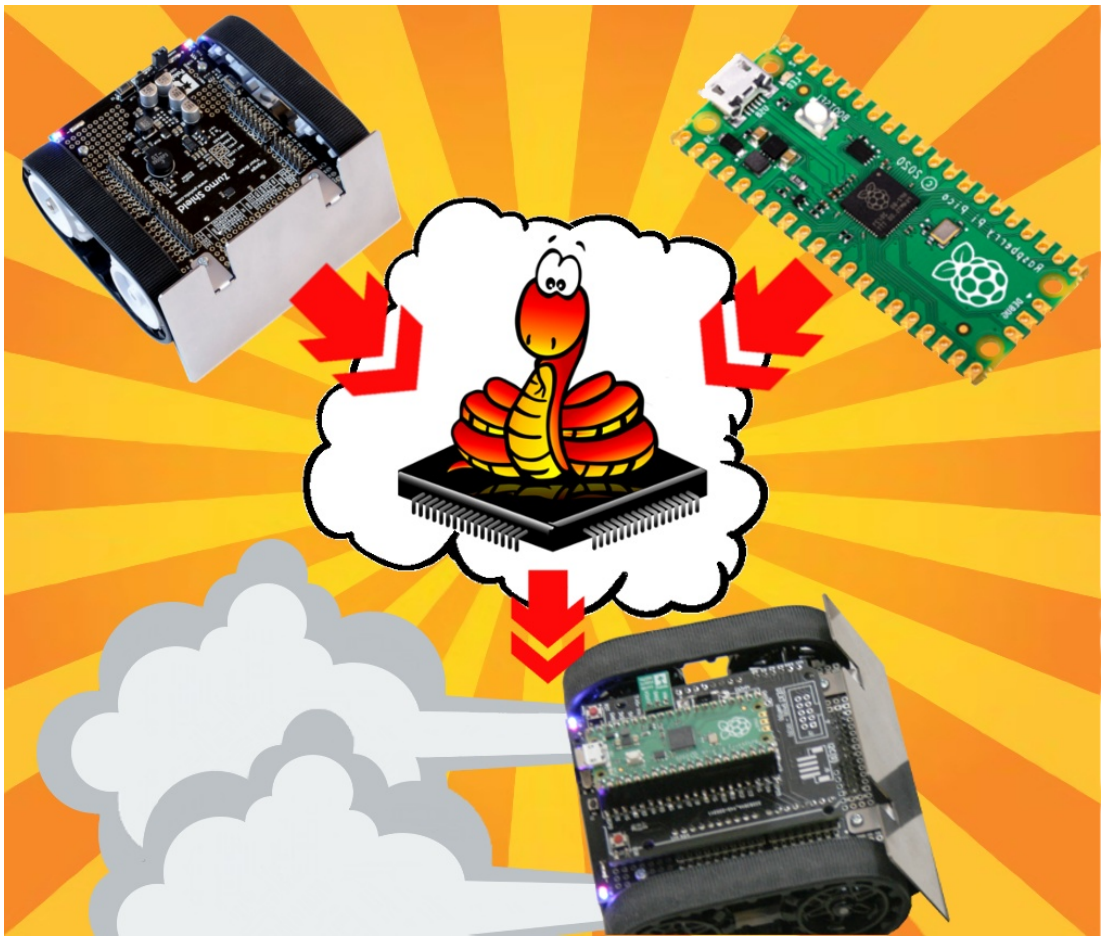
Pico, Zumo Robot et MicroPython

Programmer le Zumo Robot avec Python pour Microcontrôleur



© Tous droits réservés – 2023 – MC Hobby Srl

MicroPython et Raspberry-Pi Pico

Table des matières

© Tous droits réservés – 2023 – MC Hobby Srl

© Tous droits réservés – 2023 – MC Hobby Srl

1.Avant-propos

Avant de se lancer à corps perdu dans le projet Pico-Zumo, il convient d'avoir quelques notions concernant le microcontrôleur RP2040 ainsi que sur MicroPython.

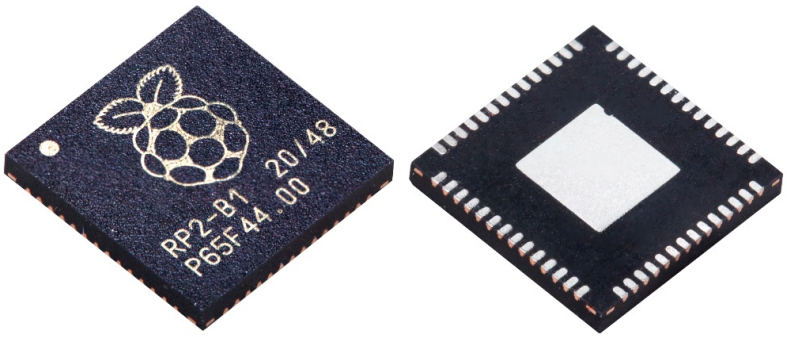
Ce chapitre va donc couvrir assez largement l'utilisation du Raspberry-Pi Pico.

Comprendre son environnement de travail est tout aussi important que comprendre les raccordements entre le Pico et le Zumo Robot.

2.Microcontrôleur RP2040

Le Raspberry-Pi Pico, dont il est question dans cet ouvrage, exploite le microcontrôleur RP2040 conçu par la fondation Raspberry-Pi.

La conception de ce premier microcontrôleur est une grande réussite et surpasse, sur bien des points, les concurrents du marché.



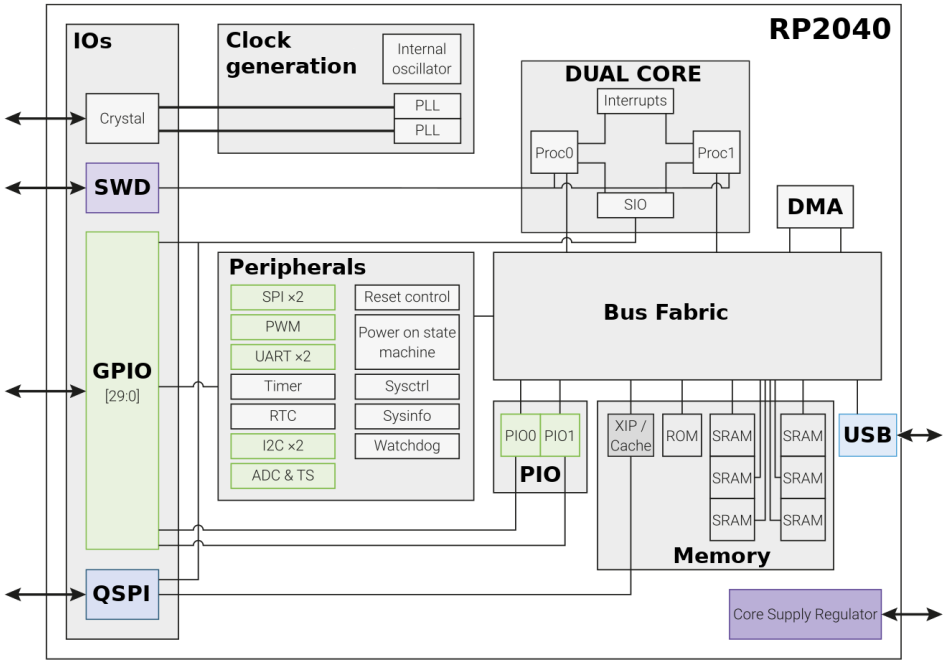
02RI01 – Fig : Microcontrôleur RP2040

Pour commencer, ce microcontrôleur dispose d'un double cœurs, ce qui n'est pas commun pour un composant à 1,50 EUR.

Le RP2040 est un microcontrôleur haute performance cadencé à 133 Mhz disposant de tous les périphériques standards d'un microcontrôleur (bus I2C, bus SPI, bus série/UART, Timer) tout en étant accompagné de fonctionnalités très avancées (PIO).

Le graphique ci-dessous reprend les différents blocs fonctionnels du RP2040.

© Tous droits réservés – 2023 – MC Hobby Srl



02RI02 - Fig : blocs fonctionnels du RP2040.

Voici ce que ce graphe nous apprends :

•Le **bloc mémoire** (*memory*) reprend 6 banques de mémoire RAM pour un total de 264 Kio. Ce qui est très conséquent par rapport à un Arduino UNO qui ne dispose que de 2 Kio. Le bloc mémoire contient aussi un élément ROM (mémoire en lecture seule) contenant, entre autre, le bootloader UF2 permettant de téléverser un nouveau firmware sur le Pico via USB.

•Le **bloc USB** indique que le RP2040 dispose d'un support USB natif ! Celui-ci est utilisé pour téléverser un nouveau firmware (via le bootloader UF2). Le bloc USB permet aussi au RP2040 d'établir une connexion USB-Série, émuler un périphérique HID (clavier/souris) et même de se comporter comme un hôte USB (pour y connecter un périphérique USB).

•Le **bloc Dual Core** présente les deux cœurs disposant d'un accès indépendant au BusFabric ! Deux scripts MicroPython peuvent s'exécuter en parallèles en toute indépendance. L'élément SIO (*Software control gpIO*) permet aux programmes utilisateurs de manipuler les broches d'entrées/sorties (les GPIOs).

•Le **bloc DMA** (*Direct Memory Access*) présent dans tous les systèmes évolués, le module DMA permet à un périphérique de prendre le contrôle des bus de données et d'adresse. Grâce à cela, un périphérique peut stocker/lire directement des données dans/depuis la mémoire sans passer par le processeur. L'exemple microcontrôleur typique est l'échantillonnage d'un signal qui stocke directement les données dans la mémoire.

•Le **bloc périphériques** (*Peripherals*) reprend les périphériques habituels d'un microcontrôleur. Le projet Pico-Zumo de cet ouvrage utilise plusieurs de ces périphériques comme le bus I2C, signal PWM, entrée analogique (ADC).

•Le **bloc PIO** (*Programmable IO*) est un périphérique spécialisé RP2040. Il est composé de deux entités PIO0 et PIO1 reprenant deux machines à états permettant d'exécuter, **de façon autonome**, des instructions rattachés aux entrées/sorties. Ces blocs PIO permettent de créer de nouveaux périphériques spécialisés sans faire intervenir les deux coeurs d'exécution. Par exemple : PIO est utilisé pour l'émission des données d'un ruban NeoPixel (très exigeant sur le timing), la création d'un port

© Tous droits réservés – 2023 – MC Hobby Srl

parallèle, la génération d'un signal VGA. PIO permet de mettre en place des protocoles matériels sans consommer, ni bloquer, des ressources processeurs.

•Le **bloc GPIO** (*General Purpose Input Ouput*) prend en charge les 30 broches d'entrées/sorties réparti autour du microcontrôleur. Ce bloc est manipulé par les blocs : périphériques, PIO et SIO.

•Le **bloc Bus Fabric** est un des points fort du RP2040 destiné aux applications industrielles. Le Bus Fabric permet de connecter un bus données (I2C, SPI, UART, etc) sur l'une ou l'autre broche du RP2040. Sur les microcontrôleurs d'entrée de gamme, la position des bus est figée. Le bus Fabric est comme un tableau d'inter-connexion géant au coeur du microcontrôleur.

•Le **bloc QSPI** (*Quad* SPI) est un bloc spécialisé dans la communication avec les modules de mémoire flash. Comme l'indique le graphique, cette mémoire Flash est externe au microcontrôleur. Un bus *Quad* SPI est capable d'utiliser peut utiliser jusque 4 lignes de données permettant ainsi de lire plusieurs bit de la mémoire externe en un seul cycle d'horloge. Le RP2040 ne disposant pas de mémoire flash interne (seul la ROM est interne), le bus *Quad* SPI à haut débit permet de raccorder la mémoire de stockage sur le RP2040 (stockage en Flash).

•Le **bloc SWD** (*SoftWare Debug*) est une interface matérielle permettant de déboguer les programmes utilisateurs en cours d'exécution. Cette fonctionnalité avancée concerne surtout les programmes écris en C/C++. MicroPython offre d'autres possibilités de débogage.

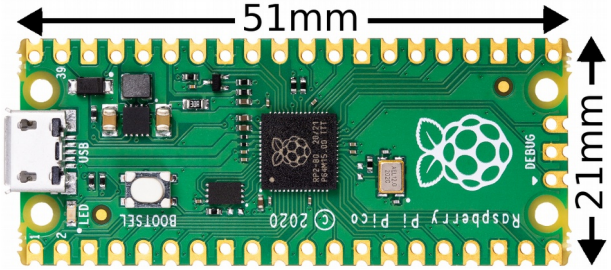
•Le **bloc Quartz** (*Crystal*) également reporté sur les broches d'entrées/sorties, ce dernier sera utilisé pour y brancher le crystal générant le signal d'horloge à 133 Mhz.

3.Présentation du Pico

3.1.Survol du Pico

Le Raspberry-Pi Pico est la plateforme de référence de la fondation Raspberry-Pi pour le microcontrôleur RP2040.

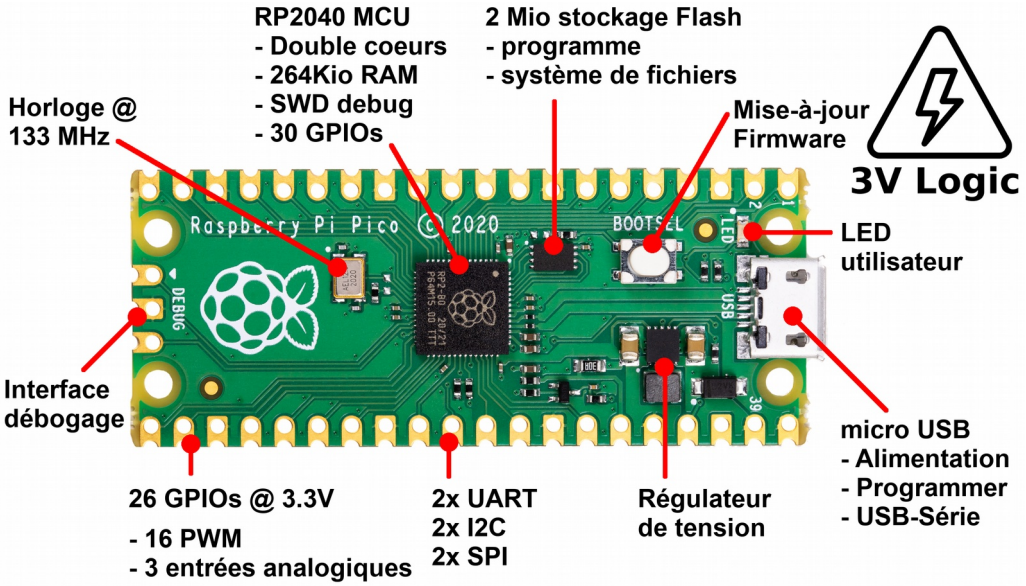
Cette plateforme représente l'implémentation minimale d'un RP2040 avec les seuls composants essentiels à son bon fonctionnement et tout en maintenant le Pico sous la barre des 5 EUR.



02RI03 – Fig : Raspberry-Pi Pico

Placée sur une carte de 5 x 2 cm, la plateforme Pico supporte les développements en C/C++, Arduino, MicroPython, CircuitPython, Rust, Ada, etc.

© Tous droits réservés – 2023 – MC Hobby Srl



02RI04 – Fig : éléments du Raspberry-Pi Pico

3.1.1.Port micro USB

Le connecteur USB est généralement utilisé pour alimenter la plateforme en 5V.

Brancher le Pico sur un ordinateur permet de :

•téléverser un nouveau programme/firmware sur la carte (croquis Arduino ou le firmware MicroPython). Il est parfois nécessaire d'activer le mode de programmation DFU (*Device Firmware Updade*) en pressant le bouton **bootsel** sur le Pico.

