dans un repère (*cf* ci-contre).
   a) Voir le détail de la commande `display.set_pixel` avec la fonction :

```
help(display.set_pixel).
```

   b) Colorer les LEDs situées aux quatre extrémités en testant différentes intensités lumineuses pour chaque coin.
   c) Effacer puis créer une double boucle qui colorie les lignes 1 et 2 en faisant progressivement passer l'intensité lumineuse de 0 à 9.



Exercice 2 : *Boutons*

Nous allons désormais laisser de côté le mode console et allons flasher la carte.
Pour cela, écrivons dans la partie éditeur du programme.
Tout programme commencera par l'importation de la bibliothèque **microbit** :

```
from microbit import *
```

1. La manipulation des boutons A et B de la carte se fait à l'aide des commandes `button_a` et `button_b`.
   Compléter le programme suivant afin qu'il affiche le message "A" si le bouton A est pressé, et rien sinon :

```
from microbit import *

while True: # Cette boucle infinie sert à réaliser le
    test suivant en continu
    if button_a.is_pressed():
        ........................
    else:
        ........................
```

→ Pour flasher le programme sur la carte, il suffit d'appuyer sur le bouton correspondant dans mu-editor. La LED orange située au verso, près du bouton Reset, clignote jusqu'à ce que le fichier ait totalement été écrit dans la ROM du microcontrôleur.

2. Faire un programme qui affiche trois images distinctes selon que A, B ou les deux boutons soient simultanément pressés.
   Remarque : pour les images, on pourra choisir d'utiliser l'objet Image qui contient plein de figures prédéfinies, comme par exemple :

   `display.show(Image.HEART)`

   ou on pourra aussi les décrire pixel par pixel sous une forme condensée, ligne par ligne, selon l'intensité :

   `display.show(Image('00300:' '03630:' '36963:' '03630:' '00300'))`

3. **a)** Créer un programme qui affiche un cœur qui bat.
   Pour cela, faire une boucle infinie qui alterne toutes les demi-secondes l'affichage de l'image entre HEART et HEART_SMALL.
   Indication : on pourra se servir de la fonction `sleep(`n`)` qui permet de patienter n millisecondes avant la prochaine instruction.

   b) Améliorer le programme en faisant en sorte que le bouton A ralentisse la fréquence de battement et que le bouton B accélère la fréquence de battement.

Exercice 3 : *Température*

Cet exercice vise à utiliser le capteur de température de la carte. Ce capteur mesure la température du microcontroleur principal (désigné dans la nomenclature en page 2.1 par PROCESSOR). Ce type de composant chauffe assez peu et a donc tendance à refléter la température extérieure.

1. Créer une boucle infinie (c'est la dernière fois qu'on le signale explicitement : ce sera toujours le cas par défaut) qui affiche la température toutes les 3 secondes.
   Indication : la commande qui récupère la température sous forme de nombre entier de degrés Celsius est `temperature()` . Attention, c'est un nombre, donc pour l'afficher il faut le convertir en `string` avec `str(`$nombre$`)` .
2. recouvrir doucement de son doigt le microcontrôleur central afin de réchauffer le processeur et vérifier que cela a bien un impact après quelques secondes.
3. Créer un programme qui affiche un icône de satisfaction (HAPPY) ou de tristesse (SAD) selon que la température dépasse 20°C ou non. On testera la carte en extérieur (si la pluie n'est pas au rendez-vous !) à l'aide des batteries.

Exercice 4 : *Compas*

1. Afficher la valeur du compas lorsqu'on appuie sur le bouton A à l'aide du programme suivant :

```
1 from microbit import *
2
3 # Calibrage indispensable au départ
4 compass.calibrate()
5
6 while True:
7     if button_a.is_pressed():
8         display.show(str(compass.heading()))
9     display.clear()
```

   Indication : pour la phase de calibrage, il vous faudra incliner la carte jusqu'à colorer l'ensemble des 25 LEDs rouges du test.
2. Que renvoie la valeur ? à quoi correspond le 0 ?
3. Créer une boussole : faire en sorte qu'au lieu d'un nombre, l'image affichée soit un trait qui indique le nord.
   Indication : on pourra utiliser l'image de l'aiguille de l'horloge : `display.show(Image.ALL_CLOCKS[`$heure$`])` (où $heure$ est un nombre entre 0 et 11)

Exercice 5 : *ChiFouMi... again ? !*

L'objectif est ici de transformer sa carte Micro :Bit en plateforme pour ChiFoumi, afin d'affronter un adversaire. Il s'agit surtout d'introduire l'accéléromètre de la carte.
Les règles sont les suivantes :

- Chaque joueur presse A (Pierre), ou B (Feuille), ou les deux boutons simultanément (Ciseaux) sans le montrer à l'autre.
- Une fois ce délai passé, les joueurs secouent la carte et celle-ci affiche (au choix selon l'envie du programmeur) un nombre, une lettre ou un symbole correspondant à Pierre, Feuille ou Ciseaux.
- Chaque partie est réinitialisée avec le bouton Reset.

