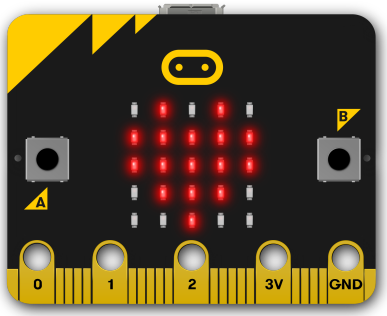
Vincent Le Mieux

Etendre les fonctionnalités d'un microcontrôleur



00000010

python powered

Table des matières

1 Introduction :....3
2 Retour sur la carte Micro:Bit....5
2.1 Le microcontrôleur Nordic NRF51822....5
2.1.1. Schéma de mise en oeuvre....5
2.1.2. Spécifications et valeurs limites d'utilisation....8
2.2 Alimentation du montage....11
2.2.1. Les trois façons d'alimenter le microcontrôleur....11
2.2.2. Alimentation et compatibilité du GPIO....14
2.3 Le connecteur d'extension :....18
2.3.1. Vue globale du connecteur....18
2.3.2. Les différentes façons de s'y connecter....19
2.4 Accéder aux entrées/sorties avec Python....24
2.4.1. Le problème des broches partagées....24
2.4.2. Entrées/sorties numériques....27
2.4.3. Entrée analogique....31
2.4.4. Sortie PWM (Pulse Width Modulation)....39
2.4.5. Composant I2C (Inter Integrated Circuit)....39
2.4.6. Composant SPI (Serial Peripheral Interface)....39
3 Utiliser le système Grove....40
3.1 Connectique des modules Grove....40
3.2 Le module Led....42
3.2.1. Présentation....42
3.2.2. Mise en œuvre :....44
3.2.3. Utiliser l'aliasing....44
3.2.4. Fonctions et module de fonctions....46
3.3 Le module bouton poussoir....47
3.3.1. Présentation et mise en oeuvre....47
3.3.2. Fonctions et module de fonctions....49
4 Solutions aux exercices....50
5 Annexes....60

Typographie utilisée dans ce manuel :

Ce manuel a été intégralement composé avec le logiciel Writer de la suite bureautique LibreOffice. Les logiciels Gimp et Inkscape ont permis la gestion de certaines images. Merci à tous ces développeurs !

On y trouvera :

- du code écrit ainsi :

```
from microbit import *
i = 0
while i < 5 :
    display.set_pixel(i, 2, 9)    # x va varier de 0 à 4 ; y fixé à 2
    i = i + 1
```

- ***Des remarques écrites en violet avec une police mise en italique***
- **Des exercices dont les solutions sont fournies à la fin de ce manuel. On reconnaîtra les exercices à cette typographie (en gras et bleu).**

Dans les solutions, les parties de code à repérer tout particulièrement sont « fluotées » en jaune :

```
display.set_pixel(2,1,9)
display.set_pixel(2,2,9)
```

Pour aller plus loin, des remarques sont parfois ajoutées dans les corrections : il est important d'aller les voir.

Repérage : un exercice noté Exercice 4.3.2_2 signifie qu'il s'agit du deuxième exercice du paragraphe 4.3.2. Le corrigé reprend la même numérotation.

Quelques schémas trop grands pour la pagination A5 du manuel ont été mis en Annexe au format A4. Il pourra être judicieux d'en faire un tiré à part pour pouvoir les consulter facilement tout en lisant le texte d'accompagnement.

1 Introduction :

Ce manuel fait suite au premier ouvrage intitulé *« Découvrir le langage Python en programmant un microcontrôleur »* qu'il faudra avoir lu et expérimenté avant d'aborder celui-ci.

On pourra par contre, et ce sera même bienvenu, avancer en parallèle dans ce deuxième ouvrage, avec la lecture du troisième tome : « Utiliser un microcontrôleur en Sciences Physiques ».

Dans le premier volume, nous n'avions volontairement que travaillé avec une partie des ressources disponibles sur la carte BBC Microbit. Si cela a pu être suffisant pour découvrir en même temps le langage Python, le lecteur a pu se trouver limité dans des idées d'applications personnelles qui pouvaient lui venir à l'esprit parce que :

- le nombre de boutons poussoirs présents (2) est insuffisant

- l'afficheur a une résolution très faible, ce qui entraîne des images de piètre qualité, ou nécessite de faire défiler chiffre par chiffre une valeur numérique

- les capteurs disponibles ne répondent pas à l'application visée :

    - le capteur température intégré au microcontrôleur ne permet que la mesure de la température de l'air immédiatement au contact du microcontrôleur
    - le capteur de luminosité est trop grand et pixellisé (les 25 Leds de l'afficheur), de plus nous n'avons aucune information sur la fonction de transfert de ce capteur : qu'en est-il de sa dynamique ou de sa linéarité ?
    - on souhaite mesurer d'autres grandeurs physico-chimiques...

Tout cela nous amène à vouloir étendre les fonctionnalités de cette carte (d'où le titre de ce deuxième tome), et donc à apprendre à utiliser le connecteur d'extension qui est en lien direct avec le microcontrôleur.

De nombreuses carte de développement ont vu le jour depuis l'arrivée du système Arduino, avec la volonté affichée de rendre simple et transparente

la réalisation d'une chaîne électronique programmable… Il est vrai que l'utilisation de modules additionnels que l'on a juste à clipser sur la carte, sans devoir passer par les étapes de type gravure de circuit imprimé et soudure de composants fait gagner un temps précieux et se trouve être d'une grande efficacité dans une phase d'apprentissage.

Par contre, c'est une erreur de croire que l'on peut s'engager sur cette voie sans connaître les caractéristiques du microcontrôleur utilisé, ses limites physiques ou celles imposées par le langage choisi. Cela ne peut qu'aboutir à des déconvenues dès lors que l'on tombera sur un comportement non prévu du système, ou que l'on voudra s'écarter un peu des chemins battus.

Par ailleurs, en tant qu'enseignant, il est nécessaire d'avoir une vision à la fois globale mais aussi précise du système pour être le plus à l'aise possible devant l'apprenant.

C'est là l'objectif de cet ouvrage.

Remarque : le titre de cet ouvrage « Etendre les fonctionnalités d'un microcontrôleur » pourra paraître trompeur vis à vis du contenu qui se restreint en fait à la seule carte BBC Micro:Bit… Le lecteur qui dispose d'une autre carte de développement, pourra mettre en application toutes les recommandations qui sont faites ici au niveau électrique, en allant chercher les informations équivalentes dans les fichiers de données constructeurs (les « datasheets » des composants utilisés)

2 Retour sur la carte Micro:Bit

2.1 Le microcontrôleur Nordic NRF51822

2.1.1. Schéma de mise en oeuvre

Le schéma 1, fourni en Annexe, ne représente qu'une partie de la carte BBC Micro:Bit, celle qui concerne le microcontrôleur que l'on utilisera (en fait il y a un deuxième microcontrôleur sur la carte, destiné à réaliser l'interface entre l'USB et celui-ci). Sur ce schéma nous avons entouré les principaux groupes fonctionnels. (En fait pas grand-chose comme c'est souvent le cas avec les microcontrôleurs, l'essentiel étant à l'intérieur du composant). On y trouve classiquement :

- Du découplage d'alimentation à l'aide de petits condensateurs destinés à éliminer les parasites transitoires sur les lignes d'alimentation. Celles-ci ne sont sensées véhiculer que de la tension continue (déjà filtrée et stabilisée en amont), mais entre le module d'alimentation et le microcontrôleur, il est possible que les lignes de cuivre du circuit imprimé récupèrent des perturbations électromagnétiques (notamment dans ces environnements numériques ou l'on réalise des commutations de niveaux à haute fréquence). Une petite capacité se comporte quasiment comme un court-circuit vis-à-vis des hautes fréquences, ce qui évacue ces parasites transitoires vers la masse. C'est l'occasion de rappeller que l'impédance d'un condensateur de capacité C est donnée par la relation :

$$Z = \frac{1}{2 \cdot \pi \cdot f \cdot C}$$

il faut donc que C soit petite pour que Z tende vers 0 quand la fréquence f de ces parasites est grande).

- un oscillateur à quartz de fréquence 16MHz. La fréquence d'horloge est une caractéristique importante car la vitesse d'exécution d'une instruction peut y être directement liée. Comme cela a été évoqué au paragraphe 2.2 Langage de programmation

dans le manuel « Découvrir le langage Python en programmant un microcontrôleur », le processeur ne comprend in fine que des codes machines, représentés plus simplement par des mnémoniques lorsque l'on travaille en langage d'assemblage. Voici un extrait de la table décrivant toutes les instructions machines du Cortex M0 (le coeur du microcontroleur utilisé) :

Table 3-1 Cortex-M0 instruction summary

Operation	Description	Assembler	Cycles
Move	8-bit immediate	`MOVS Rd, #<imm>`	1
	Lo to Lo	`MOVS Rd, Rm`	1
	Any to Any	`MOV Rd, Rm`	1
	Any to PC	`MOV PC, Rm`	3
Add	3-bit immediate	`ADDS Rd, Rn, #<imm>`	1

On y donne pour chacune d'entre elles le nombre de cycles machine nécessaires à l'exécution de ces instructions de base. Les plus simples ne consommeront qu'un seul cycle machine(soit une durée de $1/16.10^6$ = 62,5 ns) , d'autres en consommeront plusieurs.

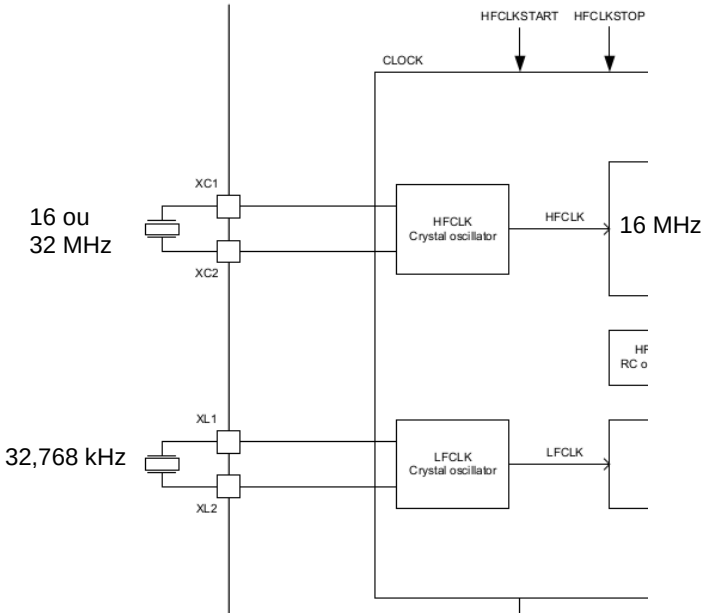
Remarques :

- c'est sur la connaissance de ces timings d'exécution que l'on peut régler finement la durée d'exécution d'une portion de programme en assembleur.

- la fréquence du quartz ne peut pas être choisie comme on le veut : le fabricant impose parfois une valeur ou une gamme de valeurs possibles. La plus grande fréquence posible n'est pas toujours celle qui est au final retenue. En effet, pour pouvoir réaliser des communications avec certains périphériques, il faut pouvoir disposer de fréquences spécifiques (dérivées du signal d'horloge originel). C'est d'ailleurs la raison qui fait qu'une horloge de voiture peut se dérègler beaucoup plus vite qu'une montre à quartz. Dans une telle montre, on utilise un quartz dit horloger de fréquence 32768 Hz, qui après plusieurs divisions successives par deux donne une fréquence

de 1Hz pour « battre » la seconde. Les microcontrôleurs embarqués dans un véhicule peuvent nécessiter des fréquences d'horloges spécifiques dont les divisions n'aboutissent pas exactement à une fréquence d'1 Hz, d'où les dérives constatées au bout de plusieurs semaines. On peut faire aussi le même constat pour des appareils audio/vidéo affichant l'heure (s'ils ne sont pas remis automatiquement à l'heure par transfert de données)

Pour éviter ce problème et donc assurer un heure stable dans le temps, certains fabricants, dont Nordic pour le microcontrôleur utilisé sur la carte Micro:Bit, implantent la possibilité d'avoir deux sources d'horloges simultanées :



Sur la carte Micro:Bit, seul un quartz de 16MHz a été implanté entre XC1 et XC2, libérant les broches XL1 et XL2 (broches 45 et 46 en fait) pour être utilisées autrement : l'une dédiée au bouton B, l'autre servant à la communication avec l'accéléromètre.

On découvre également sur le schéma 1 en Annexe, un circuit spécifique à ce microcontrôleur pour faire de la communication radiofréquences à 2,4 GHZ (ex Bluetooth) et que nous n'avons pas mis en œuvre dans le premier manuel.

Pour finir, sur toute la partie droite du schéma, on a les différentes lignes constituant le GPIO (General Purpose Input Output = Entrées Sorties à usage général) du microcontrôleur, dont une partie va se retrouver disponible directement sur le connecteur d'extension. Il suffit de voir les « étiquettes » marquées sur chacune de ces lignes (schéma 1 en annexe) : on les retrouve sur les broches du connecteur (schéma 2 en annexe).

Il n'y a donc aucune protection électrique entre le connecteur d'extension et le microcontrôleur !!! Cette remarque impose de faire un point sur les caractéristiques électriques de la carte et du microcontrôleur.