•d'obtenir une liaison série via la connexion USB (terminal Arduino ou ligne de commande MicroPython/REPL).

•d'émuler un périphérique USB (ex : périphérique HID comme clavier et souris)

•accessoirement : alimenter la plateforme en 5V.

Brancher un périphérique USB sur le Pico :

•le Pico se comporte alors un hôte USB permettant d'acquérir les données envoyées par le périphérique USB (ex : un clavier USB).

•Le Pico doit être alimenté en 5V par l'intermédiaire de sa broche VUSB de sorte a aussi alimenter le périphérique USB.

3.1.2.Broches d'entrées/sorties

Le Pico expose 26 broches GPIOs pouvant être utilisée en entrée ou en sorties.

*Le Pico est un microcontrôleur en logique 3.3V et n'est **pas tolérant 5V** ! Il ne faut, en aucun cas, dépasser la tension de 3.3V sur l'une des broches GPIOs du Pico.*

Seize (16) de ces broches peuvent être utilisés pour générer un signal PWM (*Pulse Width Modulation* ou modulation la largeur d'impulsion). Dans le monde Arduino, cette fonctionnalité est abusivement appelée « sortie analogique » car il permet de moduler la puissance de sortie de signal (par exemple, la luminosité d'une LED) faisant passer la fonctionnalité PWM pour une pseudo sortie analogique.

© Tous droits réservés – 2023 – MC Hobby Srl

Trois (3) broches sont des entrées analogiques, elles permettent de lire une tension analogique entre 0 et 3.3V.

Une LED utilisateur, placée juste à côté du port USB, est raccordée sur le GPIO (25) du Pico. Attention, il n'en va pas de même pour le Pico Wireless.

Enfin, les GPIOs support aussi différents bus de données (SPI, I2C, UART) utilisé pour communiquer avec des périphériques plus avancés comme des afficheurs LCD/TFT, capteurs avancés et autre.

A noter que le GPIO 25 n'est pas la seule broche non accessible sur les connecteurs de la carte (dite « *Not breakout* »). Parmi ces GPIOs, il y a :

• GPIO 24 : utilisée pour détecter la présence d'une tension sur le port micro USB. Donc si VBUS = 5V.

•GPIO 29 : permet de relever la tension sur la broche VSYS. Le GPIO 29 est l'entrée analogique 3 (ADC3) et la tension est relevée par l'intermédiaire d'un pont diviseur de tension (ADC3 = VSYS/3). Cette broche est particulièrement utile pour surveiller la tension d'un accumulateur LiPo ou bloc pile utilisé pour alimenter le Pico.

•GPIO 23 : utilisé pour activer le mode d'économie d'énergie du régulateur d'alimentation en plaçant le GPIO au niveau bas. Le mode d'économie ne peut être activé que lorsque le Pico consomme peu. En cas de forte consommation, l'état du GPIO 23 est ignoré.

3.1.3.Détails sur le RP2040

Déjà aborder en début de chapitre, le microcontrôleur RP2040 apporte à MicroPython (tout comme aux autres langages) des caractéristiques hors du commun pour un prix dérisoire. Si le RP2040 offre les services traditionnels d'un microcontrôleur, il apporte aussi toute sa puissance à la plateforme Raspberry-Pi Pico avec quelques caractéristiques remarquables :

•**264 Kio de mémoire RAM** : 132 fois plus qu'un Arduino UNO, cette quantité de mémoire est très confortable pour exécuter tout script Python et bibliothèques.

•**Mémoire Flash externe** : le microcontrôleur est conçu pour être équipé d'une mémoire Flash externe (voir point suivant). De nombreuses plateformes à base de RP2040 utilise cette particularité pour proposer une offre varié en matière de stockage.

•**Double coeurs** : permettant l'exécution en parallèle de scripts Python (via le module _thread) qui, au contraire de Python sur PC, offre une vraie exécution *multithread* des deux processus. Il faut donc prévoir des mécanismes de synchronisation lors d'échanges de données entre les deux threads.

•**PIO** : Programmable IO est une fonctionnalité avancée du RP2040 permettant créer des entrées/sorties autonomes (fonctionnant indépendamment des deux coeurs du RP2040). PIO permet d'implémenter des protocoles de communication matériel à l'aide d'un ensemble de quelques instructions spécialisées. Cette fonctionnalité n'est pas exploité dans cet ouvrage.

•**Support natif USB** : supportant USB 1.1 à la fois comme client USB et hôte USB.

•**Bus matériels** : les bus SPI, I2C, UART sont disponibles comme sur tous les microcontrôleurs classiques. Pico dispose de plusieurs de ces bus accessible via le *Bus Fabric*. Ces bus sont relocalisables sur différentes broches du Pico (voir description du Bus Fabric en début de chapitre). A noter que **PIO** permet, en autre, de créer de nouveaux types de bus.

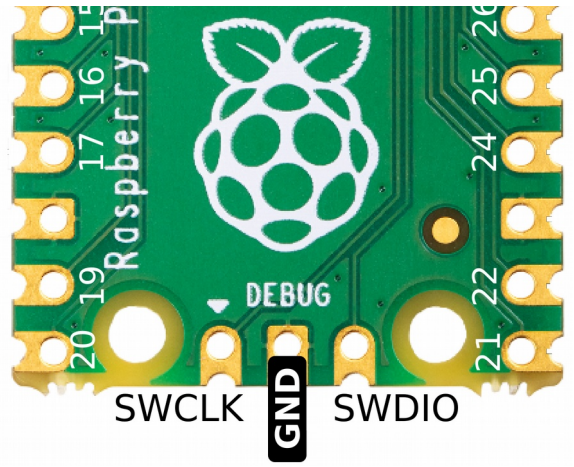
© Tous droits réservés – 2023 – MC Hobby Srl

•**Port de débogage** : s'utilise en conjonction avec OpenOCD pour déboguer du code en C/C++.

•**Broche / Bouton Boot** : en cas de blocage complet du microcontrôleur, il est possible d'activer le mode Boot (presser le bouton en mettant la carte sous tension). Dans ce mode, la carte Pico apparaît comme un lecteur USB où il est possible de téléverser un nouveau firmware par simple glissé/déposé. Ce mode est exploité par les développeurs C/C++ pour placer un nouvelle version de leur programme (firmware) sur la carte Pico. Sous MicroPython, le firmware est « MicroPython » lui-même et ne doit être installé qu'une seule fois sur le Pico.

3.1.4.L'interface de débogage

Un peu a l'écart du brochage d'entrée/sortie, il y a un port port 3 broches portant la mention « DEBUG » sur la sérigraphie. Ce port de débogage est composé d'un signal de données SWDIO (*SoftWare Debug Input/Output*), d'un signal d'horloge SWDCLK (*SoftWare Debug ClocK*) et d'une masse.



02RI05 – Interface de débogage

Ce connecteur permet de déboguer des programmes écrit en C/C++ à l'aide d'OpenOCD (*Open On Chip Debugger*). La fondation recommande de souder un connecteur orienté vers le haut si cette interface doit être utilisée.

Le débogage des scripts python n'exploite pas le port «DEBUG» mais exploite plutôt les avantages de la session interactive Python (dite REPL). La session interactive permet de voir les messages utilisateurs, les exceptions et erreurs Python ainsi que le détail de la pile d'appel.

3.1.5.La mémoire Flash

Au plus stricte, la mémoire Flash sert à stocker le *firmware*. C'est le cas lors du transfert d'un firmware produit avec un compilateur C/C++.

Dans le cas de MicroPython, cette mémoire flash à une double usage :

•**Le firmware MicroPython** : machine virtuelle qui exécute les scripts Python.

•**Le système de fichiers MicroPython**.

Le système de fichiers MicroPython est basé sur LittleFS (https://github.com/littlefs-project/littlefs), un système de fichiers spécialement conçu pour les systèmes à faible ressources nécessitant une excellente résilience et tolérance au coupures d'alimentation.

© Tous droits réservés – 2023 – MC Hobby Srl

Le système de fichiers MicroPython est manipulé par le firmware MicroPython et permet de stocker les scripts Python ainsi que tout autres types de fichiers de données (comme images, fichiers de données, fichier de configuration, etc). MicroPython est capable de manipuler des fichiers en lecture et en écriture.

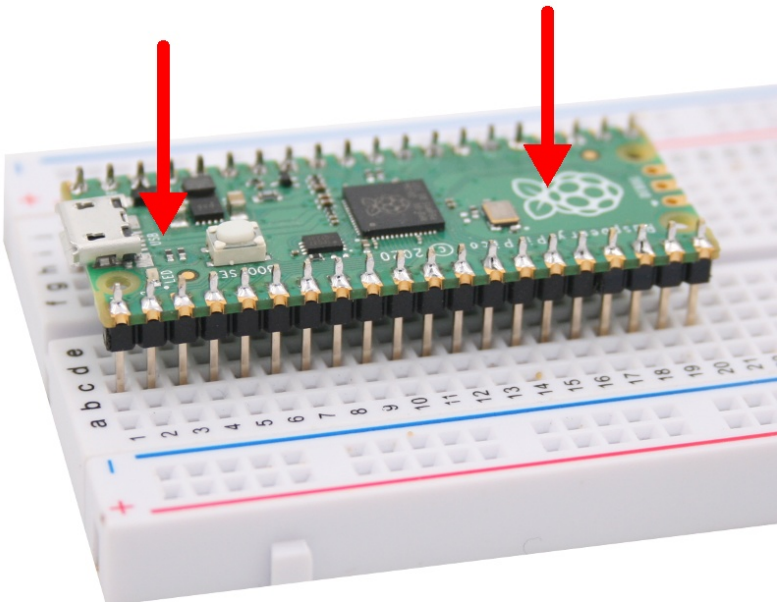
Sur un Raspberry-Pi Pico, le **module flash est programmé avec le firmware MicroPython**.

Des 2 Mio de mémoire Flash, 1.3 Mio est disponible pour le système de fichiers MicroPython. Cet espace disponible dépend directement de la taille du firmware MicroPython (qui peu sensiblement varier d'une version à l'autre).

3.1.6.Empattement 2,54mm

Le Pico, comme de nombreux produits orienté vers les Makers et le grand publique, dispose d'un empattement standard de 2.54mm (espace entre deux broches consécutives).

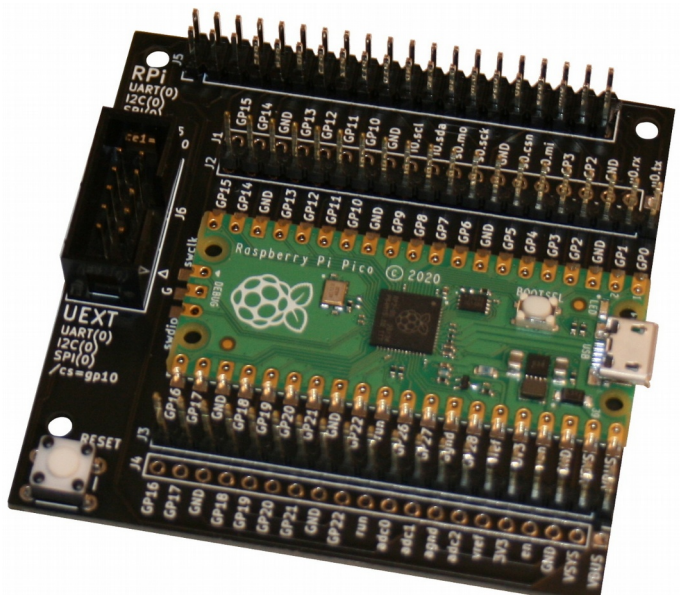
Cela signifie qu'il est possible de placer le Pico sur une plaque de prototypage sans soudure (dit « *breadboard* ») ou perfboard (plaque pré-perforée).



02RI31 – Pico sur un breadboard demi-taille

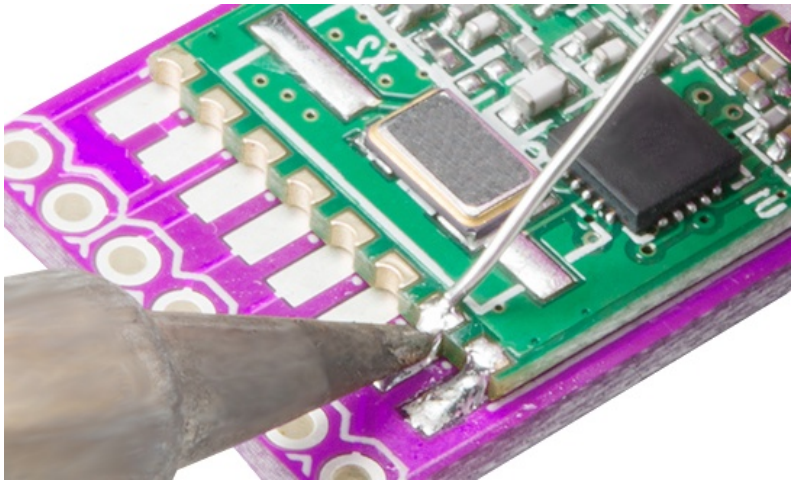
La forme singulière des pastilles de connexion est dites « *castellate holes* ». Cette forme particulière permet de souder directement la carte Pico sur une carte hôte sans utiliser de connecteurs.

© Tous droits réservés – 2023 – MC Hobby Srl



02RI07 – Utilisation des *castellate holes* du Pico

Pour une soudure pico-sur-carte, la carte hôte doit proposer des pastilles de connexion rectangulaires légèrement plus longue que celle du Pico. De la sorte, il est possible de déposer de la soudure à la jonction du connecteur *castellated* du Pico, cette soudure mouille alors les deux deux éléments offrant ainsi un excellent contact électrique.



02RI08 – Souder une connexion *castellated*

3.2.Installer MicroPython / En cas de problème

Il peut arriver que la carte Pico soit livrée sans firmware (ou sans firmware MicroPython).

Il peut aussi arriver qu'un script Python en cours d'exécution rende la plateforme MicroPython inaccessible. Si cela est un cas relativement rare, il peut néanmoins arriver.

Si un redémarrage du Pico ne permet pas de récupérer un système fonctionnel alors il faudra envisager une méthode plus radicale.

Dans ces deux cas de figures il faut installer/réinstaller le firmware MicroPython sur le Pico (ou Pico Wireless).

Voici les différentes étapes de cette opération.

© Tous droits réservés – 2023 – MC Hobby Srl

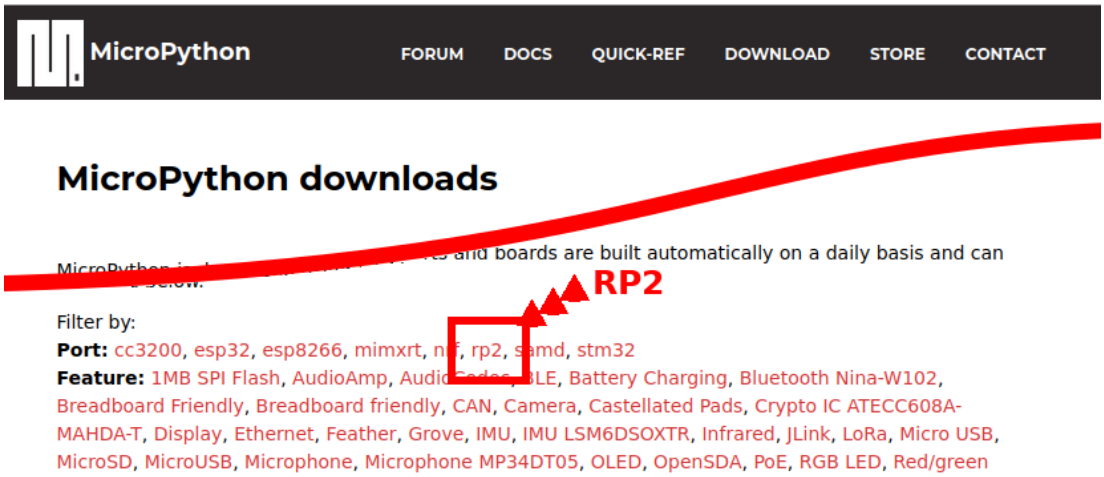
3.2.1.Télécharger le firmware MicroPython

En toute première étape, il est nécessaire de télécharger la toute dernière version du FirmWare MicroPython pour le Raspberry-Pi Pico. Les firmwares sont accessibles dans la section « *Download* » (téléchargement) du site http://micropython.org .