Indications :

- pour les boutons, utiliser plutôt `button_x.was_pressed()`.
- pour la mesure d'une secousse, on utilise l'accéléromètre, qui est capable de classer certains gestes, notamment :

```
accelerometer.is_gesture('shake')
```

3 Robot MaQueen

Il est temps de prendre les commandes d'un robot !

Le Robot MaQueen, développé par le fabricant chinois Zhiwei Robotics Corp (exporté en europe et aux USA sous le nom DFRobot), est une extension à la carte Micro :Bit, qui vient le piloter.

Pour le manipuler, nous nous servirons ici d'une classe Maqueen développée par la BBC et adaptée au français par Olivier Lecluse, enseignant au lycée Allende d'Hérouville Saint Clair.

Exercice 6 : *Premiers ~~pas~~ tours de roues !*

1. Si le robot n'est pas encore monté... le monter ! Bien voir avec l'enseignant sur l'étape de collage du compartiment des piles.
2. Placer (s'il n'y est pas encore) le fichier *maqueen.py* dans le dossier `/home/eleve/mu_code`.
   Dans l'éditeur mu, cliquer sur le bouton Fichiers et transférer sur la carte le fichier *maqueen.py* .
   Ce fichier servira lors de l'écriture des programmes dans la ROM.
   Une fois le transfert fait, cliquer à nouveau sur le bouton Fichiers.
3. **a)** Commencer par coder le programme suivant :

```
from microbit import *
from maqueen import Maqueen

mq = Maqueen()

mq.avance(50) # 50 : pourcentage de la vitesse maximale
sleep(2000)
mq.stop()
```

b) Le flasher sur la carte. Il est normal qu'un message d'erreur apparaisse.

c) Vérifier que le robot est bien éteint, puis brancher la Micro :Bit dessus (attention, les LEDs sont tournées vers l'avant du robot, à l'opposé du compartiment des piles).

d) Placer le robot à plat avec environ 50 cm de marge devant lui puis l'allumer. Que se passe-t-il ?

4. Tester le programme suivant :

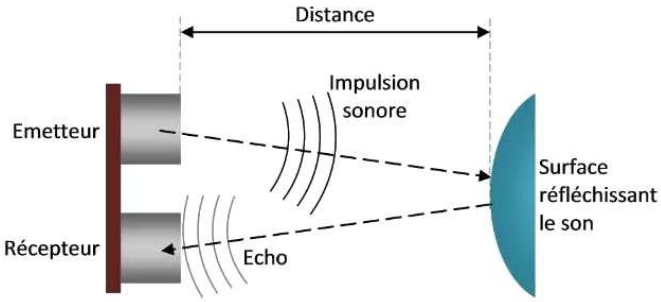
```
from microbit import *
from maqueen import Maqueen

mq = Maqueen()

mq.moteurDroit(60)
mq.moteurGauche(30)
sleep(2000)
mq.stop()
```

Que se passe-t-il désormais ?

5. Programmer une série de trois virages (Gauche puis Droite puis Gauche) durant 4 secondes chacun, en faisant clignoter à chaque fois la LED P8 ou P12 correspondante et en augmentant le braquage à chaque virage.
Attention à bien prendre de la place au sol pour les tests !
Indication : pour allumer ou éteindre la LED P8, faire `pin8.write_digital(1)` .

Exercice 7 : *Obstacles*

On peut utiliser le capteur à ultrasons pour détecter des obstacles devant le robot.
Pour cela, il suffit d'appeler la méthode `distance()` de la classe Maqueen, qui renvoie la distance (en *cm*) à un obstacle à un instant *t*.



1. Écrire un programme qui :

- affiche par défaut un smiley heureux ;
- fait rouler par défaut le moteur à 30 % de la vitesse maximale ;
- affiche un smiley surpris (SURPRISED) si la distance à un obstacle passe sous les 20 *cm* et ralentit alors le robot à 10 % de sa vitesse maximale.
- affiche un smiley triste et arrête définitivement le robot si la distance à un obstacle passe sous les 10 *cm*.

2. Écrire programme qui améliore le précédent, en faisant faire au robot un quart de tour vers la gauche si l'obstacle passe sous les 10 *cm* de distance (on fera faire une marche arrière puis une marche avant au robot comme dans le dessin ci-contre.