2.1.2. Spécifications et valeurs limites d'utilisation

Le tableau suivant spécifie les valeurs d'utilisation normale ainsi que les valeurs limites à ne pas dépasser pour le microcontrôleur :

- en **noir** les valeurs limites (minimum MIN et maximum MAX) entre lesquelles le microcontrôleur peut fonctionner normalement, TYP étant la valeur typique dite encore nominale de fonctionnement.
- en **rouge** les valeurs extrêmes au delà desquelles il y a un risque majeur de destruction partielle ou totale du microcontrôleur. L'utilisation prolongée du composant à ces valeurs limites peut en affecter la fiabilité.

	MIN.	MIN.	TYP.	MAX.	MAX.	Unité
ALIM.						
V_{DD}	-0,3	1,8	3,0	3,6	+3,9	V
V_{SS}					0	V
GPIO						
V_{IO}	-0,3				V_{DD}+0,3	V
V_{IH}		0,7.V_{DD}		V_{DD}		V
V_{IL}		V_{SS}		0,3		V
V_{OH}		V_{DD}-0,3		V_{DD}		V
V_{OL}		V_{SS}		0,3		V
T°		-25	+25	+75		°C

Tableau 1

V_{DD} : reliée à la ligne d'alimentation positive

V_{SS} : reliée à la masse de l'alimentation

V_{IO} : tension admissible sur une broche du GPIO

V_{IH} : Sur une entrée du GPIO, c'est la tension à appliquer pour qu'elle soit interprétée par le microcontrôleur comme étant un niveau 1

V_{IL} : Sur une entrée du GPIO, c'est la tension à appliquer pour qu'elle soit interprétée par le microcontrôleur comme étant un niveau 0

V_{OH} : tension générée pour un niveau haut (« 1 »), par le microcontrôleur, sur une sortie du GPIO

V_{OL} : tension générée pour un niveau haut (« 0 »), par le microcontrôleur, sur une sortie du GPIO

Exemple : si le microcontrôleur est alimenté par une tension V_{DD} = 3,3 V

- ne pas appliquer sur une broche du GPIO une tension supérieure à

$V_{IO} = V_{DD}$ + 0,3 V soit 3,3 + 0,3 = 3,6 V

- les niveaux logiques vont alors valoir :

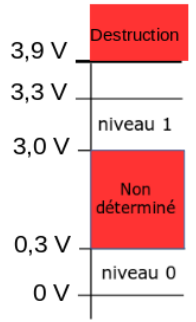
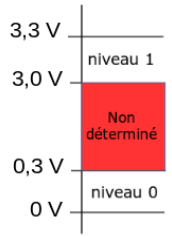
	En entrée	En sortie
V_{IH}	2,3 à 3,3 V	
V_{IL}	0 à 0,3 V	
V_{OH}		3,0 à 3,3 V
V_{OL}		0 à 0,3 V
Visualisation graphique :		

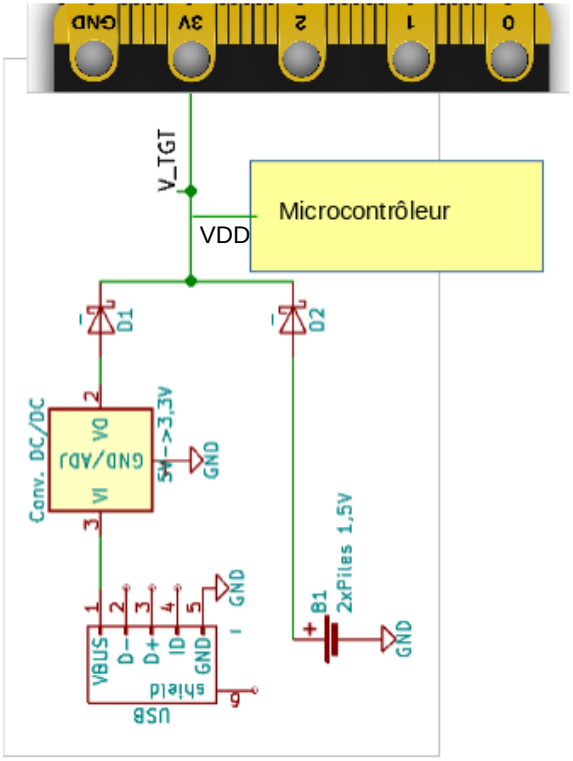
Tableau 2

2.2 Alimentation du montage

2.2.1. Les trois façons d'alimenter le microcontrôleur

Le schéma 1 en annexe montre que le microcontrôleur Nordic est alimenté au niveau de ses broches V_{DD} par une ligne référencée V_TGT provenant de la partie alimentation de la carte, dont on donne ci-dessous une représentation simplifiée. Le microcontrôleur peut être alimenté :

- par le câble USB relié au PC
- par un coupleur de piles (2x1,5V)
- par un montage externe connecté sur la broche « 3V »



En effet ces trois voies d'alimentation se retrouvent en un même point noté V_TGT.

Retenir que la valeur de la tension V_{DD} dont on a parlé au paragraphe 2.1.2 est égale à la tension présente sur cette ligne V_TGT qui conditionne donc également les niveaux de tension admissibles sur les broches du GPIO !!!

Analysons ce circuit d'alimentation :

- Par le câble USB :

  Un câble USB véhicule non seulement des données mais aussi du courant pour alimenter le montage auquel il est connecté. Ce courant électrique se fait sous une tension de 5 volts avec une intensité maximum de 500 mA. Cette tension de 5V est trop grande pour alimenter le microcontrôleur Nordic. Il faut donc intercaler un circuit de conversion DC/DC (continu/continu) pour ici abaisser la tension à 3,3V. Des circuits spécifiques DC/DC existent. Les concepteurs ont préféré se servir d'un module existant sur le deuxième microcontrôleur de la carte (celui qui sert d'interface entre l'USB et le microcontrôleur Nordic). Cependant l'intensité disponible en sortie de ce convertisseur est limitée.

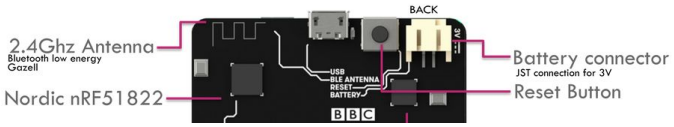
  Si on a l'habitude de décoder un schéma électronique, on pourra consulter en Annexe les schémas 3 et 4. Pour suivre plus facilement le cheminement, on a entouré en vert les différentes étiquettes affectées à cette ligne d'alimentation : au niveau du connecteur USB, elle est notée VBUS_IF et devient VBUS_IF2 après passage dans une résistance de limitation de courant et arrive dans le module convertisseur de tension DC/DC du deuxième microcontrôleur de la carte. Cela ressort sur la broche nommée VOUT3.3 avec l'étiquette +3.3V_IF. On passe alors au schéma 4 : après traversée de la diode D1 elle est notée V_TGT. (Les changements d'appellation sont nécessaires à cause des changements de valeur de cette tension tout au long de ce chemin)

  La notice de la carte Micro:Bit indique que la carte admet jusqu'à 120 mA par cette source d'alimentation, mais qu'elle en consomme une trentaine elle même. Donc :

90 mA, ce sera l'intensité maximale disponible pour faire fonctionner un montage additionnel si celui-ci est alimenté par les broches « 3V » et « GND » du connecteur d'extension lorsque l'on travaille avec le cordon USB.

- Par le connecteur pour piles (« Battery connector ») :

Ce petit connecteur permet de relier un boitier contenant deux piles « 1,5V » mises en série, soit une tension totale de « 3V » :



L'alimentation par piles permet une autonomie de mouvement de la carte Micro:Bit. Par contre il y a un petit souci dont il faut avoir conscience : le f.é.m d'une pile n'est pas constante durant toute sa durée de vie et le marquage « 1,5V » est plutôt une indication moyenne. Lorsqu'elle est neuve elle est voisine de 1,7 V et tombe autour de 1 V lorsque la pile est « usée ». Par ailleurs, en fonctionnement, la tension disponible aux bornes de la pile suit une loi du type :

$$U_{PN} = E - r.I$$

ce qui donne une tension inférieure à le f.é.m E (r étant la résistance interne de la pile). Si on reprend le tableau 1 des spécifications électriques :

	MIN.	MIN.	TYP.	MAX.	MAX.	Unité
GPIO						
V_{IO}	-0,3				V_{DD}+0,3	V
V_{IH}		0,7.V_{DD}		V_{DD}		V
V_{IL}		V_{SS}		0,3		V
V_{OH}		V_{DD}-0,3		V_{DD}		V
V_{OL}		V_{SS}		0,3		V

il faut avoir conscience que les valeurs des niveaux liés à V_{DD} varieront à la baisse au fur et à mesure de l'usure des piles utilisées.

Par ailleurs, pour des raisons d'économie, on peut vouloir mettre des « piles » rechargeables. La f.é.m de ces batteries, lorsqu'elles sont rechargées est plus basse que celle d'une pile alcaline neuve : autour de 1,2 ou 1,3 V.

Remarque : lorsque l'on regarde le schéma simplifié de l'alimentation, on constate que les deux lignes que l'on vient de décrire se rejoignent mais protégées l'une de l'autre par deux diodes (D1 et D2). Il s'agit de deux diodes à très faibles chute de tension (autour de 0,1 V pour ces modèles de type Schottky au lieu des 0,7 V pour de simples diodes au silicium). On pourra mesurer effectivement cette petite perte de tension : avec l'alimentation par USB on a pour V_TGT entre 3,1 et 3,2 V au lieu des 3,3 V.

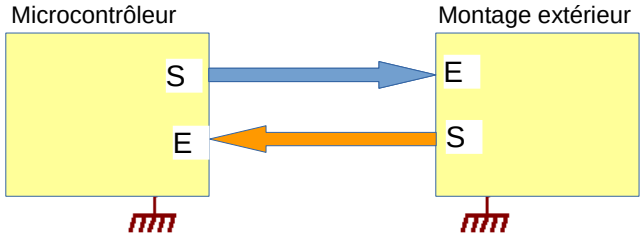
- La dernière possibilité, c'est l'alimentation de la carte Micro:Bit par celle du montage externe, reliée alors au broches « 3V » et « GND » du connecteur…

Il faudra être sûr de soi car là il n'y a aucune protection : si l'on applique une tension supériere à 3,9V ou si on fait une inversion de branchement on aura destruction de composants de la carte !

Nous déconseillons fortement cete possibilité, sauf si l'alimentation externe est réalisée avec un régulateur 3,3V avec impossibilité pour l'utilisateur de faire une inversion des polarités au niveau de la carte Micro:Bit.

2.2.2. Alimentation et compatibilité du GPIO

On rappelle tout d'abord un grand principe : une sortie est connectée à une entrée ; les masses doivent être communes.



La connexion d'un montage extérieur au GPIO se réalisera dans des conditions satisfaisantes si on sait répondre aux questions suivantes :

1. la tension appliquée par une sortie de l'un restera-t-elle dans les limites tolérées par l'entrée de l'autre ?
2. Une tension appliquée comme étant un niveau 1 par la sortie de l'un sera-t-elle interprétée correctement comme un niveau logique 1 par l'entrée de l'autre ? Même question pour un niveau 0 (problème beaucoup moins fréquent)
3. L'entrée attaquée ne réclame-t-elle pas une intensité plus grande que ce que peut délivrer la sortie de l'autre circuit ?

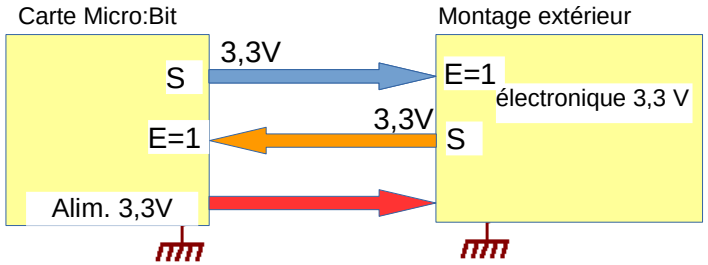
   Une partie de la réponse à cette dernière question se trouve dans la « datasheet » du composant : l'intensité maximum que peut délivrer une sortie du microcontrôleur est de **5 mA** … ce à quoi s'ajoute le fait qu'il ne peut y avoir plus de trois sorties à délivrer simultanément une telle intensité !

   Lorsque l'on veut acheter un module d'extension extérieur, il y a nécessité de vérifier que ce module sera bien compatible. en effet, il existe des modules « génériques » sensés servir pour différentes cartes de développement, mais initialement conçus pour le système Arduino qui fonctionne en 5V.