Pour un Pico Wireless, il faut bien entendu choisir la version « wireless » du firmware.

02RI20 – Naviguer vers la section téléchargement du Raspberry-Pi Pico

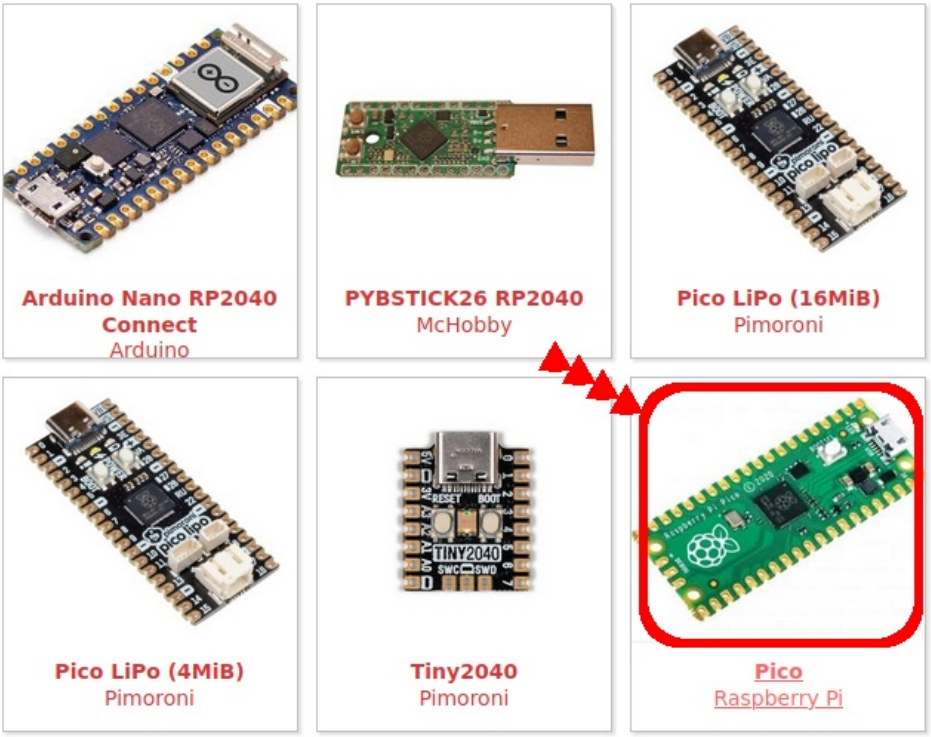
Dans la section des téléchargements, réduire la sélection des plateformes supportées en sélectionnant le portage « **rp2** » .



02RI21 – Sélection du portage (Port) RP2

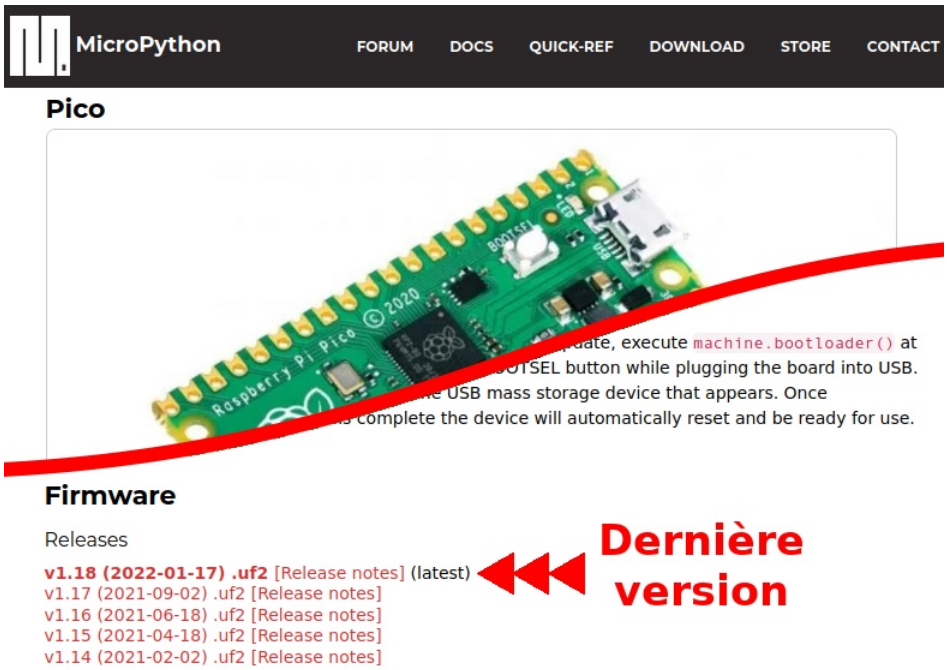
Parmi les plateformes RP2040 sélectionnées pour le filtre sur le portage « RP2 », sélectionner l'entrée correspondant au « **Raspberry Pi Pico** ».

© Tous droits réservés – 2023 – MC Hobby Srl



02RI22 – Sélection de la plateforme Raspberry-Pi Pico

La page des firmwares pour Pico présente une liste des derniers firmwares disponibles pour le Raspberry-Pi Pico. Au moment de l'écriture de ces lignes, le dernier firmware disponible est la version 1.18.



02RI23 – Liste des firmwares

Sur les plateformes Pico les fichiers Firmware porte l'extension de fichier « **uf2** ».

Cliquer sur le lien permet de télécharger le firmware en question. Il sera disponible dans la zone de téléchargement du navigateur Internet.

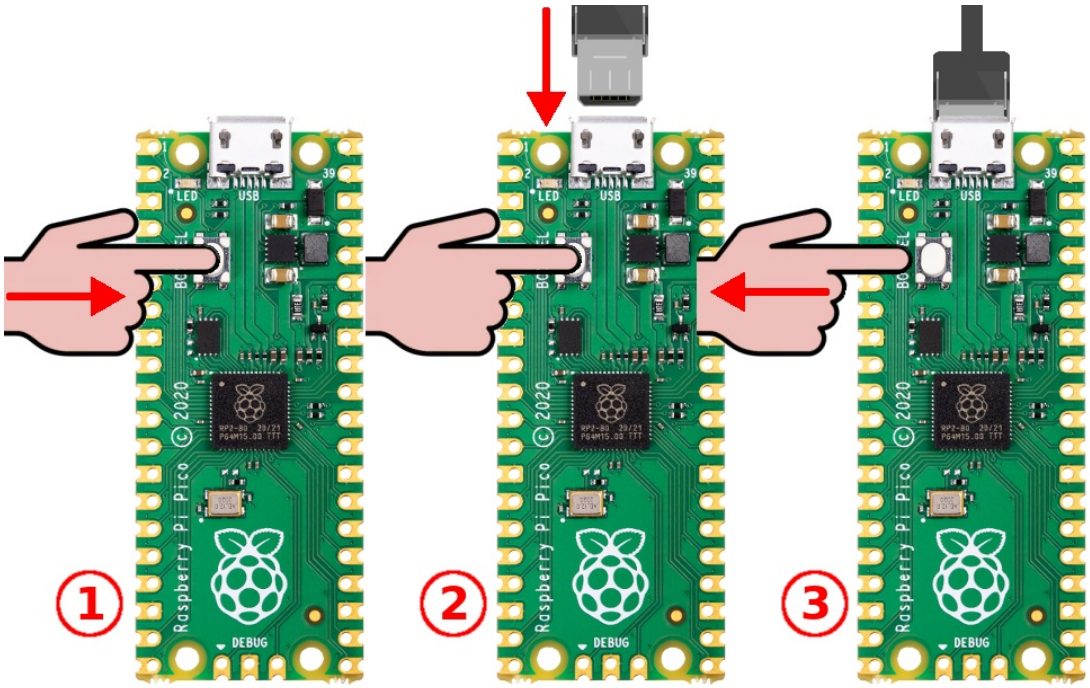
© Tous droits réservés – 2023 – MC Hobby Srl

Le FirmWare MicroPython pour le Raspberry-Pi Pico (donc le microcontrôleur RP2040) peut également être utilisé sur toutes les plateformes à base de RP2040. Il ne se cantonne pas uniquement au Pico, c'est là un firmware générique.

3.2.2.Activer le mode Boot

La deuxième étape consiste à placer manuelle le Pico dans le mode « Boot » (mise-à-jour du *FirmWare*).

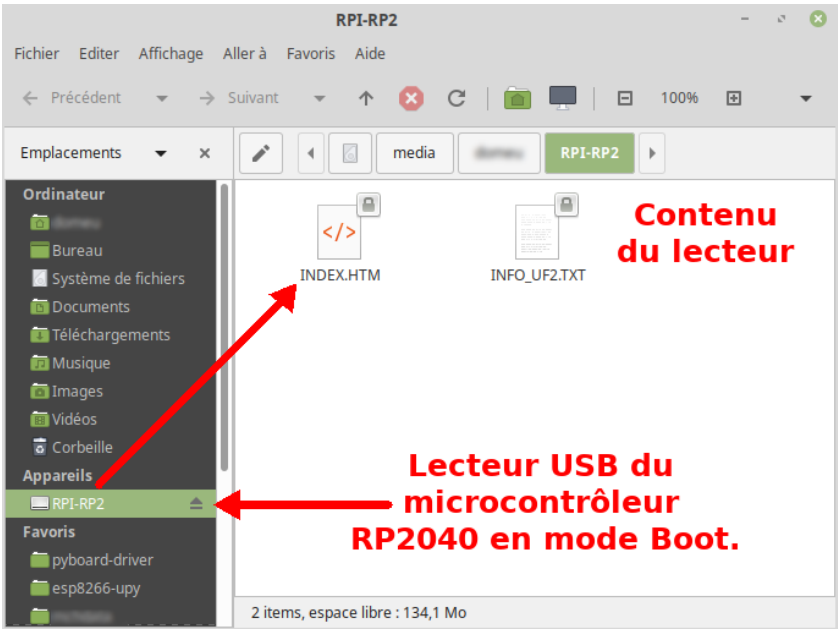
Pour cela, presser le bouton « BootSel » et le maintenir enfoncer pendant que l'on branche le Pico sur un ordinateur.



02RI24 – Activation du mode Boot.

Une fois le mode « Boot » activé, la carte expose un lecteur USB nommé « RPI-RP2 » sur l'ordinateur. Cette opération peut prendre quelques secondes.

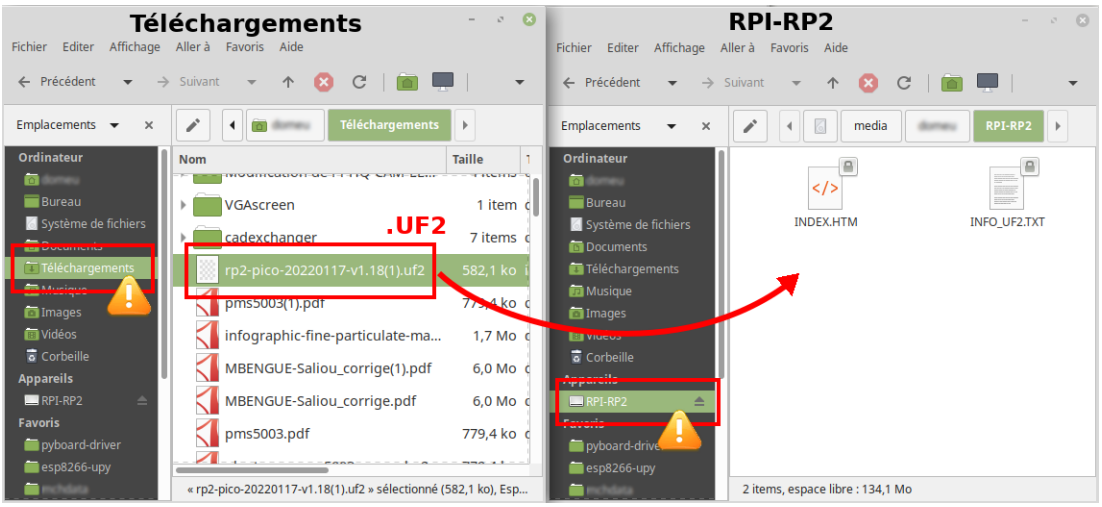
© Tous droits réservés – 2023 – MC Hobby Srl



02RI25 – Le lecteur USD d'un Pico en mode Boot.

3.2.3.Copier MicroPython sur le Pico

Il ne reste plus qu'à glisser/déposer le firmware MicroPython téléchargé (fichier UF2 disponible dans le dossier de téléchargement) dans le lecteur « RPI-RP2 » ouvert par le microcontrôleur.



02RI26 – Copier le firmware MicroPython sur le Pico

Après la copie du fichier uf2 (le *firmware*) sur le lecteur USB correspondant au Pico, opération qui peut prendre plusieurs dizaines de secondes, le Raspberry-Pi Pico redémarre en mode normal.

En conséquence, le disque disparaît et le firmware MicroPython est déjà en cours de fonctionnement sur la carte Pico.

3.3.Effacer la Flash / En cas de problème

Réinstaller la firmware MicroPython n'efface pas le système de fichiers. Par conséquent un fichier `boot.py` ou `main.py` bogué est de nouveau exécuté même après une réinstallation de firmware MicroPython.

© Tous droits réservés – 2023 – MC Hobby Srl

Dans pareil cas, la seule solution est d'utiliser un fichier firmware détruisant le système de fichiers MicroPython.

Le fichier `flash_nuke.uf2` pour RP2040 permet justement d'effacer le système de fichiers MicroPython. Il suffit de le copier sur votre Pico de la même façon que le firmware MicroPython.

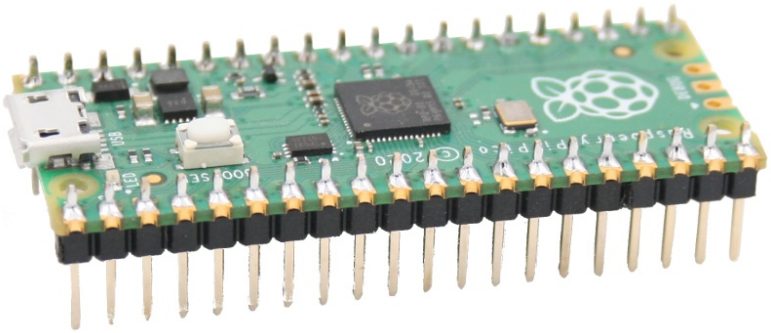
Le fichier flash_nuke.uf2 est accessible dans les Forum de MicroPython et depuis le serveur de MCHobby.

https://arduino103.blogspot.com/2023/04/micropython-effacer-le-systeme-de.html

Ensuite, replacer le Pico en mode Boot et reflasher le firmware MicroPython. Cette fois, le système de fichier sera vierge.

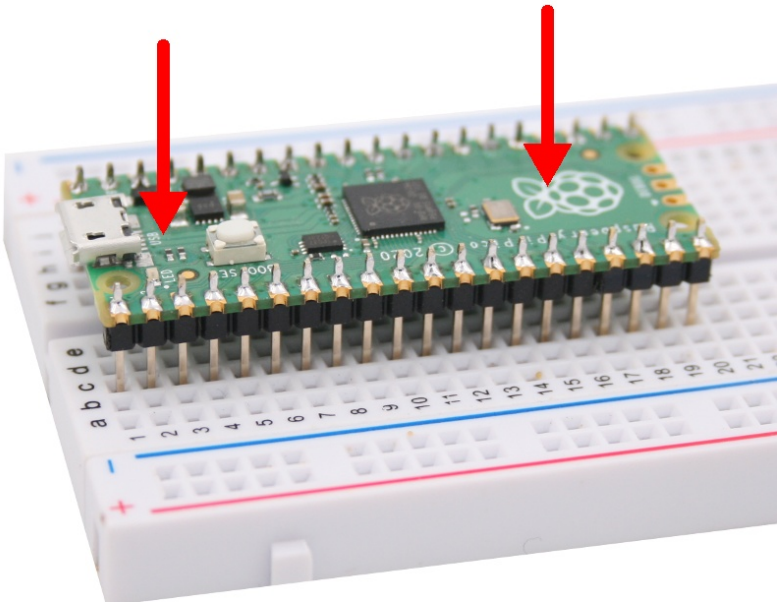
3.4.Quel connecteurs pour le Pico ?

Les recommandations de la fondation Raspberry-Pi est d'utiliser des connecteurs mâles (PinHeader, empattement 2.54mm) que l'on soude sous la carte.



02RI30 – Connecteurs sur le Pico

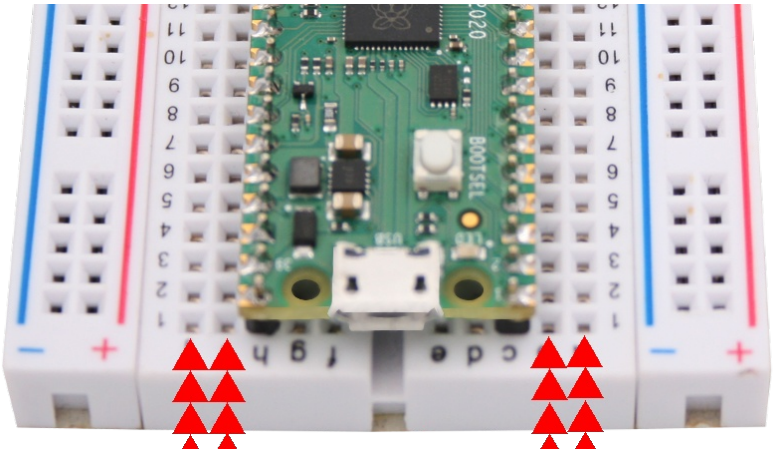
Cela permet de brancher facilement et rapidement le Raspberry-Pi Pico sur une carte de prototypage ou sur une plaque de prototypage sans soudure (un *breadboard*).



02RI31 – Pico sur un breadboard demi-taille

Une fois en place sur la plaque de prototypage, il reste de deux points de connexions de chaque côté du Pico pour y brancher les fils de prototypages.

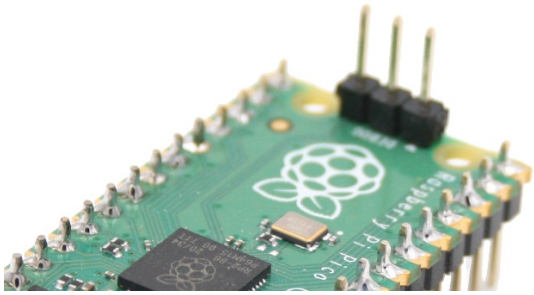
© Tous droits réservés – 2023 – MC Hobby Srl



02RI32 – Rang disponibles pour le prototypage

Concernant le port de débogage utilisé pour les développement C et C++: la fondation recommande d'y placer des connecteurs diriger vers le haut.

A moins de vouloir déboguer le fonctionnement interne de MicroPython (écrit en C), il n'est pas nécessaire de souder ce connecteur pour suivre cet ouvrage.



02RI33 – Port de débogage (optionnel)

3.5.Ressources graphiques

En tant qu'*Approved Reseller* de la fondation Raspberry-Pi, la société MCHobby (dont fait partie votre hôte) a eu accès à la nouvelle plateforme avant sa sortie. Cette découverte précoce a été l'occasion de combler quelques lacunes documentaires en produisant, très tôt, des ressources graphiques autour du Pico.

Ainsi, le dépôt GitHub pyboard-driver de MCHobby contient une entrée pour le Pico.

https://github.com/mchobby/pyboard-driver/tree/master/Pico

Le lecteur y trouvera :

•La fiche de brochage officielle du Pico

•Une fiche de brochage simplifiée pour MicroPython (sera abordé plus loin dans ce chapitre).

•Un Pico annoté

•Un fiche d'identification des broches (dite *overlay*)

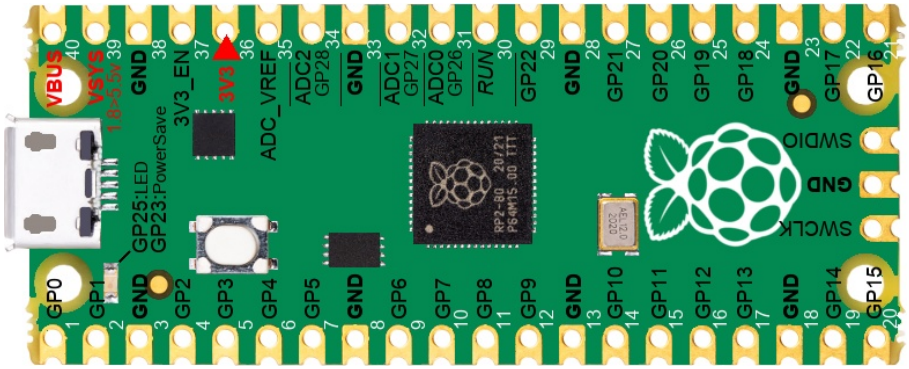
Le Pico annoté

Réaliser des schéma de raccordement filaire est particulièrement intéressant pour communiquer du savoir aux non initiés. Ce type de graphique peut encore être

© Tous droits réservés – 2023 – MC Hobby Srl

amélioré en ajoutant des annotations parfois indispensables si les composants comptent de nombreuses broches.

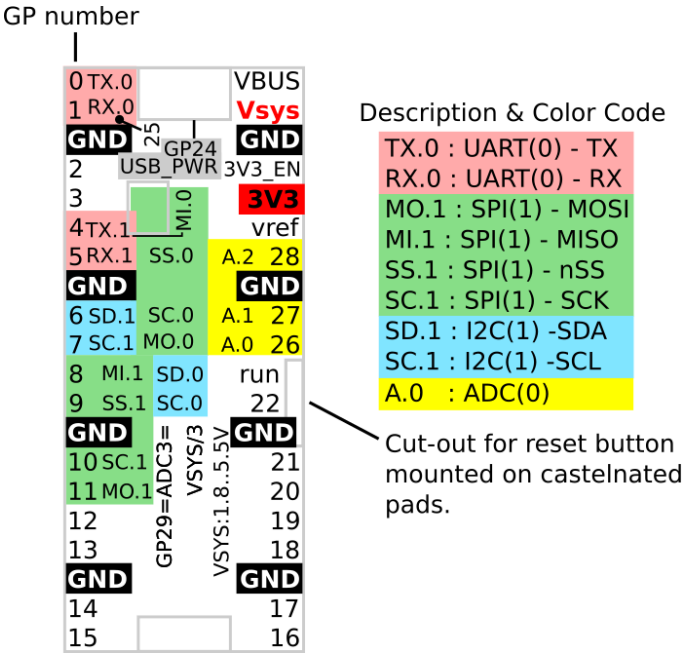
Utilisant souvent le logiciel de dessin Gimp (www.gimp.org), il a vite été évident qu'un Pico annoté serait de la plus grande aide pour repérer rapidement les broches pendant la réalisation d'un schéma filaire.



02RI40 – Pico annoté

Le Pico overlay

Le document « pico overlay » est un fichier PDF à imprimer et à découper. Une fois placer sur le Pico, l'overlay permet de repérer et identifier rapidement les broches et bus principaux disponibles sur la carte. Cette ressource est en anglais pour profiter au plus grand nombre.

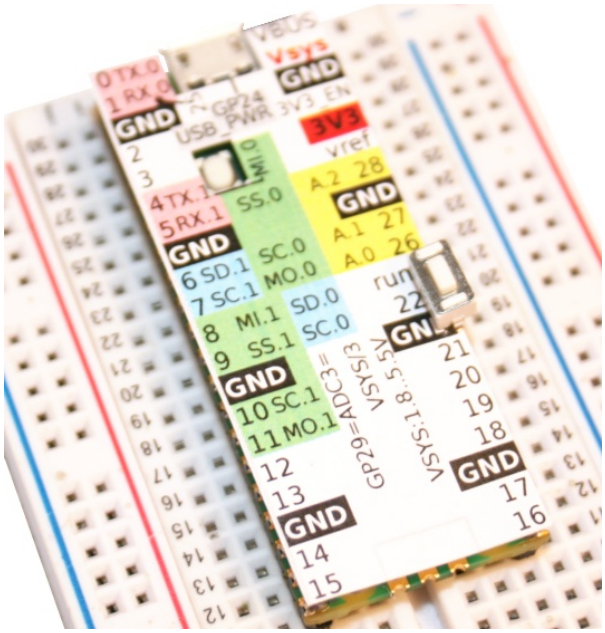


02RI41 – principe d'encodage de l'information

Le code couleur permet d'identifier rapidement le type de fonction alternative. La syntaxe compacte est rapide et facile à décoder révélant les éléments clés pour coder vos scripts Pythons. Ainsi `SD.0` *correspond au signal SDA du bus I2C* `I2C(0)` *.*

Une fois découpé et installé sur le Pico à l'aide d'un collant double face, l'identification des différentes broches devient triviale. L'overlay est très pratique pour éviter les erreurs et faciliter les vérifications.

© Tous droits réservés – 2023 – MC Hobby Srl



02RI42 – utilisation de l'overlay sur un Pico

4.Le Pico en détails

Cette section du chapitre va surtout s'attarder sur les différents éléments matériel de la plateforme Pico et présenter également quelques lignes de code à titre indicatif.

> *Bien que le Pico Wireless n'est pas concerné par cet ouvrage, il reste utile de préciser qu'il existe quelques différences mineures entre le Pico et le Pico Wireless.*

Cet ouvrage exploite exclusivement un Pico normal. Il s'avère que l'emploi d'un Pico Wireless (avec WiFi et Bluetooth) fait intervenir des mécanismes d'interruption supplémentaires difficilement compatibles avec la résolution problèmes temps réel (comme le suivit de labyrinthe).

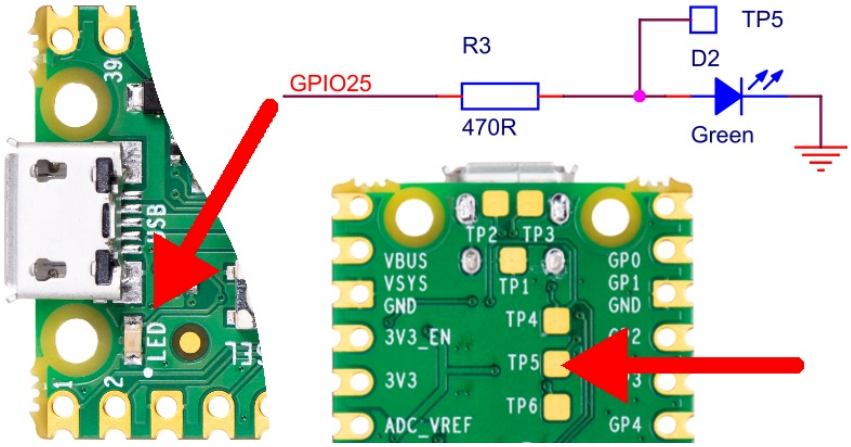
Si la lecture de cette section n'est pas indispensable il est recommandé d'en parcourir le contenu afin d'avoir un aperçu.

4.1.LED utilisateur – Pico uniquement

La LED utilisateur est, comme son nom l'indique, à disposition de l'utilisateur.

Cette LED est directement raccordée sur le GPIO 25 du Raspberry-Pi Pico.

© Tous droits réservés – 2023 – MC Hobby Srl



02RI50 – LED utilisateur sur le Raspberry-Pi Pico

Comme le schéma l'indique ce GPIO n'est pas accessible sur les connecteurs du Pico si bien que vous pouvez utiliser cette LED sans vous préoccuper des incidences de son utilisation un éventuel montage (ce qui n'est pas le cas d'un Arduino UNO).

La LED utilisateur peut être contrôlé à l'aide des lignes de code suivantes :

```
from machine import Pin
led = Pin(25, Pin.OUT)
led.value( 1 ) # allumer la LED
led.value( 0 ) # éteindre la LED
```

Si d'aventure un projet manquait d'un signal, celui du GPIO 25 (LED) pourrait être récupéré sur le point de test TP5 au verso de la carte.

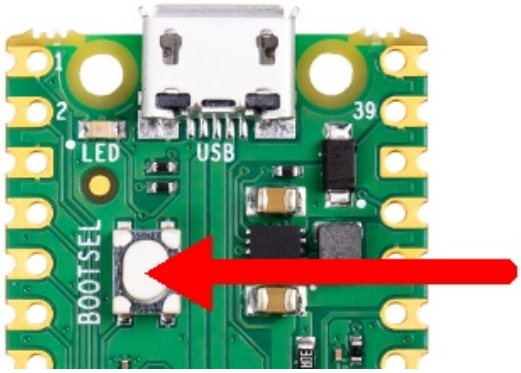
La tension y évolue entre 0 et 1.9V. Les 1.9V correspondent à la chute de tension aux bornes de la LED lorsque celle-ci devient passante. Il faudra dessouder la LED utilisateur si le niveau logique souhaité sur TP5 doit être de 3.3V.

4.2.Bouton Bootsel

Le bouton BootSel (souvent appelé « Boot ») permet d'activer le *bootloader UF2* sur la carte Pico. Le *bootloader* UF2 est un *firmware* spécial présent dans la ROM du RP2040. Ce *bootloader* permet de charger un programme utilisateur (*firmware*) dans la mémoire Flash du Pico.

Le bouton BootSel doit être maintenu enfoncé pendant la mise sous-tension du Pico. Une fois le Pico sous tension, le bouton BootSel peut être relâché.

Une fois le *bootloader* actif, le Pico présente sa mémoire Flash sous la forme d'un disque USB Flash.



02RI51 – Démarrer le mode Boot.

© Tous droits réservés – 2023 – MC Hobby Srl

Ce mode n'est pas utilisé avec MicroPython à moins de devoir réinstaller le *firmware* MicroPython sur une carte Pico (cfr : ch2 MicroPython et Raspberry-Pi Pico, Présentation du Pico – voir section « en cas de problème »)

Il est également possible d'activer le mode Boot en maintenant le bouton Bootsel enfoncé pendant le reset/réinitialisation du Pico (voir point « Reset » ci-dessous pour plus d'information sur cette fonctionnalité).