   Analysons différents cas de figure :

1. <u>Montage extérieur « 3,3 V » alimenté par la carte Micro:Bit</u>

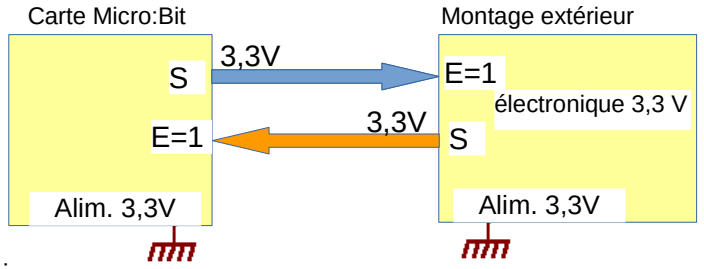
Si l'on conçoit soit même son propre montage extérieur, c'est assurément vers ce type de configuration que l'on doit se tourner :



Oui si :

- la consommation du montage extérieur reste inférieure à 90 mA si l'alimentation provient du câble USB
- l'électronique du montage extérieur est bien compatible 3,3V

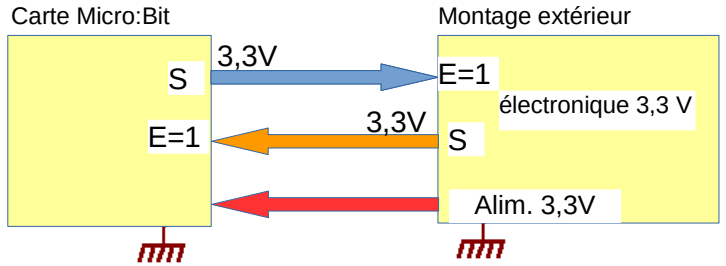
2. <u>Montage extérieur ayant sa propre alimentation 3,3 V :</u>



Oui si :

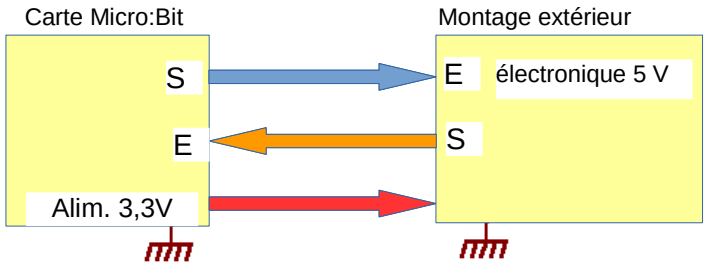
- la consommation du montage extérieur est supérieure aux 90 mA restant disponibles lorsque l'alimentation provient du câble USB
- l'électronique du montage extérieur est bien compatible 3,3V

On aurait alors intérêt à alimenter la carte Micro:Bit avec cette alimentation extérieure (Attention : avec les réserves exprimées au 2.2.1). Cela donnerait le montage suivant :



Oui : si on s'est assuré au niveau de la carte Micro:Bit de l'impossibilité d'inversion de polarité de l'alimentation extérieure

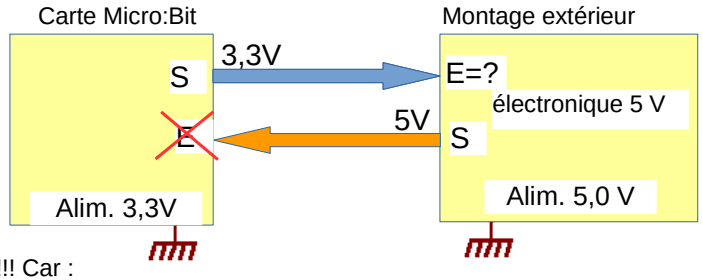
3. Montage extérieur « 5 V » alimenté par la carte Micro:Bit



ATTENTION risque de non fonctionnement ou de dysfonctionnement du montage extérieur, la tension d'alimentation de 3,3V étant insuffisante. Consulter la notice en espérant qu'elle soit précise sur ce sujet.

A priori, pas de risque à essayer sauf si le montage extérieur dispose d'un convertisseur DC/DC 3,3V → 5V auquel cas une sortie du montage extérieur, mise au niveau 1, génèrerait une tension de 5V sur l'entrée correspondante de la carte Micro:Bit ce qui risquerait de la détruire ! (Cela serait identique en fait au cas ci-dessous).

4. Montage extérieur ayant sa propre alimentation 5V

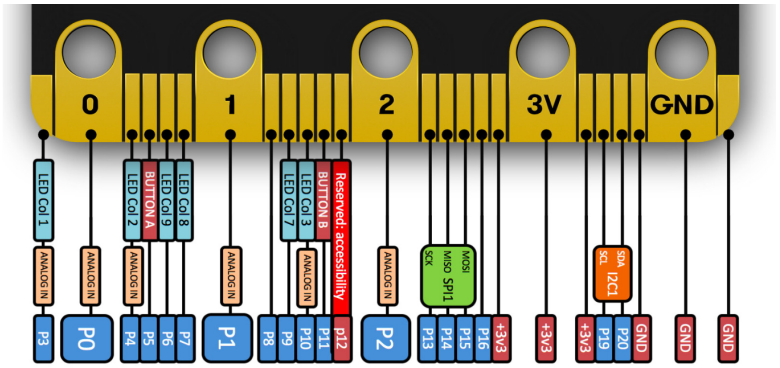


NON !!! Car :

- un niveau 1 en sortie de la carte Micro:Bit pourrait ne pas être interprété convenablement…
- ...mais surtout, un niveau 1 appliqué sur une sortie du montage extérieur, c'est une tension de 5,0V que l'on retrouve sur l'entrée correspondante de la carte Micro:Bit ce qui risque d'entraîner sa destruction

La solution consisterait en la mise en place d'une interface d'adaptation de niveaux entre la carte Micro:Bit et le montage extérieur… Le jeu en vaut-il la chandelle ??? N'est-il pas plus simple de chercher une solution moderne en 3,3V ?

2.3 Le connecteur d'extension :



2.3.1. Vue globale du connecteur

Le connecteur d'extension présente un total de « 25 » broches pour étendre les fonctionnalités de la carte. En fait plusieurs sont redondantes et liées à l'alimentation (trois broches de masse (GND) et trois pour la ligne positive 3,3V), ce qui en soit n'est pas inutile, mais ne participe pas directement à la communication avec l'extérieur. A ces six broches, il faut ajouter encore une ligne numéroté P12 et indiquée « Reserved accessibility »… ce qui n'incite pas à chercher à l'utiliser). Il nous reste donc toutes les broches représentées en bleu sur ce schéma, soit un total de 18 broches qui constituent ce que l'on appelle le GPIO de la carte.

On peut remarquer que certaines broches peuvent avoir des fonctions différentes :

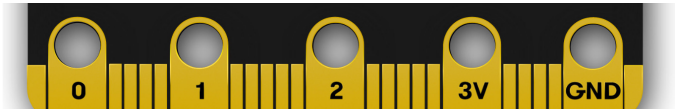
- toutes peuvent être des entrées ou sorties numériques

- six d'entre elles peuvent servir d'entrée analogique (marquées « Analog In »). Associées en interne à un convertisseur analogique-numérique (CAN), elles permettent la réalisation de mesures sur des grandeurs analogiques, et donc à travers un capteur, permettront des mesures de grandeurs physico-chimiques.
- cinq peuvent être utilisées pour des communications série de type SPI ou I2C

Enfin, plusieurs d'entre elles se trouvent partagées avec l'afficheur 5x5 Leds ou les boutons poussoirs.

2.3.2. Les différentes façons de s'y connecter

1- A l'aide fiches bananes :



Cinq broches du connecteurs ont été élargies et percées pour pouvoir être utilisées avec des fiches bananes au format standard de 4 mm.

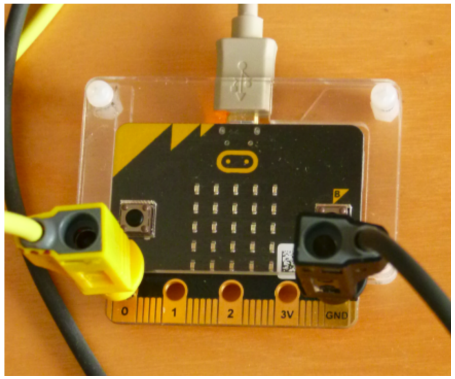
Les deux de droite concernent l'alimentation :

- « 3V », la ligne positive
- « GND » (= ground), la masse (0V)

On peut se servir de ces deux bornes pour alimenter le montage externe à partir de la carte Micro:Bit , ou à l'inverse, alimenter la carte Micro:Bit à partir de l'alimentation du montage externe (Bien relire le paragraphe 2.2.2 si on a un doute : attention avec des élèves débutants qui ont vite fait d'inverser ce type de fils...)

Les trois autres bornes, notées 0, 1, et 2 peuvent être utilisées en entrée ou sortie numérique ou bien encore, en entrée analogique. Notons qu'il existe une troisième façon de se servir de ces bornes : comme un bouton poussoir en se servant du corps humain comme « conducteur » en appuyant simultanément sur l'une de ces bornes et sur la borne de masse…

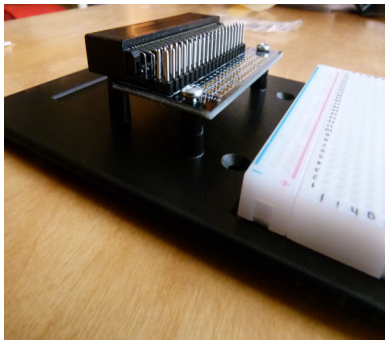
Retenons que cette possibilité de connexion par fiches bananes permet de se connecter à moindre frais sur un montage traditionnel d'électricité, mais qu'il faudra être sûr des niveaux générés ou admis par ce montage externe :

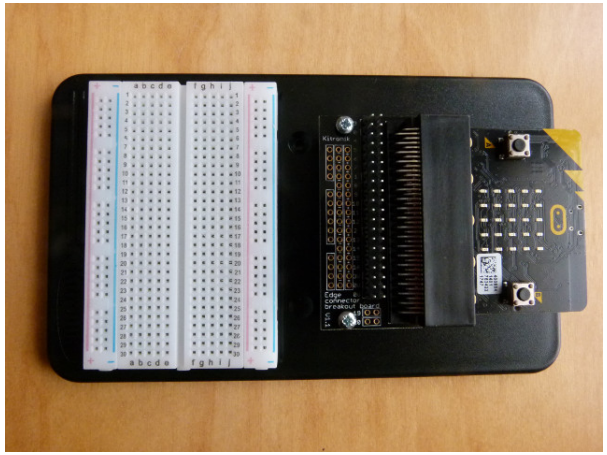


2- Avec un kit de prototypage :

Ci-dessous le kit de prototypage de la marque Kitronic :

Sur la plaque support, on vient fixer un connecteur (pour recevoir la carte Micro:Bit) et une plaque d'expérimentation qui accueillera les composants du montage exxpérimental. Deux types de fils sont fournis : femelle-mâle pour relier le connecteur à la plaque d'expérimentation et mâle-mâle pour réaliser des jonctions sur la carte d'expérimentation



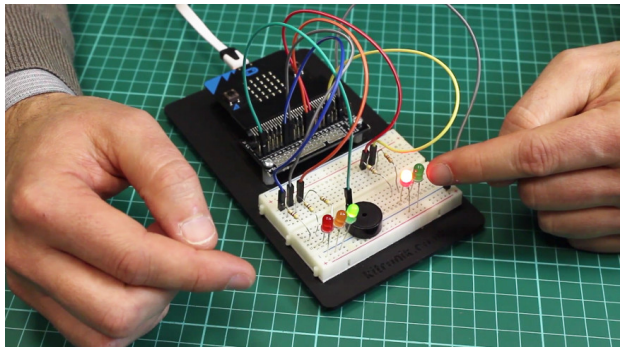


Le kit assemblé avec une carte Micro:Bit

(attention : non fournie dans le kit)

Ce type de matériel, comme son nom l'indique, est destiné à du prototypage, soit pour faire des expérimentations, soit pour mettre au point un montage avant de passer à la version définitive sur circuit imprimé. Très pratique quand on dispose d'un stock varié de composants, ce type de kit nécessite cependant une certaine habitude pour ne pas faire d'erreurs (le montage réalisé se trouve directement relié au GPIO du microcontrôleur, sans aucune protection)

Exemple d'utilisation :



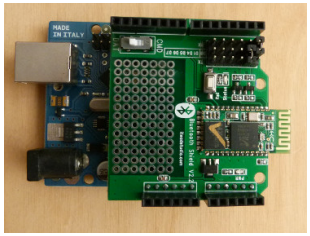
Source : Kitronik

<u>3- Avec des modules d'extension du commerce :</u>

Suite au développement des cartes de développement clé en main (Arduino, Micro:Bit, Beagle etc) de nombreuses sociétés se sont positionnées sur le développement de modules d'extension avec deux modèles différents :

- module d'extension adapté à la carte de développement :

Ci-contre une carte Arduino accueillant une carte d'extension (un « shield ») Bluetooth

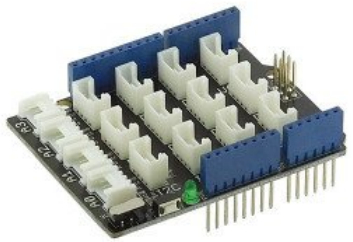


Cette carte d'extension ne peut fonctionner qu'avec une carte Arduino ou ou une carte compatible

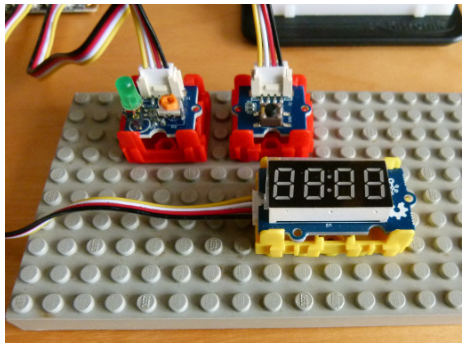
- module d'extension adapté à une carte de connexion spécifique à la carte de développement. Les modules d'extensions peuvent alors être réutilisés sur d'autres cartes de développement, moyennant l'achat de la carte de connexion spécifique.