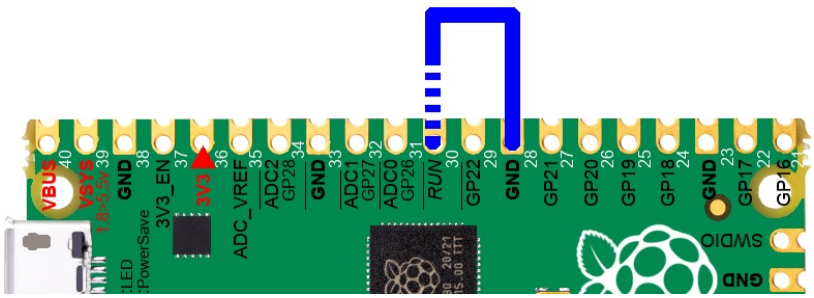
Bien que cela présente un intérêt limité, il est également possible d'activer le *bootlloader* depuis MicroPython.

```
import machine
machine.bootloader()
```

4.3.Reset et bouton Reset

De nombreuses plateformes disposent d'un bouton « Reset » permettant de redémarrer le *firmware* rapidement. C'est une fonctionnalité très pratique en cours de développement (quoique moins utilisée sur un produit fini).

Ce bouton fait cruellement défaut sur un Raspberry-Pi Pico (son seul vrai défaut pour les Makers). La carte dispose néanmoins d'une broche « run » qu'il suffit connecter à la masse pour réinitialiser le microcontrôleur

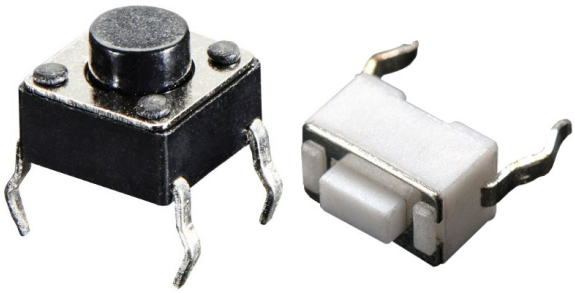


02RI52 – Utiliser la broche « run »

Utiliser un fil volant est fort peu pratique pour réinitialiser une carte, d'autant que les broches ne portent pas de libellé sur le dessus de la carte Pico. Utiliser un fil volant présente le risque de raccorder accidentellement une autre broche à la masse, ce qui peu détruire votre microcontrôleur selon les circonstances.

Par chance, l'espace séparant la broche « run » de la masse la plus proche est de 5mm (deux empattement standard, donc 2 * 2,54mm pour être exacte).

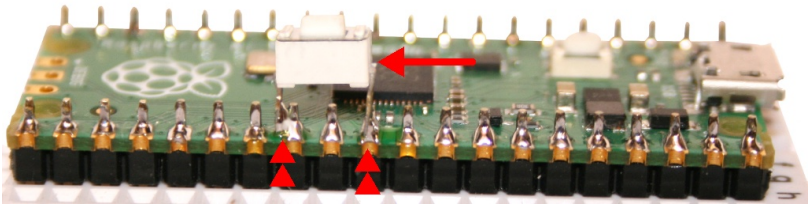
Il existe des boutons poussoirs de 6x6mm très répandus dans le monde maker. Ces boutons disposent d'une variante « demi-bouton » (deux fois moins larges, soit 6x3mm).



02RI53 – Bouton poussoirs 6x6mm et 6x3mm

Avec son empattement de 6mm, le bouton 6x3mm pourra être soudé sur le dessus des broches GND et « run » du Pico. Une façon élégante d'ajouter un bouton Reset sur un Pico.

© Tous droits réservés – 2023 – MC Hobby Srl



02RI54 – ajouter un bouton reset sur le Pico

Tout comme le bootloader, il est possible de provoquer un redémarrage par voie logicielle à l'aide des instructions suivantes :

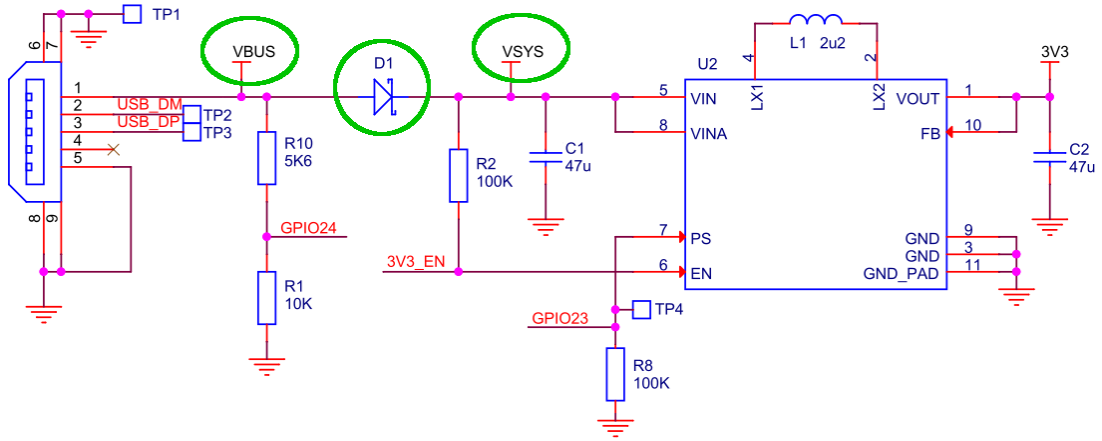
```
import machine
machine.reset()
```

4.4.Alimentation du Pico

Il existe de multiples façons d'alimenter le Raspberry-Pi Pico. La plus évidente est bien entendu le connecteur micro-USB qui sera la principale méthode utilisée.

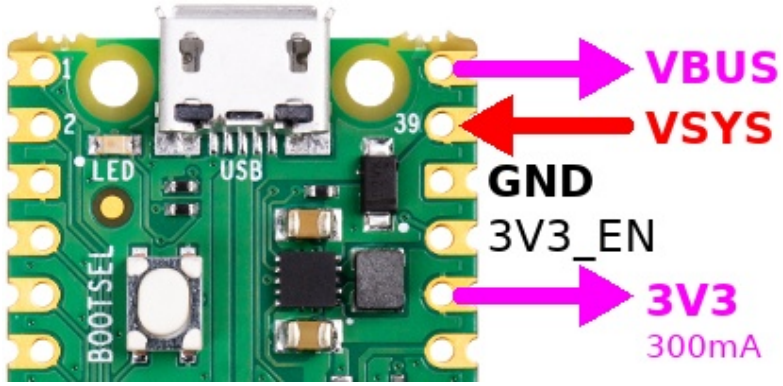
Ceci étant, la transformation d'un Pico en solution autonome à été anticipé par la fondation Raspberry-Pi. Cette section détaille les possibilités offertes par le Pico.

Le schéma ci-dessous reprend la chaîne d'alimentation du Raspberry-Pi Pico.



02RI55 – Chaîne d'alimentation du Pico

Le schéma ci-dessus reprend les signaux VBUS, VSYS et 3.3V accessibles sur les connecteurs du Pico.



02RI56 – broches d'alimentations du Pico

© Tous droits réservés – 2023 – MC Hobby Srl

4.4.1.Le régulateur de tension

Sur le schéma, le régulateur de tension se trouve à l'extrême droite (noté U2). Il s'agit d'un convertisseur de tension continu/continu basé sur un hacheur. Les hacheur sont des convertisseur à très haut rendement, ce qui limite les pertes et échauffements du circuit de régulation.

Le microcontrôleur RP2040 fonctionne sous 3.3V, tension produite par le régulateur de tension. Les 3.3V sont également disponibles pour les projets utilisateurs via le connecteur du Pico (broche `3V3`, à concurrence de 300mA maximum).

Le régulateur de tension du Pico est de type step-up/step-down. Cela signifie qu'il est capable de produire la tension de 3.3V en sortie à partir d'une tension d'entrée supérieur (fonctionnement *step-down* pour réduire la tension) ou inférieure à 3.3V (fonctionnement *step-up* pour élever la tension).

Ce régulateur de tension peut-être désactivé (mis en veille) en raccordant la broche `3V3_EN` à la masse.

4.4.2.Alimentation via USB

Lorsque le Pico est alimenté depuis le port micro-USB, la broche VBUS est à 5V (comme l'indique le schéma VBUS est connecté directement sur le connecteur micro-USB sans aucun intermédiaire).

Un pont diviseur réduit la tension VBUS qui est alors injecté sur le GPIO 24. Cette astuce permet au programme utilisateur de détecter la présence d'une tension d'alimentation sur le connecteur micro-USB.

Après VBUS le courant passe par la diode D1 pour atteindre le régulateur de tension (VSYS). **Le point VSYS doit absolument rester dans la gamme de tension de 1.8V à 5.5**.

A noter que dès qu'une tension est présente sur VSYS, le signal 3V3_EN est placé au niveau haut par l'intermédiaire de la résistance R2 (de 100K).

Placer la broche 3V3_EN à la masse permet de désactiver le régulateur de tension.

4.4.3.Alimentation via VSYS

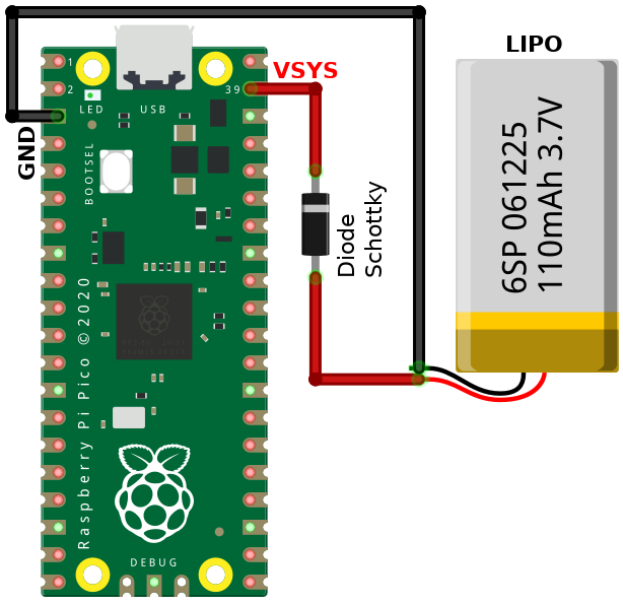
Le signal VSYS étant également disponible sur le connecteur du Pico, il est également possible d'alimenter Pico directement via VSYS.

Le point VSYS doit absolument rester dans la gamme de tension de 1.8V à 5.5.

En utilisant un bloc pile ou un accu LiPo, la caractéristique step-down/step-up du régulateur peut être mise à contribution pour produire les 3,3V du Pico.

Grâce à ce régulateur step-up/down Le Pico exploitera l'accu jusqu'à ses dernières ressources puisqu'il est capable de produire une tension de 3.3V même avec une tension VSYS inférieure à 3.3V.

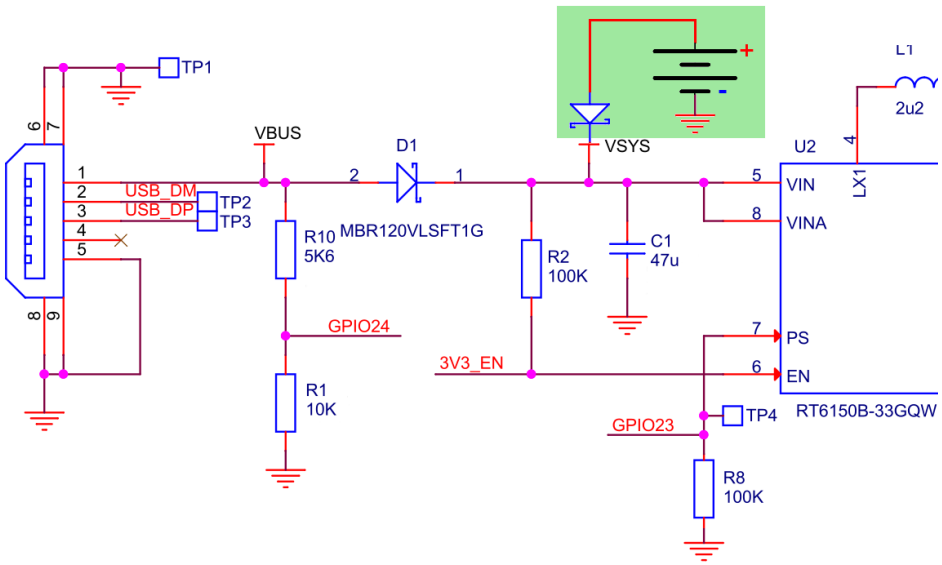
© Tous droits réservés – 2023 – MC Hobby Srl



02RI57 – Accu. LiPo avec sur LiPo

Le lecteur attentif notera la **présence de la diode Schottky** en série l'accu branché sur VSYS. Cette diode protége l'accu et l'empêche de recevoir tension et courant de recharge arbitraire si le Pico était également branché via USB.

Le schéma ci-dessous indique que le régulateur peut être alimenter indistinctement par l'accu ou le connecteur USB. Grâce aux diodes Schottky, le connecteur USB ne peut pas déverser de courant dans l'accu et l'accu ne peut pas déverser de courant dans le connecteur USB.



02RI58 – Chaîne d'alimentation avec accu Lipo

L'utilisation de diode Schottky n'est pas anodin. Ce type de diode commute très rapidement et présente une très faible chute de tension (0,15 à 0,45V). La commutation rapide permet d'éviter la coupure d'alimentation du Pico lorsque la prise USB est débranchée.

© Tous droits réservés – 2023 – MC Hobby Srl

4.4.4.PowerBoost : circuit de charge intelligent

Le montage ci-dessous présente la solution ultime pour alimenter pour un Raspberry-Pi Pico. Il permet :

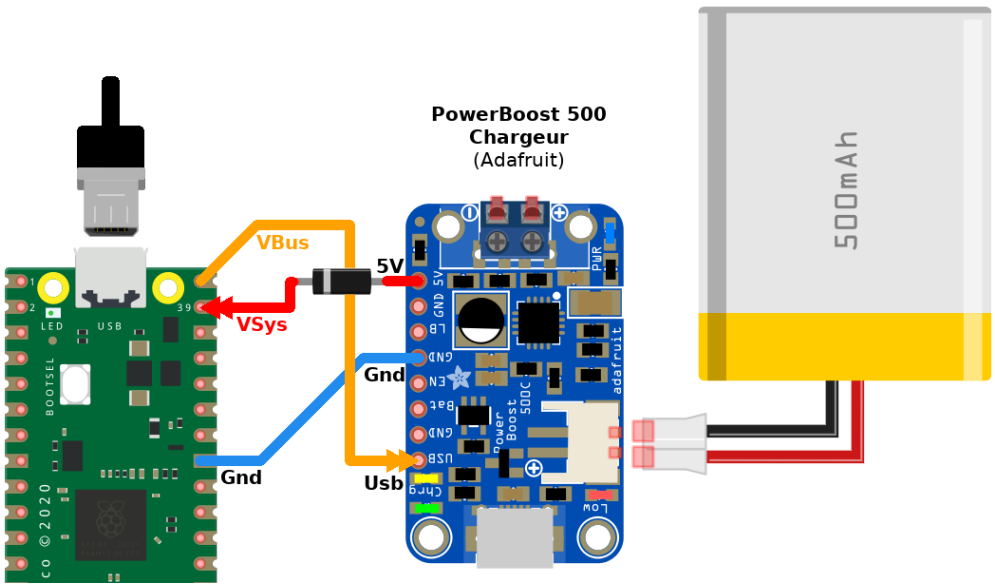
•D'alimenter et programmer le Pico depuis un ordinateur

•De recharger un accumulateur LiPo lorsque le Pico est branché sur un ordinateur (ou sur une alimentation USB).

•D'alimenter automatiquement le Pico depuis un accu Lipo (lorsque le Pico est débranché de la source d'alimentation USB).

•D'être alerté lorsque l'accumulateur est trop déchargé (avec LED)

Le montage ci-dessous utilise un circuit « PowerBoost charger » (Adafruit 1944, 2465), produit documenté en Français sur le site de MCHobby (https://shop.mchobby.be/product.php?id_product=534) .



02RI61 – Utiliser un PowerBoost 500 avec Pico

Lorsque le Pico est branché sur un ordinateur (ou alimentation USB), la broche VBUS du Pico est à 5V. Cela permet au PowerBoost de recharger l'accumulateur tout en continuant à produire une tension de 5V.

Lorsque le Pico est débranché de l'ordinateur alors les 5V générés par le PowerBoost traversent la diode Schottky pour alimenter le Pico par l'intermédiaire de la broche VSYS. Dans cette configuration la broche VUSB du Pico reste à 0V et l'accumulateur se décharge dans le PowerBoost pour produire les 5V injectés sur VSYS.

Quelques détails complémentaires sur le PowerBoost :

•Le PowerBoost propose également un bornier de 5V permettant d'alimenter d'autres éléments du projet.

•La **broche LBO** (*Low Battery* Output) passe au niveau bas lorsqu'il est temps de recharger l'accumulateur (tension d'accu sous 3.2V). A défaut, cette broche retourne la tension de l'accumulateur. Etant donné que cette broche peut présenter une tension de 4.2V, il est préférable de placer un pont diviseur de tension entre le PowerBoost et le Pico.

© Tous droits réservés – 2023 – MC Hobby Srl

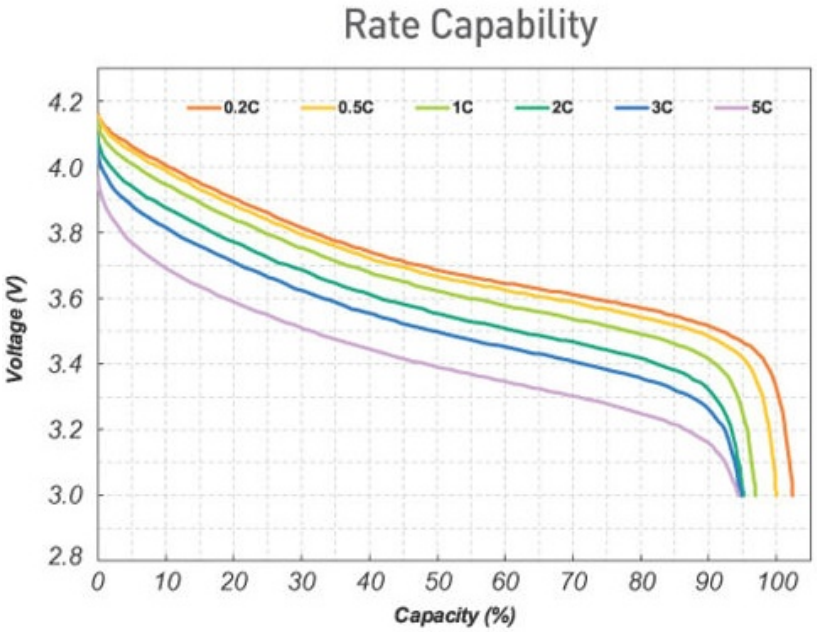
•La **broche EN** (*Enable*) lorsqu'elle est raccordée à la masse, elle permet de mettre le PowerBoost en mode veille. Un projet autonome peut ainsi être mis complètement hors-tension.

4.4.5.Décharge et surveillance d'un accu LiPo

Bien que les accumulateurs LiPo présentent une tension générale de 3.7V, il est important de noter qu'à pleine charge, un accumulateur Lipo présente une tension de 4.2V .

Le graphique ci-dessous présente diverses courbes en couleur présentant les courbes de tension en fonction de la charge appliquée.

*Les courbes sont présentées en multiple de C qui la capacité de l'accumulateur (ex : pour un accu de 500mAh, C=0,500). Ainsi la courbe 3C, en bleu, représente un courant de décharge de 3*500mA = 1500mA = 1.5A. Pour un accumulateur de 1300mAh, la courbe 3C représenterait un courant de décharge de 3*1300mA = 3900mA = 3.9A*



02RI59 – Courbe de décharge (source : li-polymer-battery.com)

La tension d'un accumulateur diminue assez rapidement vers 3.7V, tension qui reste assez constante pendant la majorité de la décharge de l'accumulateur.

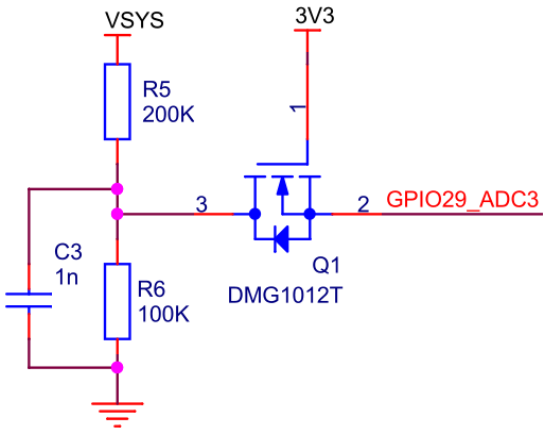
En fin de décharge, la tension chute rapidement vers 3V. C'est à cette tension que le circuit de protection d'un accumulateur LiPo s'active pour débrancher celui-ci du circuit (avant que l'accumulateur devient instable si la tension chutait sous 3,0V).

Il est donc raisonnable de permettre au Pico de surveiller la tension VSYS si l'accumulateur était branché directement sur VSYS (option B).

Pico standard – surveillance de l'accu Lipo

C'est justement ce que permet par l'entrée analogique (GPIO 29) branché sur VSYS par l'intermédiaire d'un pont diviseur de sorte que tension_ADC3 = VSYS/3 .

© Tous droits réservés – 2023 – MC Hobby Srl



02RI60 – Acquisition de la tension VSYS par le Pico

```
ADC3 = VSYS / 3
```

La tension VSYS est présentée par l'intermédiaire d'un transistor MosFet activé dès la mise sous tension du Pico (par la tension 3.3V produite par le régulateur). Cela évite d'injecter une tension sur l'entrée ADC alors que le microcontrôleur RP2040 serait lui-même hors tension. En effet, dans le cas où la broche 3V3_EN est à la masse, le régulateur de tension est désactivé et donc le microcontrôleur RP2040 aussi.

Powerboost – surveillance analogique de l'accu Lipo

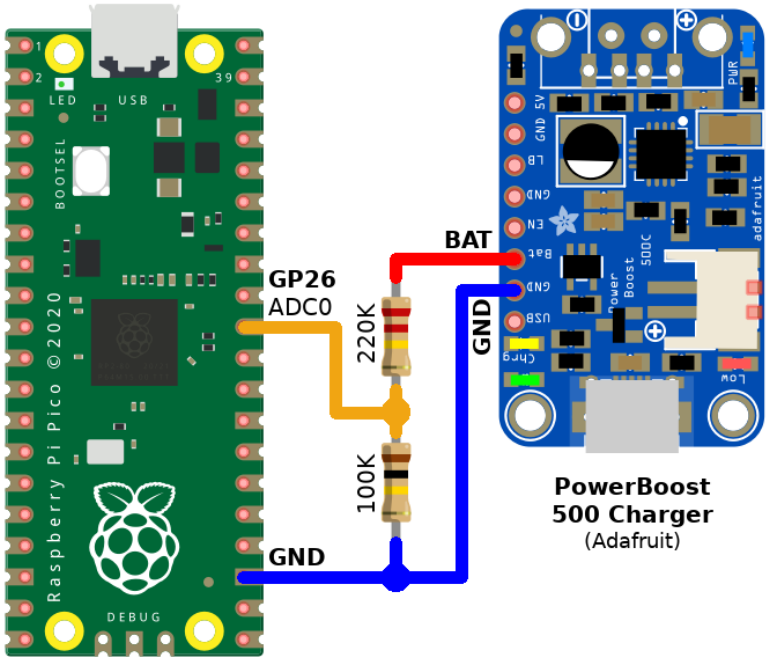
Si l'accumulateur est branché par l'intermédiaire du PowerBoost (option C) alors le PowerBoost produira une tension VSYS=5V tant que cela est possible puis il cessera subitement de fonctionner lorsque la tension de l'accumulateur Lipo sera trop faible.

Dans ce cas précis, il n'est donc pas utile de surveiller la tension VSYS à l'aide du GPIO29 (ADC3).

Cependant, la tension de l'accu est exposé sur la broche BAT du PowerBoost. Si on raccorde un pont diviseur de tension 220 KΩ + 100 KΩ vers une entrée analogique du Pico alors il devient possible de surveiller la chute de la tension de l'accumulateur (Tension ADC0 = 1/3 Tension Accu).

Le montage ci-dessous reprend les raccordements nécessaires pour mesurer la tension de l'accumulateur branché sur le PowerBoost.

© Tous droits réservés – 2023 – MC Hobby Srl



02RI60c – Surveillance analogique Accu PowerBoost

<u>Powerboost – surveillance numérique de l'accu Lipo</u>

S'il n'y a plus d'entrée analogique disponible sur le Pico il est également possible d'utiliser une entrée numérique pour surveiller la sortie LBO du Powerboost. Ainsi, le programme utilisateur pourra recevoir une notification lorsqu'il faut recharger l'accu.

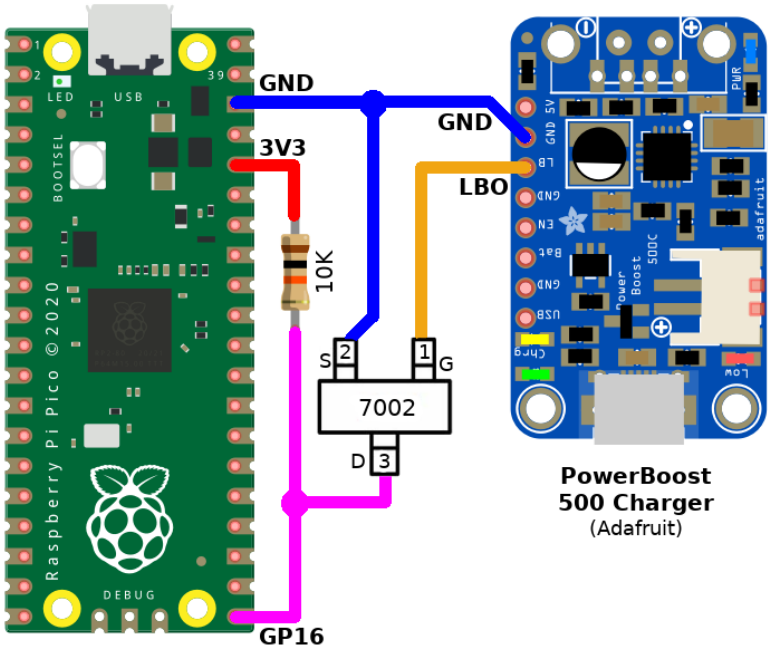
La broche LBO est maintenu au niveau haut = tension de l'accu (entre 4,2 et 3,0V) à l'aide d'une résistance pull-up. La broche LBO est placé à la masse lorsqu'il est temps de recharger l'accu.

Dans ce cas, n'est pas envisageable d'utiliser un pont diviseur de tension car il y a déjà une résistance pull-up sur LBO. Placer un pont diviseur supplémentaire perturberait l'effet de la résistance pull-up montée sur le PowerBoost.

Cependant, il est possible d'utiliser la présence/absence de tension pour commuter un transistor MosFet qui, lui, permettra de fournir un signal en logique 3,3V .

Le schéma ci-dessus utilise un transistor MosFet 2N7000 parce que celui-ci commute à partir d'une tension de ~2,0V jusqu'à 10V (donc aucun problème sur la tension d'activation est de 4,2V, tension présente sur LBO lorsque l'accu sera pleinement chargé).

© Tous droits réservés – 2023 – MC Hobby Srl



02RI60d - Surveillance numérique Accu PowerBoost

Si l'accu ne doit pas être rechargé alors la tension de la broche LBO > 3V (de 3,2V à 4,2V si l'accu est bien chargé). De fait, le transistor est passant et commute la broche GP16 directement sur la masse. GP16 est donc au niveau bas.

Si l'accu doit être rechargé alors la broche LBO est placée à la masse par le PowerBoost. Si LBO est au niveau bas alors elle draine les charges présente sur la porte/gate du transistor vers la masse. Le transistor devient bloquant. Par conséquent la broche GP16 reçoit une tension de 3,3V par l'intermédiaire de la résistance de 10 KΩ (dite résistance *Pull-Up*). GP16 est au niveau haut.

4.4.6.Sparkfun Lipo charger

Le PowerBoost d'Adafruit est l'un des meilleurs circuits DIY de charge et décharge d'accu. Cependant, il est aussi régulièrement en rupture de stock sur de tr ès longues périodes.

En alternative, il est possible d'utiliser le Lipo Charger de SparkFun (PRT-14411)

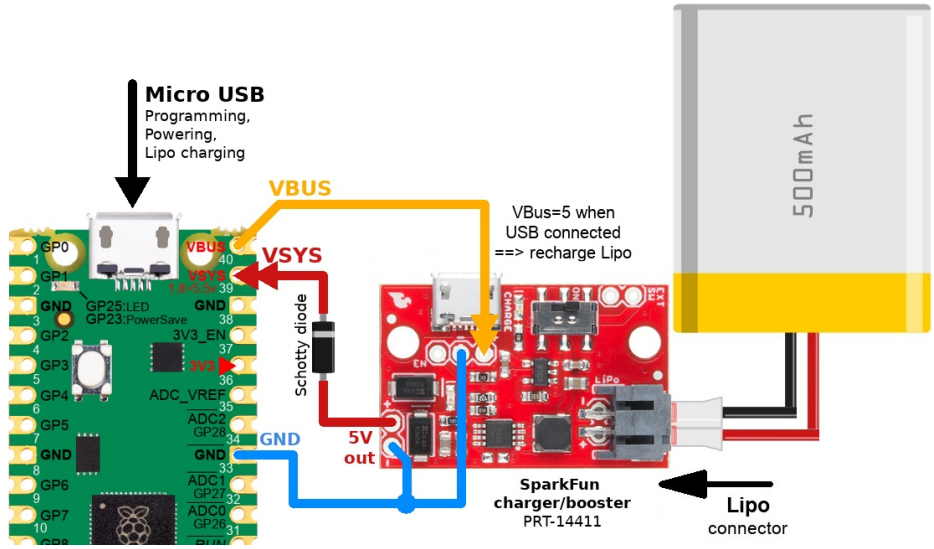
02RI62 – Lipo Charger de SparkFun

Tout comme le PowerBoost d'Adafruit, ce module est capable de produire une tension de 5V 1A à partir d'un accumulateur Lipo.

Le module est également capable de recharger l'accumulateur Lipo à partir du connecteur micro-USB.

© Tous droits réservés – 2023 – MC Hobby Srl

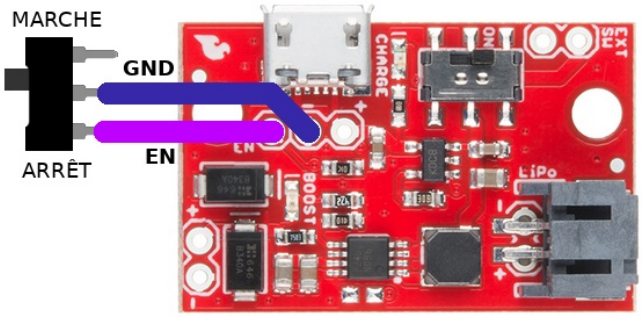
A la différence du PowerBoost, ce module est incapable d'effectuer simultanément une charge d'accu tout en produisant 5V 1A sur la sortie. Durant une recharge d'accu, le courant maximum est de 20mA (0,020 A) sur la sortie 5V.