Ci-dssous un exemple avec <u>le système Grove que nous utiliserons dans ce manuel</u> :

- à gauche le « shield » d'extension pour Arduino
- à droite le shield d'extension pour Micro:Bit (qui lui n'accepte que quatres modules mais dispose de quelques entrées de type banane)

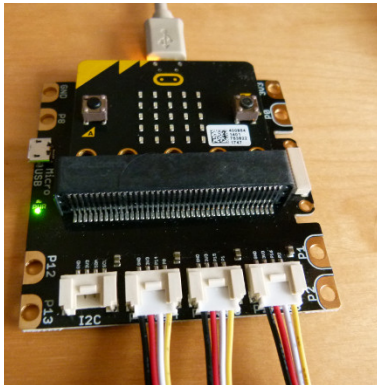


Chacun de ces shields dispose de petiits connecteurs qui permettent de connecter différents modules (et il y en a des dizaines différents, de plus à des prix très raisonnables !). Avec ce dispositif, aucune soudure à faire, pas de risque de branchement inversé : la solution la plus simple pour démarrer.



Une bonne idée : presque tous les modules de la marque Grove peuvent s'insérer dans un support compatible avec les plaques Lego !
Ici : un module Led verte, un module bouton poussoir et un module de quatre afficheurs 7 segments

Un exemple d'utilisation : la carte Micro:Bit vient s'insérer dans le shield d'extension ; ici les câbles des trois modules de la photo précédente :

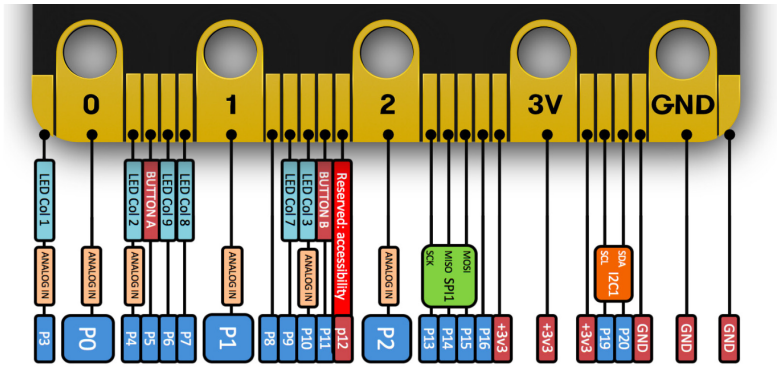


Remarque : la faible distance entre la carte Micro:Bit et le shield ne permet pas l'utilisation de coques de protection trop encombrantes pour la carte Micro:Bit...

Plus de détails sur tous les modules disponibles sur les sites de revendeurs ou ici : http://wiki.seeedstudio.com/Grove_System/

2.4 Accéder aux entrées/sorties avec Python

2.4.1. Le problème des broches partagées



Comme le montre ce schéma du connecteur, plusieurs broches accessibles en externe partagent des fonctions avec des dispositifs internes à la carte :

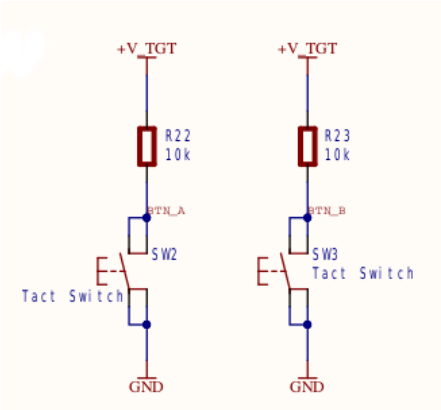
- les broches P3, P4, P6, P7, P9 et P10 sont utilisées par l'afficheur 5x5 Leds. Si l'on souhaite les utiliser, il faudra désactiver l'afficheur :

  display.off()

  Pour le réactiver, on utilisera la méthode inverse :

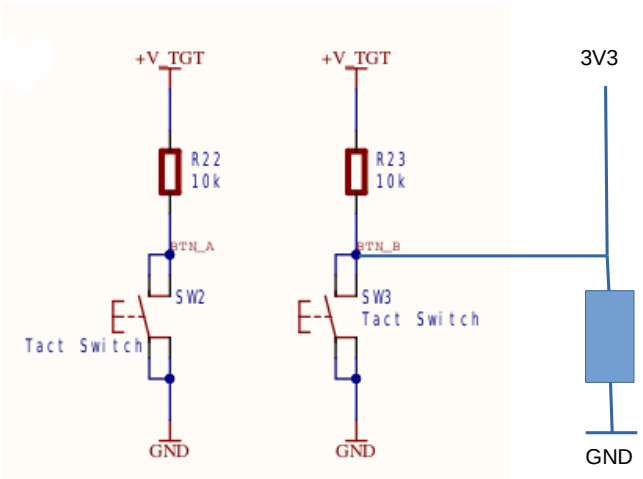
  display.on()

- les broches P5 et P11 sont utilisées par les boutons A et B de la carte. Leur implantation est donnée dans le schéma suivant. On voit que par défaut ces deux lignes (notées BTN_A et BTN_B) se retrouvent ramenées par défaut au niveau haut (3,3V) à travers une résistance de 10 kΩ. L'appui sur le bouton poussoir ramène alors la ligne correspondante à la masse donc au niveau 0.



Cela n'interdit pas d'utiliser ces broches P5 et P11 si cela convient au montage, mais il faut avoir en tête que l'appui sur le bouton poussoir entraînant donc un passage à 0 de la broche risque de modifier le comportement du programme !!!

Attention : ne pas faire ceci :



En effet l'appui accidentel sur le bouton poussoir B créerait alors un court-circuit sur l'alimentation (entre le +3,3V et la masse) !!!

A moins d'être sûr de ce que l'on fait, il est peut-être préférable de ne pas utiliser ces broches.

- les broches P19 et P20 sont associées au module I2C interne du microcontrôleur et se trouvent activées par défaut pour cette tâche.

  Les concepteurs déconseillent fortement l'utilisation de ces broches à d'autres fins que de la communication I2C car l'accéléromètre et le magnétomètre partagent eux aussi cette ressource. Ils communiquent avec le microcontrôleur via le bus I2C, ce qui n'interdit pas par contre de venir placer un autre composant I2C sur ce bus.

- les broches P13, P14 et P15 partagent un autre module de communication sérié de type SPI qui n'est pas activé par défaut et qui n'est pas utilisé par ailleurs sur la carte. On pourra donc utiliser ces broches à volonté si on n'utilise pas ce module SPI

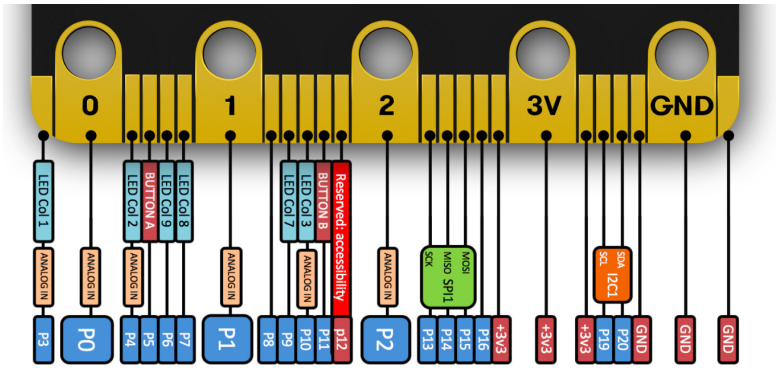
Un outil intéressant en ligne pour avoir le détail du fonctionnement de chacune des broches du connecteur se trouve ici :

https://microbit.pinout.xyz/ .

En cliquant sur une broche on a une explication sur celle-ci (en anglais) :

Remarque : le shield d'extension Grove pour Micro:bit présenté dans le paragraphe 2.3.2-3 est particulièrement bien pensé puisqu'il évite soigneusement la possibilité d'utiliser les lignes partagées qui pourraient poser problème. On aura donc une utilisation sans souci avec lui.

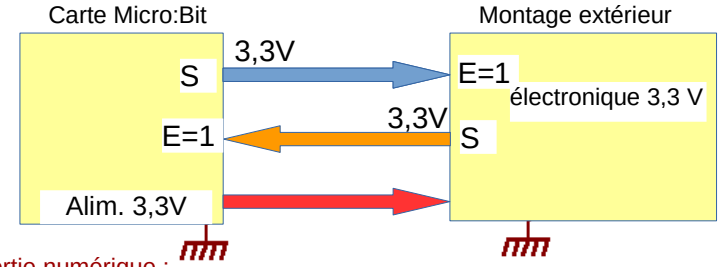
2.4.2. Entrées/sorties numériques



Toutes les broches désignées en bleu ci-dessus peuvent a priori fonctionner en entrée ou en sortie numérique (on dit encore « digitale »). Il s'agit donc des broches P0 à P11 P13 à P16 mais à l'exclusion de P19 et P20.

Remarque : bien relire le paragraphe 2.4.1 pour les broches associées à l'afficheur ou aux boutons ! On pourra sans soucis expérimenter dans un premier temps avec les broches P0, P1 et P2

On rappelle un exemple de connexion convenable (Cf. 2.2.2) :



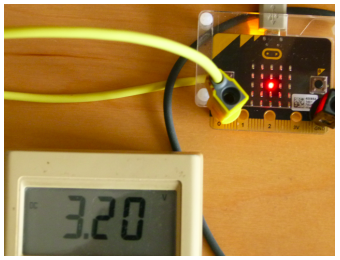
1- Sortie numérique :

Une broche configurée en sortie peut être mise au potentiel 3,3V, ce sera un niveau 1 ou bien mise au potentiel 0V, ce sera un niveau 0 (relire le tableau 2 au paragraphe 2.1.2). Cette action se réalisé par une **écriture** avec la méthode :

write_digital(b)

b ne pouvant prendre que la valeur 1 ou 0

Exercice : Relier la carte Micro:Bit à un voltmètre de façon à pouvoir mesurer la tension entre la broche P0 et la masse :




```
from microbit import *

display.set_pixel(2,2,9)
sleep(2000)

pin0.write_digital(1)

while True:
    pass
```

Le multimètre n'affiche pas 3,3V (ne pas oublier la petite chute de tension dans la diode Schottky sur la ligne d'alimentation USB. Cf 2.2.1).

On a mis quelques instructions avant d'écrire un niveau 1 sur la broche P0 pour faire remarquer que tout le temps que le niveau n'a pas été défini, la mesure de tension donne une valeur intermédiaire entre un niveau 0 et un niveau 1. C'est une sécurité : la broche est mise en « haute impédance », de cette façon elle n'est ni une entrée ni une sortie.

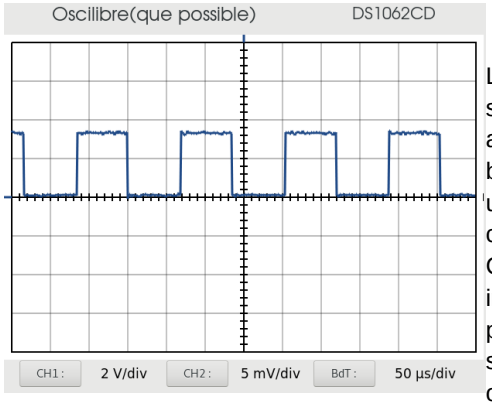
Modifier le code pour mettre au niveau 0 la broche P0. On pourra aussi aller dans le REPL pour entrer directement la ligne d'instruction :

```
pin0.write_digital(0)
```

Exercice : débrancher les fils du voltmètre pour les brancher sur l'entrée d'un oscilloscope numérique. Entrer le code suivant :

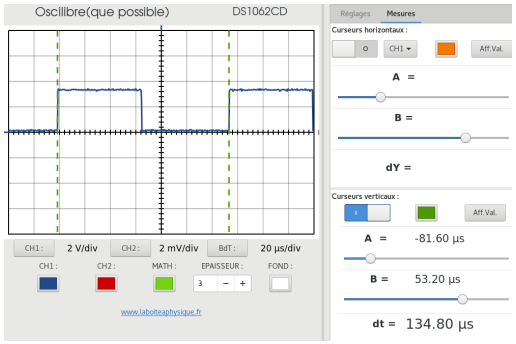
```
from microbit import *

display.set_pixel(2,2,9)
sleep(2000)

while True:
    pin0.write_digital(1)
    pin0.write_digital(0)
```

On obtient l'oscillogramme suivant :



La boucle infinie nous donne sur la broche P0 une alternance de niveaux haut et bas. On vient en fait de faire un générateur de signaux carrés d'amplitude 3,2V. Comme il n'y a aucune instruction entre la ligne pin0.write_digital(1) et la ligne suivante pin0.write_digital(0), on obtient donc ici la plus petite période et donc la plus haute fréquence possible par cette méthode.

Améliorons la précision sur la mesure :



On obtient une période de 135 µs d'où une fréquence :

$$f = \frac{1}{135 \cdot 10^{-6}} = 7{,}4 \text{ kHz}$$

Voilà une valeur qui peut sembler bien faible pour un processeur qui tourne avec une horloge de 16MHz !!! On touche là au problème des langages

interpétés, comme l'est Python, qui sont toujours plus lents à l'exécution que les langages compilés.

Exercice 2.4.2_1 : Modifier le code précédent de façon à obtenir une tension en créneaux de fréquence f = 10 Hz

Exercice 2.4.2_2 : modifier une seule des temporisations et observer le résultat à l'oscilloscope

Remarque : aller voir les corrections de ces exercices car des compléments sont donnés sur les timings !