02RI62b – Brancher un Lipo Charger PRT-14411 de SparkFun

Tout comme le PowerBoost, la broche VBUS du Pico est utilisée pour recharger l'accumulateur. Il suffit donc de brancher le Pico sur un ordinateur (ou un chargeur micro-USB) pour débuter la charge.

Etant donné que le module Lipo Charger ne peut pas produire de courant de sortie durant un cycle de recharge, il faut impérativement désactiver la sortie 5V du module. Cela est rendu possible par la présence de la broche EN qu'il suffit de raccorder à la masse pour désactiver la partie Boost de la carte.



02RI63 – Désactiver le module LiPo Charger

4.5.Brochage du Pico

La version officielle du brochage du Pico est assez déroutante et spartiate pour les nouveaux venu.

Si le *Bus Fabric* (cfr : Raspberry-Pi Pico, Microcontrôleur RP2040) permet d'avoir un même bus placé sur différentes broches du RP2040, le document récapitulatif reprenant un même bus sur différentes broches peut être source d'une grande confusion.

Puisque l'implémentation de MicroPython défini une position par défaut pour chaque bus, cet ouvrage reprend une version épurée et simplifiée du brochage du Pico.

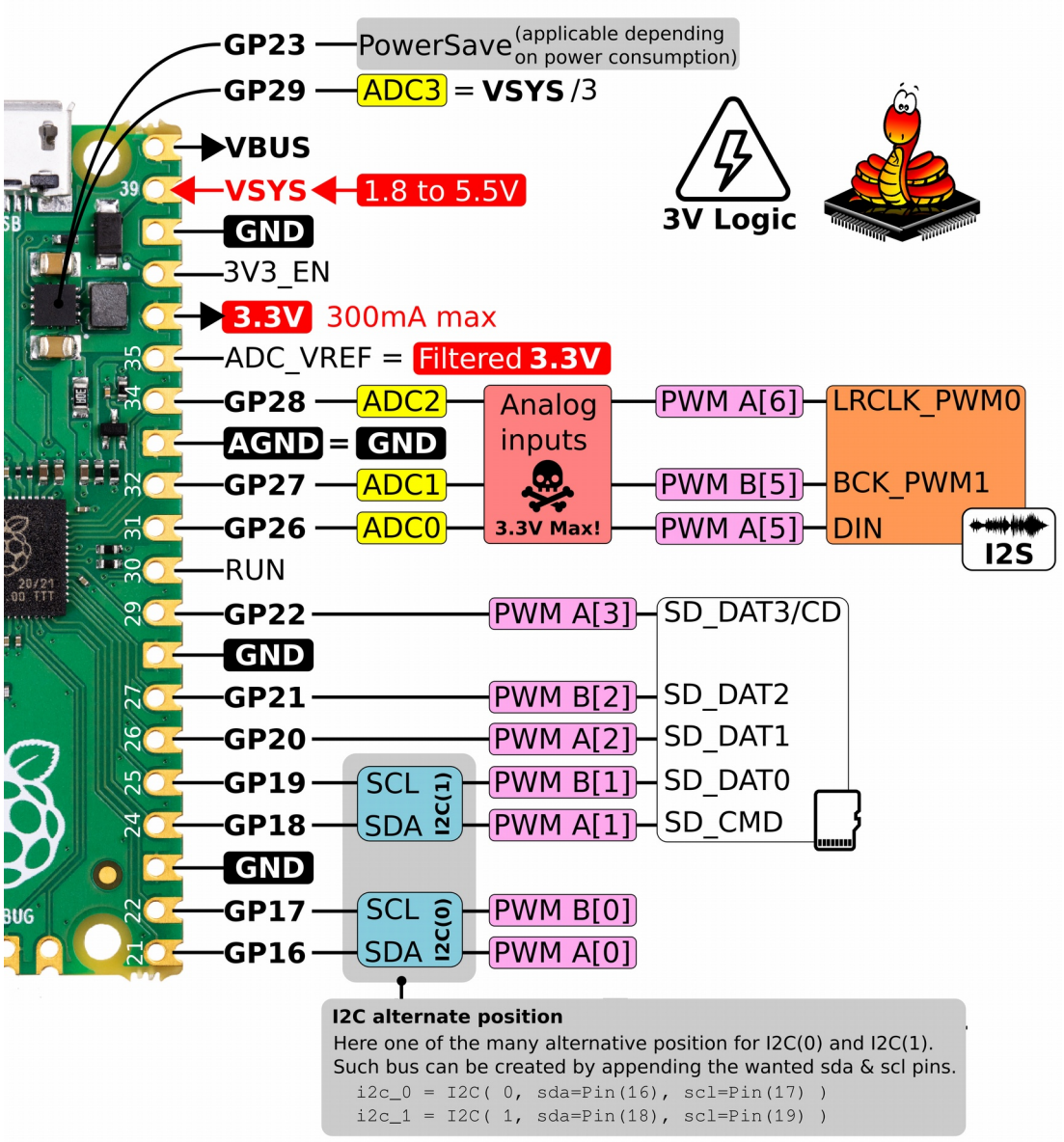
Cela ne signifie par pour autant que le MicroPython fait fit du *Bus Fabric* ! MicroPython pouvant aussi en tirer parti, la version officielle du brochage est également reprise dans l'ouvrage.

© Tous droits réservés – 2023 – MC Hobby Srl

4.5.1.Version simplifiée

© Tous droits réservés – 2023 – MC Hobby Srl

© Tous droits réservés – 2023 – MC Hobby Srl



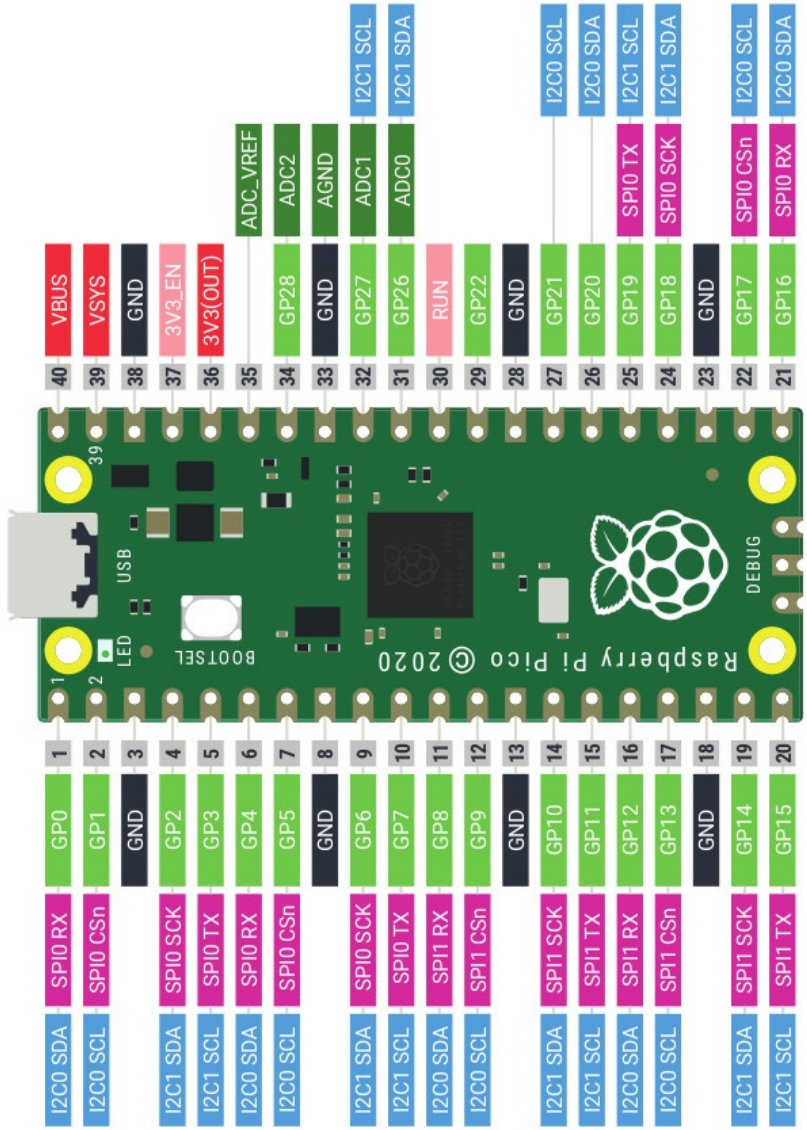
02RI71 – Brochage simplifié du Pico (partie droite)

Le bus I2C(0) et bus I2C(1) ont été relocalisé sur les broches GPIO 16 à 17 grâce au Bus Fabric. Dans pareil cas, les broches utilisées sont indiqués en paramètre en plus du numéro de bus.

4.5.2.Version officielle

Le brochage ci-dessous reprend la version officielle du brochage du Pico. Celui permettra d'identifier toutes les positions alternatives des bus du Pico.

© Tous droits réservés – 2023 – MC Hobby Srl



02RI75 – Brochage officiel du Raspberry-Pi Pico

5.Pico : tension logique et courant

Il y a deux informations importantes à garder en tête avant de réaliser ses premiers raccordements sur un Pico. Il s'agit des tensions et courants applicables.

Pour faire court :

1.La tension logique est 3.3V.

2.Pas de tolérance 5V

3.Courant de sortie 4 mA par GPIO.

4.Le courant maximum disponibles pour tous les GPIO est de 50 mA.

5.1.Niveau logique et tension

Les microcontrôleurs sont capables de fixer l'état d'une sortie numérique parmi les deux états disponibles :

© Tous droits réservés – 2023 – MC Hobby Srl

1.**Etat HAUT** : dit « *HIGH* » en anglais. Ce niveau HAUT correspond à la tension de 3,3 Volts fixée par le microcontrôleur. En Python, cela correspond aux valeurs `True`, `1` ou tout autre valeur entière positive.

2.**Etat BAS** : dit « LOW » en anglais. Ce niveau BAS correspond à la tension de 0 Volts fixée par le microcontrôleur (tension de la masse). En Python cela correspond aux valeurs `False` et `0`.

Niveau logique et tensions

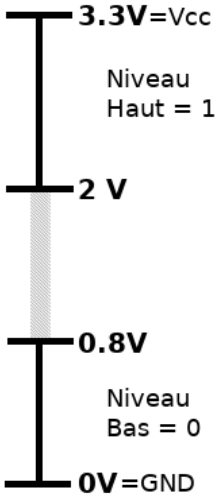
Lorsque la broche est configurée en entrée, la lecture de l'état haut/bas n'implique pas que la tension doit être strictement de 3,3V ou 0V.

La détermination de l'état logique est un peu plus permissif.

En effet, le **niveau HAUT** (`1` en Python) sera retourné par le microcontrôleur lorsque la tension présentée sur la broche est **supérieure à 2,0 Volts**.

Le **niveau BAS** (`0` en Python) correspond à une tension **inférieure à 0,8 Volts**.

Entre 0,8 et 2,0 Volts, l'état de l'entrée est indéterminé et totalement aléatoire.



02RI77 – Niveau logique

5.2.Niveau logique et Python

Le **niveau logique HAUT** correspond à toute expression Python pouvant être évaluée comme vraie. Par exemple, `True` ou `1` ou résultat d'un appel de fonction retournant `True` ou `1`.

Même si d'autres valeurs ou expressions pourraient être évaluées comme étant vraie (ex : une chaîne de caractère contenant un texte), il est vivement recommandé d'utiliser exclusivement `True` et `1` comme valeur pour désigner un état logique HAUT.

```
>>> from machine import Pin
>>> p = Pin(25, Pin.OUT) # LED pico sur GPIO25
>>> p.value(1)
>>> p.value(0)
>>> p.value(True)
>>> p.value(False)
```

Le **niveau logique BAS** correspond à toute expression Python étant évaluée comme fausse. Ce sera le cas des expressions `False` ou `0`.

© Tous droits réservés – 2023 – MC Hobby Srl

Lorsque qu'une broche est interrogée pour en déterminer son état logique, MicroPython retourne systématiquement les valeurs `0` et `1` pour les niveaux logiques `BAS` et `HAUT`.

```
>>> from machine import Pin
>>> p = Pin( 4, Pin.IN )
>>> p.value() # Lecture de l'état d'une broche
0
```

5.3.Tolérance 5V ? Non !

Le Raspberry-Pi Pico est un microcontrôleur fonctionnant exclusivement en logique 3.3V **sans aucune tolérance 5V**.

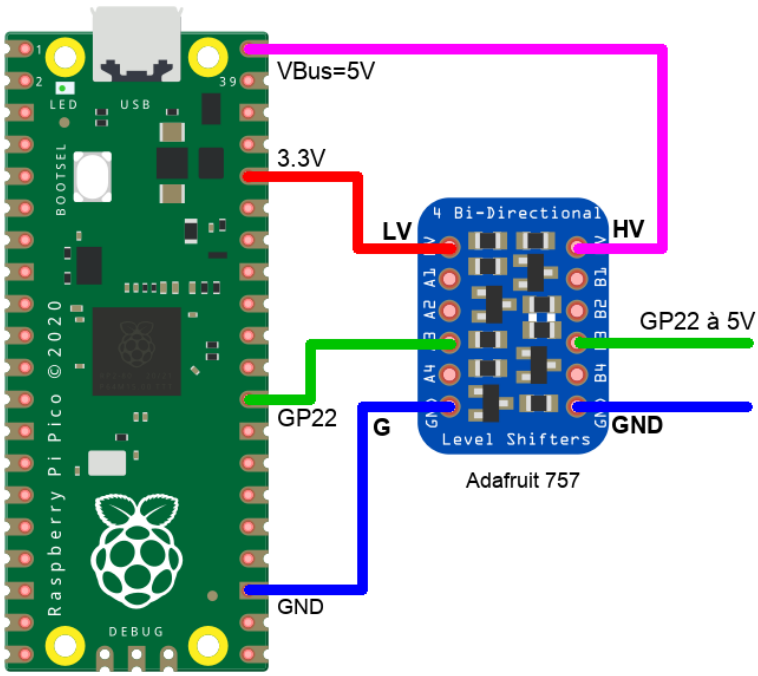
Il est donc interdit de présenter une tension supérieure à 3,3V sur une broche du Pico car cela aura pour effet de le détruire.

Les concepteurs de microcontrôleurs prévoient des mécanismes de protections mais il est préférable de ne pas les tester.

Il existe cependant des techniques pour passer d'une logique 3,3V à une logique 5V et vice-versa. Ces techniques sont nombreuses, certaines ne fonctionnent que dans un seul sens (par ex : de 3V vers 5V), d'autres sont bidirectionnelles (de 3V vers 5V mais aussi de 5V vers 3V), enfin certaines méthodes sont spécifiquement compatible avec certains bus (ex : I2C).

Cet ouvrage détaille un convertisseur de niveau logique (appelé « *Level Shifter* » en anglais) bidirectionnelle et compatible avec les bus I2C. D'autres convertisseurs et montage peuvent facilement être trouvées sur Internet.

Le schéma indique comment transformer le signal GP22 de 3,3V vers 5V à l'aide d'un convertisseur de niveau logique pré-assemblé. Ce type de breakout dispose d'une partie basse tension (LV = *Low Voltage,* 1,8V minimum) et d'une partie haute tension (HV = *High Voltage*, 10V max).