2- Entrée numérique :

On lit l'état d'une entrée numérique avec la méthode :

read_digital()

qui retourne un entier égal à 0 ou 1.

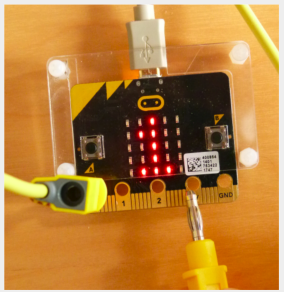
Exemple : pour lire l'état de la broche P0 et mettre le résultat (0 ou 1) dans une variable appelée val :

```
val = pin0.read_digital()
```

Exercice 2.4.2_3 : Ecrire le code permettant d'afficher le chiffre 1 ou le chiffre 0, selon le niveau auquel se trouve la broche P0.

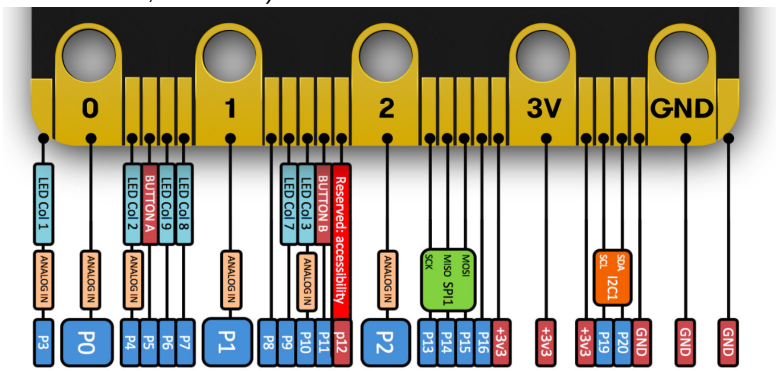
Pour tester ce programme : brancher l'extrémité d'un fil banane au niveau de la broche P0. Poser son autre extrêmité sur la broche « 3V3 » et la retirer, puis la remettre. Observer :

```
from microbit import *

display.set_pixel(2,2,9)

while True:
    val = pin0.read_digital()
    display.show(val)
    sleep(100)
```



2.4.3. Entrée analogique

Le microcontrôleur Nordic implanté sur la carte Micro:Bit dispose d'un module convertisseur analogique numérique (CAN en français ou ADC en anglais pour Analog to Digital Conversion) sur 10 bits (soit 2^{10} = 1024 valeurs possibles de conversion entre 0 et 1023). Ce CAN peut être associé à huit broches du microcontrôleur. Six d'entre elles se retrouvent sur le connecteur d'extension (broches P0, P1 et P2 accessibles avec des fiches bananes et P3, P4 et P10) :



P0, P1 et P2 n'étant pas partagées avec des fonctions internes à la carte, ce seront les entrées les plus simples à utiliser dans ce mode. On n'oubliera pas de respecter les limites électriques applicables aux entrées du GPIO. A cette occasion rappelons qu'une entrée n'accepte pas de tension négative, ce qui veut dire bien sûr de ne pas faire d'inversion de branchement (si l'on veut par exemple mesurer la tension aux bornes d'une piles avec deux fils bananes). Mais cela signifie aussi que ces entrées n'acceptent pas de tension alternative.

Pour lire le résultat de la conversion analogique-numérique représentative de la tension présente sur la broche P0 on utilise la méthode read_digital() :

```
val = pin0.read_analog()
```

La variable « val » sera un entier compris entre 0 et 1023 :

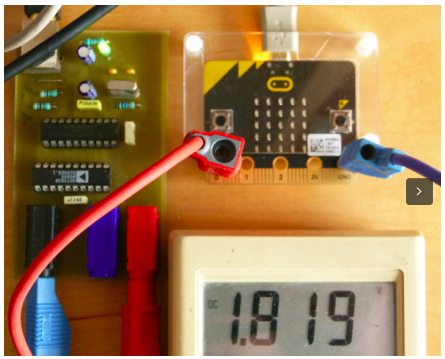
- 0 si $U_{P0\text{-}GND}$ = 0 V
- 1023 si $U_{P0\text{-}GND}$ = 3,3 V

« 3,3V » étant la tension de référence du convertisseur, valeur pour laquelle la documentation BBC n'est pas très explicite, tant sur sa provenance que sur sa précision.

Pour tester la lecture d'une entrée analogique (P0 ici) entrer le programme suivant :

```
from microbit import *

while True:
    can = pin0.read_analog()
    U = can*3.3/1024
    print ('CAN = ',can, ' U = ', U)
    sleep(500)
```

Ce programme a été testé avec le matériel suivant :



- une carte microcontrôlée développée il y a quelques années dans la seconde édition de l'ouvrage « L'USB pour tous ». Cette carte permet de disposer de deux sources de tension de référence pouvant varier de 0 à 3,3V. Une seule est utilisée ici. La sortie de cette carte est reliée à l'entrée P0 de la carte micro:bit.
- Un multimètre 2000 points sur le calibre 2V permet de vérifier la valeur de la tension annoncée par le logiciel

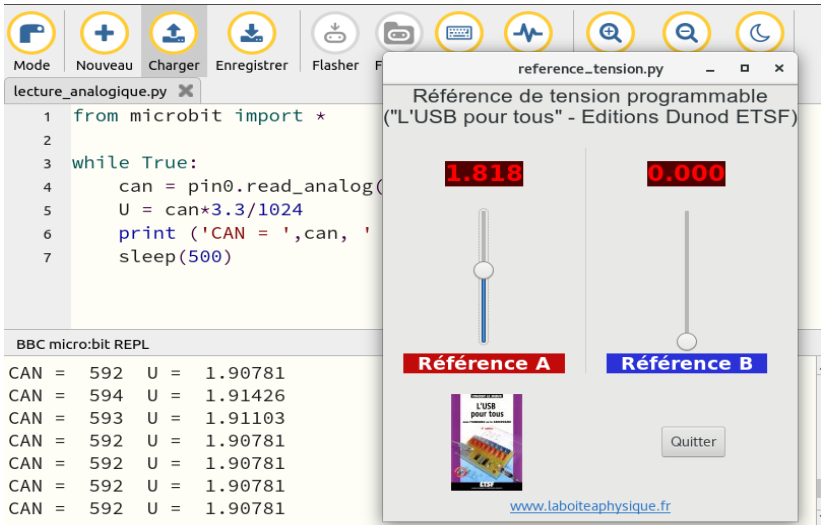
Après flashage du logiciel dans la carte micro:bit, le REPL a été activé pour suivre deux fois par seconde le résultat de la ligne d'instruction :

```
print ('CAN = ',can, ' U = ', U)
```

qui affiche sur une ligne :

- la valeur de la conversion (variable can)
- le résultat de la conversion en volts (variable U)

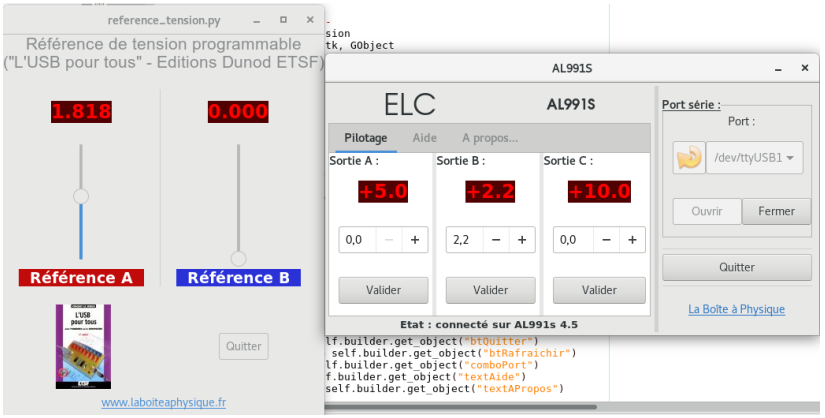
La copie d'écran ci-dessous montre les deux logiciels en fonctionnement (Pour en savoir plus sur le logiciel Référence de tension, qui a été réécrit en Python, aller à l'adresse : https://laboiteaphysique.fr/site2/index.php/programmation-et-logiciels/lusb-pour-tous)



Alors qu'il y a un très bon accord entre la valeur annoncée par le logiciel Référence de tension et l'indication donnée par le multimètre, on constate un écart non négligeable (de 5% ici) avec celle annoncée par la carte micro:bit… écart que l'on a retrouvé sur d'autres valeurs essayées, ce qui ne peut qu'inciter à investiguer davantage pour comprendre ce qui se passe…

Les soupçons se portant sur l'imprécision quant à la source de la tension de référence pour la réalisation de la conversion, l'expérience suivante a été réalisée :

la carte Micro:Bit a été déconnectée de l'ordinateur et alimentée par une alimentation externe ELC AL991S (qui est programmable et pour laquelle un logiciel de pilotage écrit lui aussi en Python est disponible ici : https://laboiteaphysique.fr/site2/index.php/programmation-et-logiciels/alimentation-al991s L'intérêt d'utiliser le logiciel de pilotage vient du fait que la tension n'est appliquée qu'après appui sur le bouton « Valider » correspondant ce qui évite un excès de tension réalisé par mégarde lors du réglage) Cette alimentation dispose de plusieurs sorties : la carte Micro:Bit a été alimentée avec la sortie B que l'on a fait varier à l'aide du logiciel de commande. On a gardé la valeur de 1,818V sur l'entrée P0 pendant toute la manipulation :



Le programme précédent a été modifié pour lire le résultat de la conversion analogique numérique sur l'afficheur de la carte :

```
from microbit import *

while True:
    can = pin0.read_analog()
    U = can*3.3/1024
```

```
    #print ('CAN = ',can, ' U = ', U)
    display.scroll(str(can))
    sleep(500)
```

(on n'a gardé que le code de conversion, car un entier est moins pénible à lire en défilement sur l'afficheur !). Voici les résultats :

VDD (V)	Code CAN	Tension calculée (V)
2,20	847	1,820
2,42	773	1,827
2,61	717	1,828
2,82	665	1,831
3,01	623	1,831
3,22	585	1,840

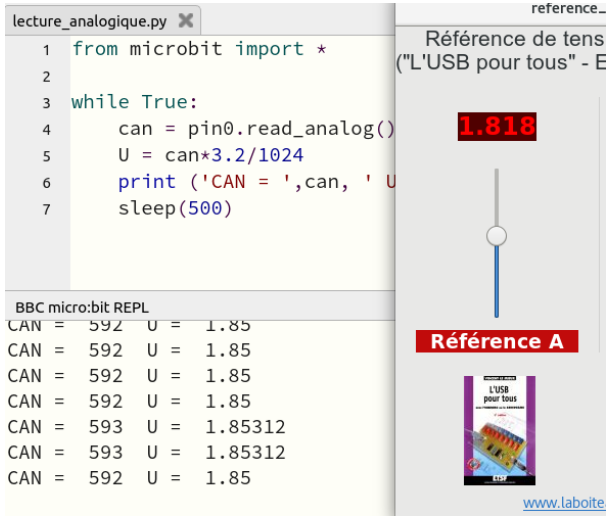
Pour une même tension appliquée sur l'entrée analogique, le code de conversion fourni dépend de la tension d'alimentation. Les soupçons étaient bons … la référence de tension pour la conversion analogique-numérique est liée à la tension d'alimentation.

Conséquences :

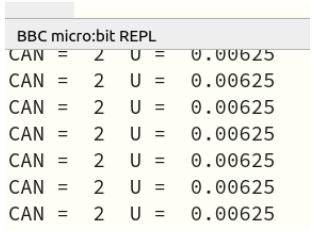
1. Si la carte Micro:Bit est connectée au PC, il faut se souvenir que la tension d'alimentation ne vaut pas 3,3V mais qu'elle **est autour de 3,2V** à cause de la chute de tension qui a lieu dans la diode Schottky D1 (Annexe : schéma 4). Le programme doit donc être modifié :

```
from microbit import *

while True:
    can = pin0.read_analog()
    U = can*3.2/1024
    print ('CAN = ',can, ' U = ', U)
    sleep(500)
```


Avec cette correction, on obtient un résultat de conversion beaucoup plus satisfaisant et conforme aux incertitudes liées aux spécifications du module ADC, comme le montre la copie d'écran reprenant la manipulation de départ avec le code modifié au niveau de la référence de tension :



Remarque : une erreur d'acquisition facile à corriger c'est l'erreur d'offset (ou décalage). elle se met fort bien en évidence en reliant l'entrée analogique P0 à la masse à l'aide d'un fil de connexion (rien d'autre n'étant bien sûr connecté à P0). Alors que l'on s'attend à avoir un code nul après conversion, on constate parfois une valeur non nulle (ici 2 sur le module utilisé) :

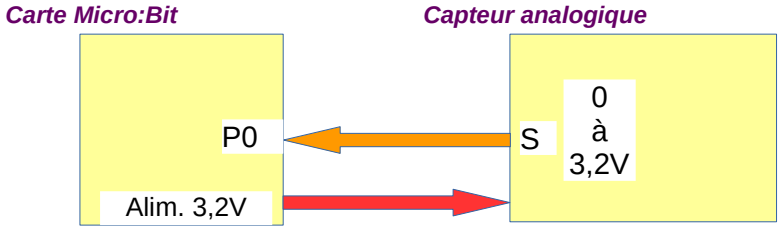


Cette erreur de décalage peut facilement être compensée par programmation comme ci-dessous :

U = (can - 2)*3.2/1024

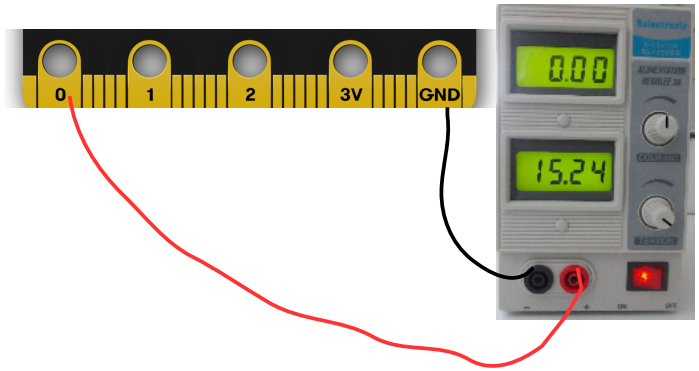
2. Si la carte est alimentée par piles, la tension d'alimentation va varier avec le type de piles utilisées, leur usure, voire également avec la consommation instantanée (qui peut varier de façon importante si ces piles servent aussi à alimenter par exemple un moteur de façon intermittente). Le résultat de la conversion analogique numérique, tributaire de cette tension ne pourra pas être considérée comme étant d'une grande précision.

La tension maximum applicable sur une entrée du GPIO étant VDD + 0,3V cela fait donc 3,5V lorsque la carte est alimentée par l'USB. Pour éviter tout risque de dépassement sur cette entrée, il est préférable que le capteur qui fournira la tension analogique à mesurer soit alimenté par l'alimentation de la carte :



Pour apprendre à travailler avec la conversion analogique-numérique :

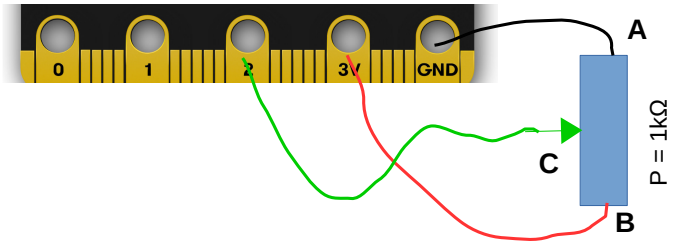
- ***ne surtout pas faire ce type de montage !!! :***



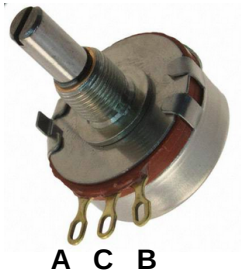
Avec une telle alimentation ajustable jusqu'à la valeur de 15V, on risque de faire accidentellement un réglage dépassant le niveau maximum toléré par l'entrée analogique.

- ***par contre un montage tel que celui proposé dans l'exercice ci-dessous convient parfaitement :***

Exercice 2.4.3_1 : réaliser le montage suivant et écrire le code permettant de voir sur l'afficheur la tension en sortie du potentiomètre. (tension entre le curseur et la masse). Cette tension sera affichée avec la précision du dixième de volt. On indiquera à l'utilisateur que chaque mesure sera déclenchée par l'appui sur le bouton A.



On rappelle le type de boitier le plus courant pour les potentiomètres de type rotatif, pour lequel une solution de branchement avec fils bananes est certainement disponible au laboratoire :



Vérifier à l'aide d'un voltmètre l'indication donnée par l'afficheur

2.4.4. Sortie PWM (Pulse Width Modulation)

2.4.5. Composant I2C (Inter Integrated Circuit)

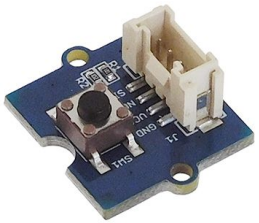
2.4.6. Composant SPI (Serial Peripheral Interface)

3 Utiliser le système Grove

Une présentation sommaire de ce système a été faite au paragraphe 2.3.2.

Ce système modulaire d'extension nous ayant semblé intéressant nous allons le détailler davantage ici.

3.1 Connectique des modules Grove



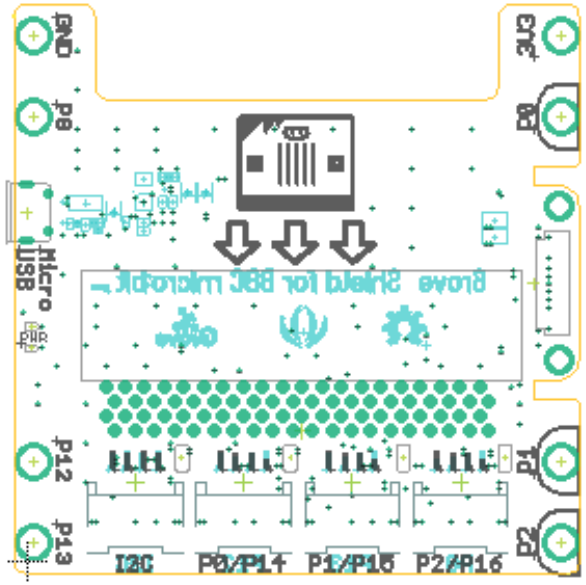
Tous les modules Grove sont reliés à la platine d'accueil avec le même type de câble constitué de 4 fils :

- **pin 1** - Jaune (ligne de signal Nº1 ; celle utilisée si le module nécessite une seule ligne de signal)
- **pin 2** - Blanc (ligne de signal Nº2 , si une deuxième est nécessaire au fonctionnement du module)
- **pin 3** - Rouge - VCC donc 3,3V pour la carte Micro:Bit ...mais 5V sur Arduino
- **pin 4** - Noir - GND ... la masse

Les connecteurs pour circuit imprimé étant disponibles à la vente, il est donc possible de créer ses propres modules. Ci-dessous les deux connecteurs (coudé ou droit) :

Pour la carte Micro:Bit, la platine de liaison (Shield Grove pour micro:bit Réf : 103100063) permet la connexion de 4 modules :

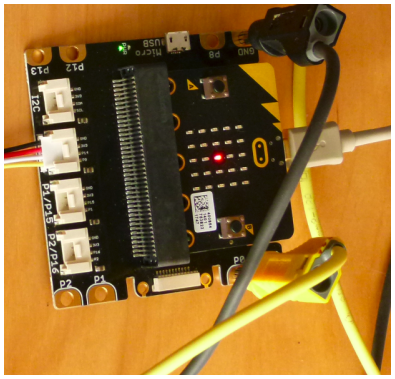
- un module supportant le protocole de communication série I2C
- 3 modules généraux travaillant en digital ou en analogique



D'après le principe de câblage évoqué un peu plus haut, on constate qu'un module Grove occupe systématiquement 2 lignes de données (même si parfois une seule n'est réellement utilisée). Pour le shield Grove utilisé ici, ce sont les broches :

- P19 et P20 sur le connecteur I2C (il permet de faire fonctionner certains périphériques utilisant cette norme de communication série)
- P0 et P14 sur le connecteur 2
- P1 et P15 sur le connecteur 3
- P2 et P16 sur le connecteur 4

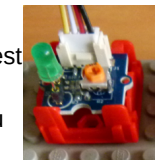
Ce shield dispose aussi de bornes bananes 4 mm, situées sur les côtés, pour pouvoir alimenter un montage externe (bornes « GND » et « 3V3 ») ou utiliser des signaux sur les broches P0, P1, P2, P8, P12, P13 et GND. Ci-dessous on avait besoin de récupérer le signal P0 utilisé par le module ultrason qui était branché sur le connecteur P0/P14. Cela a été fait à l'aide de fiches bananes (entre la pastille P0 et la pastille de masse GND). Cela nous a permis d'envoyer ce signal vers un oscilloscope pour l'analyser :



A noter également la présence d'une prise USB supplémentaire, qui ne sert pas à véhiculer des données) mais qui peut servir à alimenter les modules externes si les 90 mA que peut fournir la carte BBC Micro:Bit ne suffisent pas.

3.2 Le module Led

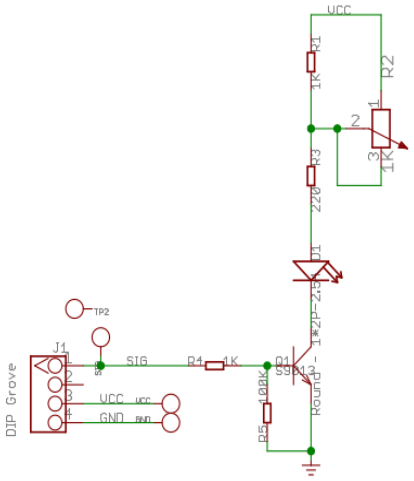
Le module décrit ici a pour référence : Seedstudio : 104030007.



Cette référence correspond au module fourni avec une Led verte. D'autres couleurs sont disponibles. La Led n'est pas soudée: juste enfoncée dans un connecteur. Cela facilite le changement de la Led (pour la maintenance ou pour le choix d'une autre couleur), mais il faut veiller à la placer dans le bon sens !

3.2.1. Présentation

Le schéma électrique du module est fourni par le fabricant :



A la lecture du schéma, on constate que la Led est pilotée par une seule ligne de signal (on pouvait s'y attendre !) et notée SIG. Cette ligne pilote un transistor NPN qui sera :

- passant lorsque le niveau appliqué sur sa base sera élevé (donc un état haut = "1") ; la Led sera donc allumée
- bloqué lorsque le niveau appliqué sur sa base sera faible (donc un état bas = "0") ; la Led sera alors éteinte.

On remarque la présence d'un "potentiomètre" monté en fait en résistance ajustable qui permet d'ajuster la luminosité de la Led (c'est aussi une petite astuce du concepteur pour pouvoir utiliser ce module en 3,3V (avec la carte Micro:Bit) ou en 5 V (avec la carte Arduino). Si le montage ne semble pas fonctionner, vérifier :

1. que la Led est montée dans le bon sens
2. si la Led n'est pas sous alimentée en tournant le potentiomètre

La ligne de signal unique utilisée est, comme indiqué dans le principe de câblage, sur la broche N°1 du connecteur du module. Selon le connecteur utilisé du shield Grove, on devra donc travailler sur le niveau de la ligne :

P0 pour le connecteur N°2 (noté P0/P14)

P1 pour le connecteur N°3 (noté P1/P15)

P2 pour le connecteur N°4 (noté P2/P16)

3.2.2. Mise en œuvre :

Pour allumer ou éteindre la Led, on doit donc appliquer un niveau logique 1 ou 0 sur la broche à laquelle le module Led est relié. On va donc utiliser l'instruction :

pin0.write_digital( ) avec une valeur de paramètre égal à 1 ou 0

Par exemple, pour faire clignoter la Led d'un module Led Grove branché sur le connecteur P0/P14 (on a choisi de faire un clignotement permanent, la Led s'allumant pendant une seconde et s'éteignant pendant une demi-seconde) :

```
from microbit import *

while True:
    pin0.write_digital(1)  # allumage de la Led
    sleep(1000)
    pin0.write_digital(0) # extinction de la Led
    sleep(500)
```

Tourner le potentiomètre intégré au module Led pour en voir l'effet.

Modifier ce code en branchant le module Led sur un autre connecteur du shield Grove (P1/P15 ou P2/P16).

3.2.3. Utiliser l'aliasing

L'allumage ou l'extinction de la Led nécessite de spécifier la broche (par exemple pin0) sur laquelle la Led est connectée. Pour des raisons de lisibilité du code, il serait plus simple de pouvoir écrire par exemple :

```
led.write_digital(1)  # allumage de la Led
```

Il suffit pour cela de créer un *alias* de l'objet pin0 en écrivant :

```
from microbit import *

led = pin0          # création d'un alias de pin0
```

et de remplacer « pin0 » par « led » dans le reste du code :

```
while True:
    led.write_digital(1)  # allumage de la Led
    sleep(1000)
    led.write_digital(0) # extinction de la Led
    sleep(500)
```

Cette possibilité d'*aliasing* a plusieurs avantages :

1. Si pour une raison quelconque on doit changer le module Led de connecteur (par exemple pour le faire passer de pin0 à pin2), il suffit de faire une seule modification au début du programme :

```
from microbit import *

led = pin2          # création d'un alias de pin0
```

2. Si on a plusieurs modules Leds utilisés, il sera plus simple d'avoir un code qui ressemble par exemple à ceci :

```
 led_verte.write_digital(1)  # allumage de la Led verte
et
 led_rouge.write_digital(0)  # extinction de la Led rouge.write
```

il suffira de définir ces deux alias en début de programme, par exemple :

```
led_verte = pin1
led_rouge = pin2
```

Remarque : attention l'utilisation de l'aliasing peut être source de problème(s) dans un programme. On va le découvrir ici sur un exemple simpliste de l'allumage ou de l'extinction d'une Led (problème qui serait vite détecté et résolu). Par contre utiliser l'aliasing sur un objet tel qu'une liste peut conduire à des bugs difficiles à repérer.


```
led = pin0          # création d'un alias de pin0
# quelque part dans le programme, on veut allumer la Led
led.write_digital(1)  # allumage de la Led
...
...
# et plus tard on veut l'éteindre :
led.write_digital(0) # extinction de la Led
# sauf que la Led était déjà éteinte avant que le programme n'exécute cette ligne !!!
```

En regardant de plus près dans le programme, il y avait ceci :

```
led = pin0          # création d'un alias de pin0
# quelque part dans le programme, on veut allumer la Led
led.write_digital(1)  # allumage de la Led
...
pin0.write_digital(0) # c'est cette ligne qui a été responsable de l'extinction « anticipée » de la Led
...
# et plus tard on veut l'éteindre :
led.write_digital(0) # extinction de la Led
# sauf que la Led était déjà éteinte avant que le programme n'exécute cette ligne !!!
```

3.2.4. Fonctions et module de fonctions

La fonction utilisée pour gérer la led, write_digital(), est une fonction généraliste pour gérer le niveau appliqué sur une sortie digitale. On pourrait souhaiter avoir une fonction dont le nom serait plus explicite, et pourquoi pas en français, pour "allumer" ou "éteindre" une Led branchée sur telle ou telle broche.

Rappel : la notion de fonction a été vue dans le Tome 1 au paragraphe 4.3.3 ; celle de module de fonctions au 4.3.4

Exercice 3.2.4_1 : Créer une fonction allumer_led(pin) qui, comme son nom l'indique allume une Led connectée à une broche « pin » de la carte Micro:Bit. De même, créer une fonction eteindre_led(pin). Utiliser ces fonctions pour faire clignoter la Led.

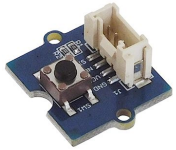
Exercice 3.2.4_1 : Créer une fonction clignoter_led(pin,th,tb) qui permet de faire clignoter une Led branchée sur une broche « pin » de la carte ; th (en millisecondes) sera la durée d'allumage (durée à l'état haut) et tb la durée d'extinction (durée à l'état bas)

Exercice 3.2.4_2 : Créer un module de fonctions que l'on nommera module_grove.py. Ce module servira à accueillir les différentes fonctions créées spécifiquement pour utiliser quelques uns des modules Grove décrits dans ce manuel.

Vérifier alors qu'un code tel que le suivant fonctionne (module Led branché sur P1/P15) :

```
from module_grove import *

verte = pin1

while True:
    clignoter_led(verte, 200, 500)
```

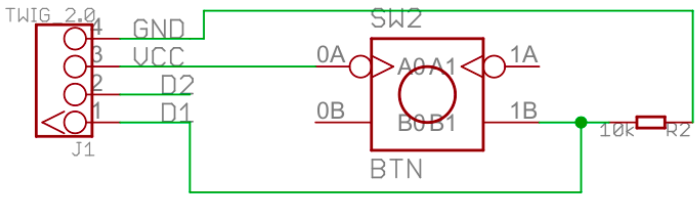
3.3 Le module bouton poussoir



Le module décrit ici a pour référence : Seedstudio : 101020003

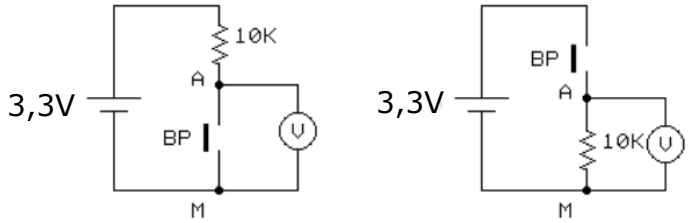
3.3.1. Présentation et mise en oeuvre

Voici le schéma (que l'on pourra considérer comme pas très pédagogique…) fourni par le fabricant :



Exercice 3.3.1_1 : On donne ci-dessous (sous une forme plus lisible) les deux possibilités de montage pour un bouton poussoir :

Montage 1 : *Montage 2 :*



1- Compléter pour chacun de ces deux montages le tableau de fonctionnement lorsque le bouton est appuyé ou relâché :

Montage 1 :				Montage 2 :		
BP :	U_{AM}(V)	Niveau logique en A		BP :	U_{AM}(V)	Niveau logique en A
Relâché				Relâché		
Appuyé				Appuyé		

2- Le bouton poussoir du module Grove suit-il le montage N°1 ou le montage N°2 ?

3- Pour connaître l'état du bouton poussoir, quelle instruction devra-t-on utiliser :

- **read_analog() ?**
- **read_digital() ?**

4- Quel sera le résultat de cette lecture :

- **lorsque le bouton est appuyé ?**
- **lorsque le bouton est relâché ?**

5- Vérifier par un petit programme affichant sur l'afficheur 5x5 Leds le résultat de cette mesure.

3.3.2. Fonctions et module de fonctions

Au paragraphe 3.2.4 nous avons commencé l'écriture d'un module de fonctions nommé « module_grove.py » pour gérer le module Led. On se propose ici d'écrire une fonction pour gérer le fonctionnement du bouton poussoir et de l'intégrer à ce module de fonctions existant :

```
#Gestion du module Bouton Poussoir :
def est_presse(pin):
    "test si le bouton connecté sur \
    l'entrée digitale (pin0, pin1, pin2...) \
    est actuellement pressé \
    Retourne 'True' si c'est le cas, sinon 'False' "
    if pin.read_digital() ==1:
        return True
    else:
        return False
```

Exemple d'utilisation avec un module bouton connecté sur P0/P14:

```
from modules_grove import *

bouton_C = pin0
while True:
    if est_presse(bouton_C):
        display.scroll('ON')
    else:
        display.scroll('OFF')
```

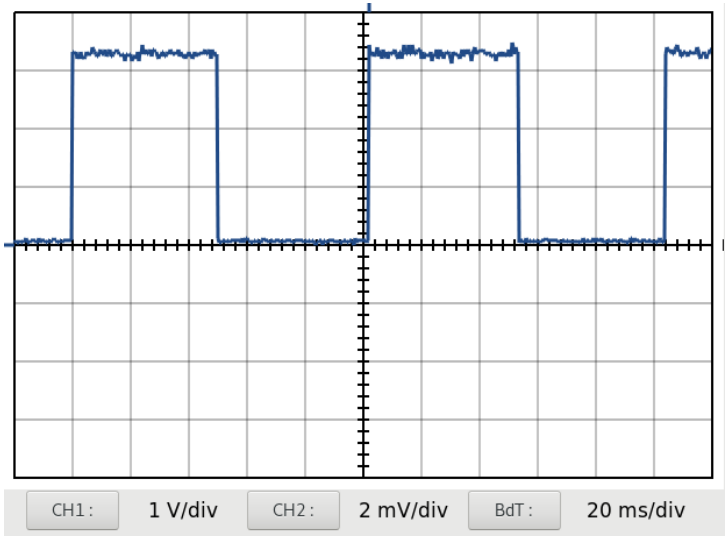

4 Solutions aux exercices

Exercice 2.4.2_1 : Modifier le code précédent de façon à obtenir une tension en créneaux de fréquence f = 10 Hz

Pour obtenir une fréquence de 10 Hz, soit une période de 100 ms, il faut générer des durées à l'état haut et à l'état bas de 50 ms chacune. On peut alors envisager le code suivant avec la fonction sleep( ) que nous avons déjà rencontrée :

```
from microbit import *

while True:
    pin0.write_digital(0)
    sleep(50)
    pin0.write_digital(1)
    sleep(50)
```

On obtient l'oscillogramme suivant :

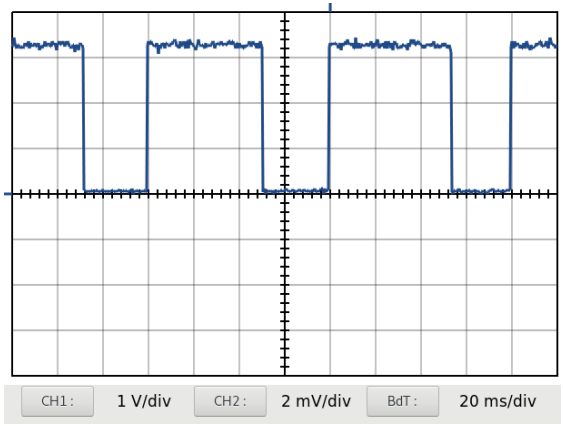


Comme on peut le voir sur cet oscilogramme, on obtient pas une durée exacte de 100 ms pour la période : ceci est dû aux temps d'exécution des différentes instructions qui constituent la boucle. Ces durées d'exécution ne nous sont pas connues ; on pourrait néanmoins essayer par tatonnements de faire une correction ad hoc du délai à injecter. L'écart relatif entre la valeur souhaitée et la valeur obtenue sera d'autant plus grand que l'on cherche à diminuer la valeur du délai dans les fonctions sleep().

Exercice 2.4.2_2 : modifier une seule des temporisations et observer le résultat à l'oscilloscope

Choisissons de modifier par exemple la durée à l'état bas :

```
from microbit import *

while True:
    pin0.write_digital(0)
    sleep(25)
    pin0.write_digital(1)
    sleep(50)
```



(pour les raisons évoquées un peu plus haut, on remarque que la période ne fait pas 50 + 25 soit 75 ms : à la mesure on relève 80 ms)

Sur le site : https://microbit-micropython.readthedocs.io/en/latest/utime.html#

est décrit un module nommé utime permettant d'accéder à l'unité de temps microseconde avec notamment la méthode

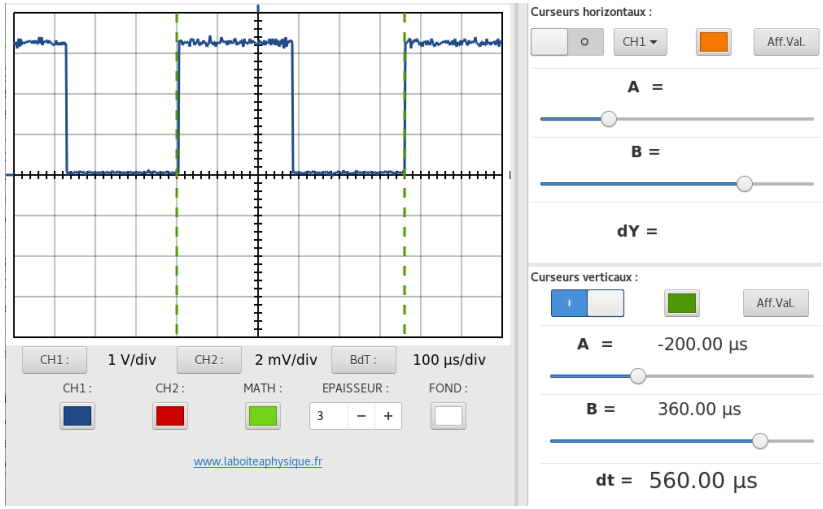
sleep_us(duree_en_µs)

que l'on se propose de tester dans le programme suivant :

```
from microbit import *
from utime import sleep_us

while True:
    pin0.write_digital(0)
    sleep_us(100)
    pin0.write_digital(1)
    sleep_us(100)
```

on commence par importer la seule fonction que l'on va utiliser dans le module utime. elle est utilisée par deux fois pour produire (en principe…) une pause de 100 µs.

Observation à l'oscilloscope… une période de 560 µs !



On a vu au paragraphe 2.4.2 que l'exécution de la boucle :

```
while True:
    pin0.write_digital(1)
    pin0.write_digital(0)
```

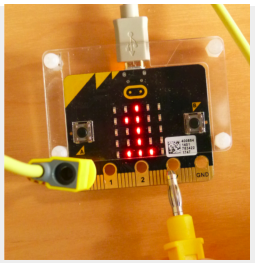
avait une durée de 135 µs. On s'attendait à en ajouter deux fois 100 µs et donc avoir un total de 335 µs et non pas 560 µs comme on l'a mesuré !

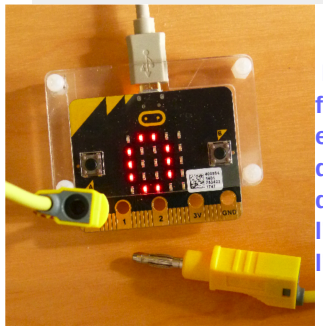
Pas d'explication à cette anomalie à part le fait que les concepteurs auraient oublié de tenir compte des durées d'exécution des instructions dans le calcul des pauses générées par les fonctions sleep ou sleep_us…

A retenir : dans l'état actuel de développement de Micropython pour la carte Micro:Bit, les timings sont loin dêtre fiables lorsque l'on touche aux durées courtes. Pour des durées longues, le problème est moindre, les durées d'exécution des instructions devenant négligeables devant ces longues durées.

Exercice 2.4.2_3 : Ecrire le code permettant d'afficher le chiffre 1 ou le chiffre 0, selon le niveau auquel se trouve la broche P0.

```
from microbit import *

display.set_pixel(2,2,9)

while True:
    val = pin0.read_digital()
    display.show(val)
    sleep(100)
```





Lorsqu'elle est positionnée en entrée (par une fonction de lecture) la broche est alors reliéee en interne à une résistance de « pull down » qui la force au niveau 0, le but étant d'éviter d'avoir un niveau indéterminé. C'est ce que l'on constate avec cette entrée P0 laissée « en l'air » mais qui reste bien au niveau 0

Exercice 2.4.3_1 : pour répondre au cahier des charges imposé :

- **on affiche une flèche qui pointe vers le bouton A, une symbolique suffisamment compréhensible au niveau de cette interface homme-machine (IHM) pour faire comprendre à l'utilisateur qu'il doit appuyer sur ce bouton. C'est plus efficace qu'une phrase du type « Appuyer sur le bouton A... » longue à faire défiler et qui pourrait ne pas être comprise par une personne non francophone.**
- **le curseur du potentiomètre est connecté sur la broche P2 donc on utilisera l'instruction pin2.read_analog()**
- **pour l'affichage formaté du résultat, revoir si nécessaire le paragraphe 5.1.2 du premier tome. On a pensé à ajouter l'unité volt pour rappeler à l'utilisateur la nature de la grandeur affichée**

```
from microbit import *

while True:
    display.show(Image.ARROW_W)
    if button_a.is_pressed():
        can = pin2.read_analog()
        U = can*3.2/1024
        display.scroll('{:.1f}'.format(U) + 'V',loop=False)
```

Pour vérifier à l'aide d'un voltmètre l'indication donnée par l'afficheur, il suffit de monter le voltmètre en parallèle sur les bornes P2 et GND

Exercice 3.2.4_1 : définition des deux fonctions pour allumer ou éteindre une Led, puis utilisation (on a branché ici une Led verte sur le connecteur P1/P15 du shield Grove):

```
from microbit import *
```

```
def allumer_led(pin):
    pin.write_digital(1)

def eteindre_led(pin):
    pin.write_digital(0)

verte = pin1  # on définit un alias pour pin1

while True:
    allumer_led(verte)
    sleep(1000)
    eteindre_led(verte)
    sleep(500)
```

Exercice 3.2.4_1 : On pourra créer la fonction demandée clignoter_led(pin,th,tb) à partir des deux fonctions précédentes :

```
from microbit import *

def allumer_led(pin):
    pin.write_digital(1)

def eteindre_led(pin):
    pin.write_digital(0)

def clignoter_led(pin,th,tb):
    allumer_led(pin)
    sleep(th)
    eteindre_led(pin)
    sleep(tb)

# mise en application pour un module Led branché sur P1/P15 :
while True:
    clignoter_led(pin1,1000,500)
```

ou plus simplement à partir de la fonction write_digital() :

```
from microbit import *
```



```
def clignoter_led(pin,th,tb):
    pin.write_digital(1)
    sleep(th)
    pin.write_digital(0)
    sleep(tb)

# mise en application pour un module Led branché sur P1/P15 :
while True:
    clignoter_led(pin1,1000,500)
```

Exercice 3.2.4_2 : Ce module de fonctions module_grove.py rassemble nos trois fonctions. On a ajouté des commentaires pour chacune d'elles. Après sauvegarde sur le PC, ce fichier doit être déposé dans le système de fichiers de la carte pour pouvoir être utilisé par un programme l'appelant :

```
# modules_grove.py
from microbit import *

# Gestion du module LED :
def allumer_led(pin):
    "allumer la Led connectée sur la sortie \
    digitale pin telle que pin0, pin1, pin2 etc"
    pin.write_digital(1)

def eteindre_led(pin):
    "éteindre la Led connectée sur la sortie \
    digitale pin telle que pin0, pin1, pin2 etc"
    pin.write_digital(0)

def clignoter_led(pin, th, tb):
    "faire clignoter la Led branchée sur la \
    sortie digitale pin (pin0, pin1, pin2 ...) \
    allumage pendant la durée th (en ms) \
    extinction pendant la durée tb (en ms)"
    allumer_led(pin)
```

```
    sleep(th)
    eteindre_led(pin)
    sleep(tb)
```

Exemple de programme utilisant le module de fonctions « module_grove.py » :

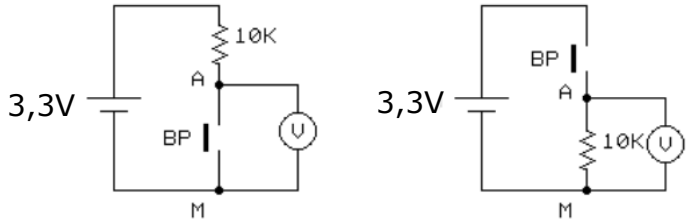
```
from module_grove import *

verte = pin1    #définition d'un alias de pin1

while True:
    clignoter_led(verte, 200, 500)
```

Les définitions de fonctions étant déportées dans le module de fonctions, le programme principal s'en trouve être d'autant réduit ce qui facilite la lecture et concentre l'attention sur le coeur du programme.

D'autres programmes pourront utiliser à leur tour ce module de fonctions : la phase d'écriture de ces programmes en sera notablement racourcie (penser néanmoins à commenter les fonctions du module lors de leur création pour qu'elles puissent être rapidement réinvesties quelques semaines ou mois plus tard !)

Exercice 3.3.3_1 : On donne ci-dessous (sous une forme plus lisible) les deux possibilités de montage pour un bouton poussoir :

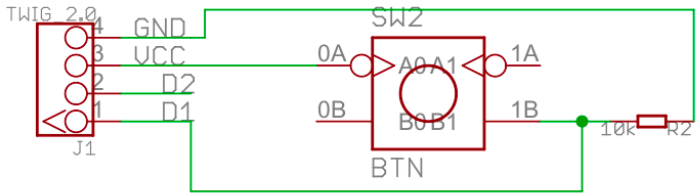
Montage 1 : *Montage 2 :*



1-

Montage 1 :				Montage 2 :		
BP :	U_{AM}(V)	Niveau logique en A		BP :	U_{AM}(V)	Niveau logique en A
Relâché	**3,3**	**1**		Relâché	**0**	**0**
Appuyé	**0**	**0**		Appuyé	**1**	**1**

2- Le bouton poussoir du module Grove suit-il le montage N°1 ou le montage N°2 ?



L'une des bornes de la résistance (R2) est reliée à la masse, ce qui correspond au montage N°2

3- Pour connaître l'état du bouton poussoir, quelle instruction devra-t-on utiliser :

Les deux instructions pourraient être utilisées puisque les deux détecteront une grandeur différente selon que le bouton est appuyé ou relâché, mais le comportement d'un bouton poussoir étant binaire, on utilisera une lecture numérique donc l'instruction read_digital (qui est d'ailleurs beaucoup plus réactive que la lecture analogique read_analog()

4- Quel sera le résultat de cette lecture :

- **lorsque le bouton est appuyé : 1**
- **lorsque le bouton est relâché : 0**

5- Vérification :

```
from microbit import *

while True:
    display.show(pin0.read_digital())  #module connecté à P0/P14
    sleep(100)
```


5 Annexes

Schéma 1 : Schéma d'utilisation du microcontrôleur NORDIC NRF51822

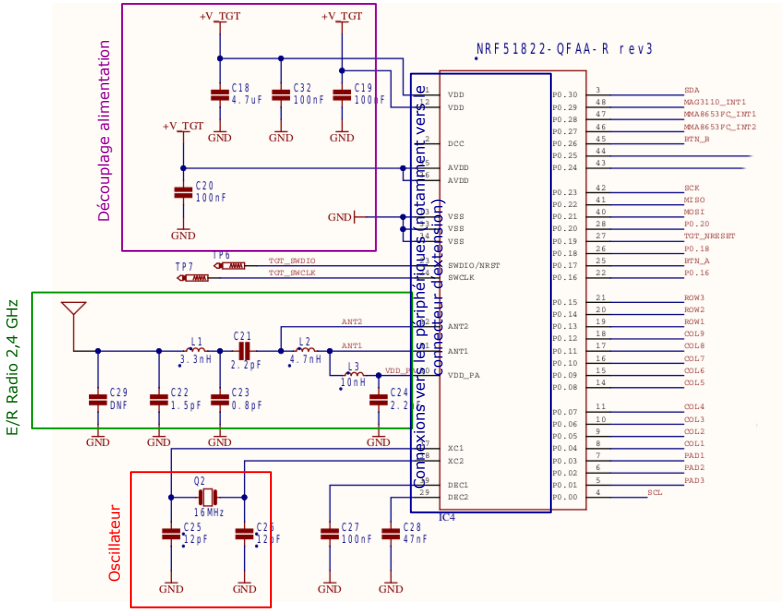


Schéma 2 : Le connecteur d'extension

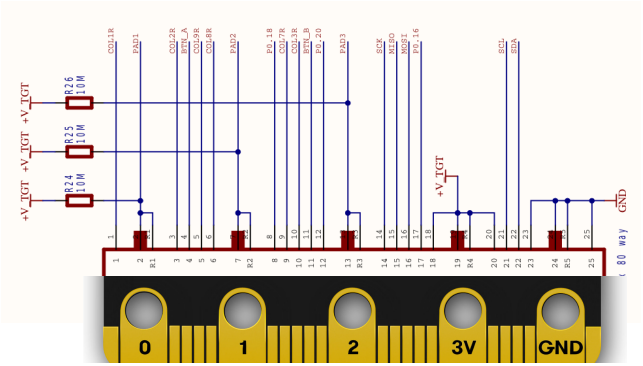


Schéma 3 : Interface

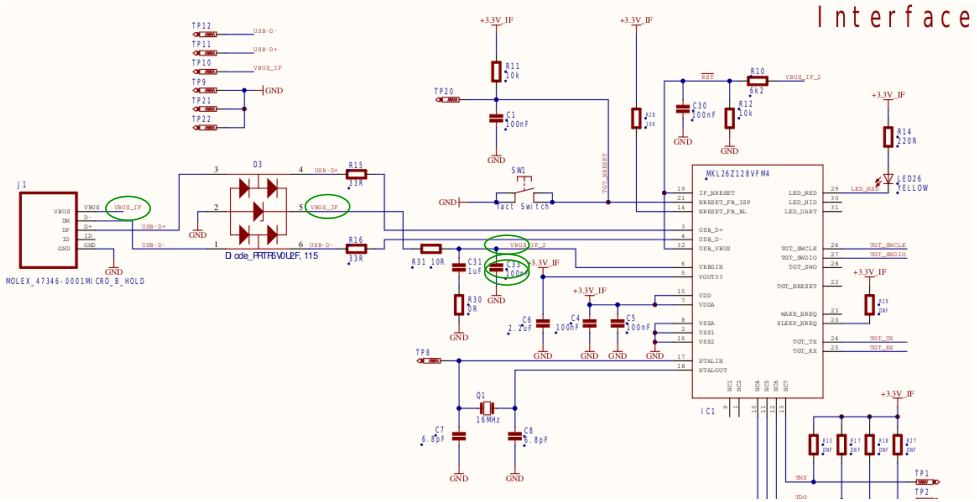


Schéma 4 : Alimentation (suite)

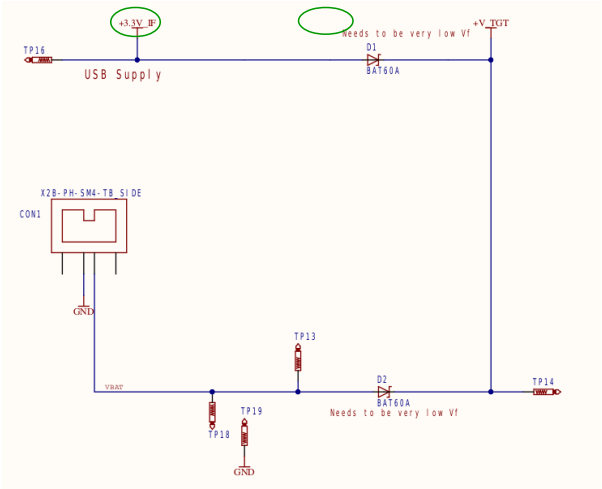
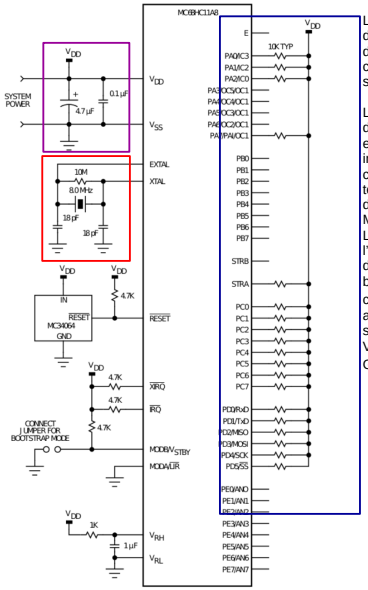


Schéma 5 : D'hier à aujourd'hui...



Le schéma ci-contre représente un microcontrôleur qui a eu son heure de gloire dans les années 90 (le Motorola 68HC11). . En comparant avec le schéma d'utilisation du Nordic ci-dessus, on retrouve des similarités ! Il y a eu cependant de grosses évolutions dans les fonctionnalités disponibles mais surtout au niveau des consommations électriques.

La photo ci-dessous met en comparaison ces deux microcontrôleurs d'un point de vue visuel : tous deux disposent d'une cinquantaine de broches (48 pour l'un et 52 pour l'autre). Le 68HC11 n'était pas si facile que cela à souder, alors inutile d'espérer le faire de façon manuelle avec le Nordic ! Tous ces composants de surface (CMS) se soudent dans des fours à refusion, une technique coûteuse pour un amateur, d'où l'essor de toutes les cartes de développement prêtes à l'emploi, telles qu'Arduino ou pour nous ici la BBC Micro:Bit

La réduction des dimensions a d'autres incidences notamment au niveau de l'évacuation de la chaleur (par la surface ou par les broches) et donc au niveau des limites électriques admissibles. Remarquer sur le schéma 1 qu'il y a deux broches V_{DD} et broches de masse V_{SS} alors qu'il n'y a qu'un seul V_{DD} ou V_{SS} sur cet ancien microcontrôleur: ceci certainemente dû à la finesse des broches actuelles qui ne permettent pas de véhiculer de « grosses » intensités. Or ce sont les lignes d'alimentation qui véhiculent les plus fortes intensités, et la ligne V_{SS} encore plus que V_{DD} puisque du courant peut revenir par les broches du GPIO mises en entrée