02RI80 - Passage en logique 5V

Le modèle ci-dessus est **bidirectionnel**, il peut donc transférer les signaux dans les deux sens.

© Tous droits réservés – 2023 – MC Hobby Srl

•Si la broche **GP22 est utilisée en sortie**, le signal de GP22 (0 ou 3,3V) sera converti en signal 0 ou 5V.

•Si le broche **GP22 est utilisée en entrée**, le signal (0 ou 5V) présenté sur la partie haute tension (HV) du convertisseur sera converti en signal 0 à 3,3V pour la broche GP22 (la partie LV du convertisseur).

L'intérêt de ce convertisseur est qu'il peut également adapter le niveau logique des signaux d'un bus I2C qui est, justement, bidirectionnel. Le bus I2C est très largement exploité dans le monde des Makers.

5.4.Courant maximum, source et sink

Conformément à la fiche technique du RP2040, le courant d'une broche peut être configuré, **la valeur est fixée à 4 mA par défaut** (fiche technique RP2040, 5.2.2 *Pin definitions*). Il n'y a pas de distinction entre courant de sortie (*source*) et courant injecté/absorbé (*sink*).

A noter que le courant total géré le microcontroleur RP2040 (donc pour tous les GPIO confondus) ne peut dépasser un **total maximum de 50 mA**.

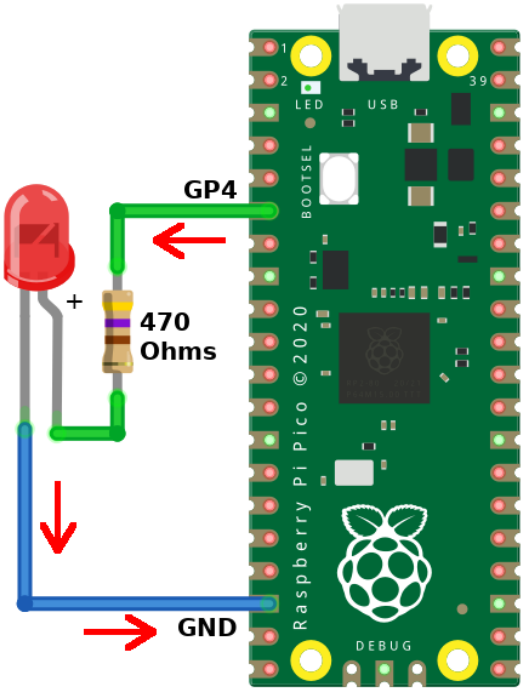
↘ *un GPIO peut être configurer supporter un courant de 2, 4, 8 ou 12 mA (fiche technique RP2040, 2.19 GPIO). Cela fait référence à la possibilité de fixer « la force du courant » (dit « current strength », Fiche technique RP2040 registre PADS_BANK0 0x7C*).

Source : broche en source de courant

L'esprit conçoit facilement qu'une broche configurée en sortie puisse fournir du courant au périphérique qu'elle commande. Dans ce cas, la broche fonctionne comme une « source de courant » (dit *source* en anglais).

Dans l'exemple suivant, lorsque la broche GP4 (configurée en sortie) est au niveau haut, la tension de la broche est de 3.3V. Il existe donc une différence de tension aux bornes de la LED et un courant peut y circuler de la broche GP4 vers les masse (GND). Ce courant, qui traverse aussi la LED, fait briller la diode électroluminescente.

© Tous droits réservés – 2023 – MC Hobby Srl



02RI90 – Montage type source

Ce cas de figure met en évidence le mode **source** qui produit le courant de fonctionnement nécessaire à la LED.

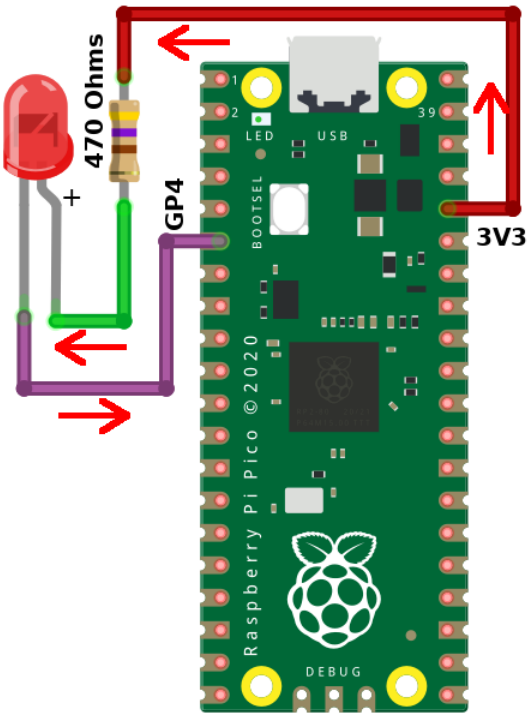
Le script de commande est :

```
from machine import Pin
p4 = Pin( 4, Pin.OUT )
p4.value( 1 ) # allumer
p4.value( 0 ) # éteindre
```

Sink : broche en injection/absorption de courant

Comme le montre le schéma ci-dessous, il est également possible de brancher une LED de telle sorte que le courant entre dans la broche GP4 (toujours configurée en sortie). Dans ce cas de figure, il faut que la broche GP4 soit au niveau bas pour qu'il existe une différence de tensions (dite « différence potentiel ») aux bornes de la LED + résistance. S'il existe une différence de potentiel aux bornes de la LED + résistance alors un courant y circulera !

© Tous droits réservés – 2023 – MC Hobby Srl



02RI91 – Montage type sink

La broche positive de la LED reste constamment au potentiel de 3.3V. Si la broche GP4 est au niveau haut (donc 3.3V) alors il n'existe pas de différence de potentiel aux bornes de la LED. Aucun courant ne peut circuler dans la LED et elle reste éteinte.

Par contre, si la broche GP4 est placée au niveau bas alors il existe une différence de potentiel aux bornes de la LED et un courant peut y circuler illuminant celle-ci.

Le script de commande est :

```
from machine import Pin
p4 = Pin( 4, Pin.OUT )
p4.value( 1 ) # éteindre
p4.value( 0 ) # allumer
p4.value( 1 ) # éteindre
```

Le mode sink, de l'anglais « *to sink* » = absorber, indique bien la fonction que joue ici le GPIO en absorbant le courant et le drainant vers la masse. Le mode sink est aussi connu sous l'appellation « injection de courant » dans la littérature Française puis du courant est injecté dans la broche.

5.5.Impédance d'un GPIO

L'impédance représente la « résistance » d'une broche au passage du courant. L'impédance est un terme employé en mode de fonctionnement dynamique où la tension est en variation constante. Cette impédance peut raisonnablement être réduite à une résistance pure lorsque la broche fonctionne avec du courant continu (donc variant peu et sans aucun aspect dynamique).

Suivant la configuration d'une broche en entrée ou en sortie, l'impédance de celle-ci sera radicalement différente. Une faible impédance ne manquera pas d'avoir des conséquences expéditives en cas d'erreur de montage (ou modification à la volée en court de fonctionnement).

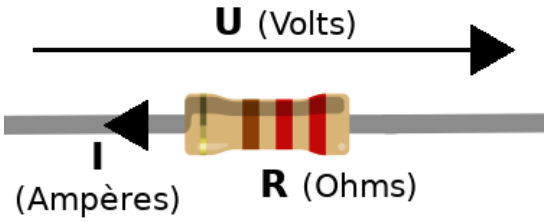
© Tous droits réservés – 2023 – MC Hobby Srl

5.5.1.Rappel sur la loi d'Ohms

Pour pouvoir remplir ses différentes fonctions, le microcontrôleur doit pouvoir contrôler la « résistance » (l'impédance) de sortie de ses broches.

Ainsi, une petite révision sur la loi d'Ohms s'impose.

Il existe une relation entre la chute de tension aux bornes d'une résistance et le courant qui la traverse.



02RI92 – Résistance et loi d'Ohms

Une résistance s'oppose au passage du courant ce qui se traduit par un dégagement de chaleur et une chute de tension à ses bornes.

`U = R x I`

Avec `U` la chute de tension (en volts, V), `R` la résistance (en Ohms, Ω) et `I` le courant traversant la résistance (en ampères, A).

Si un courant I de 2A traverse une résistance R de 10 Ohms alors la chute de tension aux bornes de la résistance est de 10 x 2 = 20 Volts.

Pour peu que deux des trois éléments U, R, I sont connus, il est toujours possible de déterminer la valeur du troisième.

5.5.2.Broche en sortie = faible impédance

Une broche en sortie est destinée a piloter un périphérique. Dans ce cas de figure, c'est le microcontrôleur qui fixe la tension de sa broche ! L'exemple typique est le contrôle d'une LED qui nécessite un courant de quelques mA (milliampères) à 10mA. Si ce courant semble très faible, c'est déjà un courant important à la dimension d'un microcontrôleur.

Il faut donc pouvoir compter sur le fait que la sortie puisse délivrer le courant nécessaire tout en gardant la tension de sortie au maximum (3,3V dans le cas du Pico).

Par conséquent, **la broche en sortie présente une faible impédance** équivalent d'une faible résistance interne (de l'ordre de 0,10 Ohms). Grâce à cette faible impédance, la tension de la broche ne chute pas lorsque le courant qui la traverse augmente.

Il y a aussi un revers : comme **rien n'arrête le courant**, il est possible d'en consommer plus que de raison. C'est le cas si la sortie est raccordée par erreur sur la masse. Dès que la broche passe au niveau haut, le courant tend vers le maximum possible (donc vers l'infini), ce qui provoque la fonte des jonctions internes et envoi illico-presto le microcontrôleur au paradis des composants.

© Tous droits réservés – 2023 – MC Hobby Srl

5.5.3.Broche en entrée = forte impédance

Le but d'une broche utilisée en entrée est de permettre l'application d'une tension logique (0V ou 3,3V) sur la broche et permettre au microcontrôleur d'en lire l'état. C'est un dispositif externe qui fixe la tension d'une broche configurée en entrée.

Dans ce cas de figure, Il faut limiter au maximum le courant qui peut traverser l'entrée (sinon elle fondra). L'entrée présente donc un **forte impédance** (une « grande résistance »), ce qui **limite le courant qui peut la traverser**. Des informations glanées dans la fiche technique du RP2040, celle-ci est de l'ordre de 1 MOhms (1 000 000 Ohms).

Lorsqu'une broche, configurée entrée, est branchée à la masse alors le microcontrôleur peut y détecter un niveau bas. La tension maximale sur une broche en entrée est de 3,3V. Si la broche en entrée est connectée sur la tension de 3,3V alors la courant qui traverse broche est I = U/R (loi d'Ohms) = 3,3V / 1 000 000 Ohms = 0,0000033 A, soit 3,3µA ! Mission accomplie.

6.Les fonctions alternatives sur le Pico

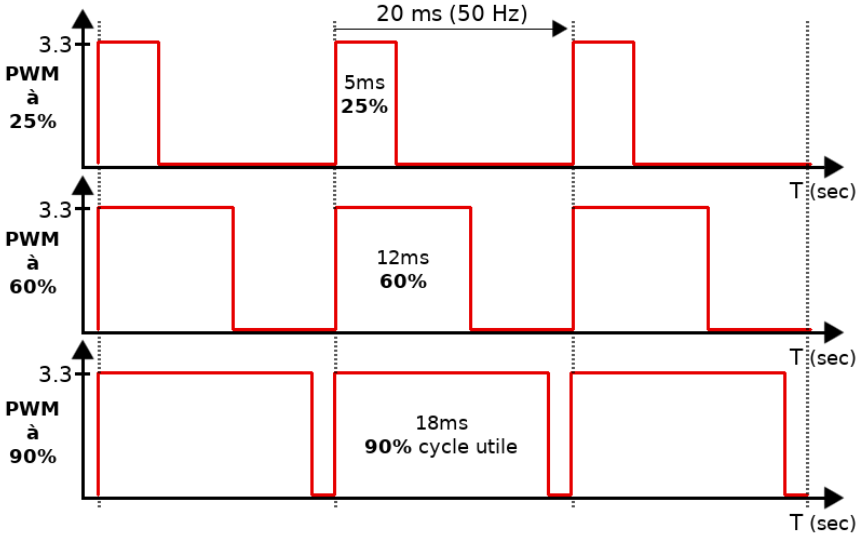
Les fonctions de bases des broches d'un microcontrôleur ne s'arrêtent pas aux opérations d'entrées (lire l'état d'une broche) ou de sorties (fixer l'état de la broche).

Les broches du Raspberry-Pi Pico, comme d'autres microcontrôleurs, propose d'autres fonctionnalités.

6.1.Sortie PWM

Une sortie PWM (*Pulse Width Modulation*) est une modulation de signal produisant un signal carré à une fréquence fixe et arbitraire (fixée par le programme utilisateur) dont il est possible de moduler le cycle utile de 0 à 100 % (le rapport entre le temps au niveau haut et le niveau bas sur une période de signal).

Le graphique ci-dessous représente un signal PWM produit à la fréquence de 50 Hertz. Il se reproduit donc toutes les 20 ms (Période = 1 / 50 Hz = 0,020 sec = 20ms)



02RI93 – Production d'un signal PWM

© Tous droits réservés – 2023 – MC Hobby Srl

Suivant la durée du cycle utile (ex : 25 %, 60 % et 90%), il est clairement évident que le signal « rempli » plus le graphique, ce qui augmente la valeur moyenne du signal.

En littérature Française, un signal PWM se nomme aussi MLI pour « Modulation de Longueur d'Impulsion ».

Utilité d'un signal PWM

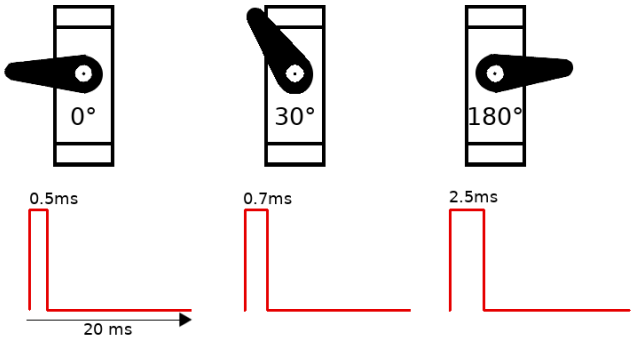
Dans le monde de l'industrie, un tel signal est régulièrement exploité dans les circuits d'asservissement (ex : régulation de la vitesse moteur).

Dans le monde des Makers, ce signal sera utilisé pour :

•Moduler la puissance lumineuse d'une LED (puisque cela ne peut pas se faire en modulant la tension).

•Contrôler un servo-moteurs dont la position angulaire de l'axe dépend du cycle utile. Ce signal PWM est un peu particulier car il n'exploite que quelques pourcents du cycle utile. C'est le type de moteur que l'on retrouve dans les jambes des robots bipèdes.

02RI93c – Servo moteur



02RI93b – Signal PWM spécifique au servo moteur

•Produire des notes de musiques en modifiant le fréquence du signal PWM (et gardant toujous un cycle utile de 30 % ou 50%). Le son « La » se jouant à 440 Hertz.

•Contrôler la vitesse de rotation d'un moteur.

Aide mémoire MicroPython

Les quelques lignes suivantes montrent comment exploiter un signal PWM sur le GPIO 4 du Pico. Si une LED + résistance sont branchés sur cette broche (voir point précédent) alors la modification du cycle utile fera varier la luminosité de la LED.

```
>>> from machine import PWM, Pin
>>> pwn = PWM( Pin(4) )
>>> # Fixer la fréquence
```


© Tous droits réservés – 2023 – MC Hobby Srl

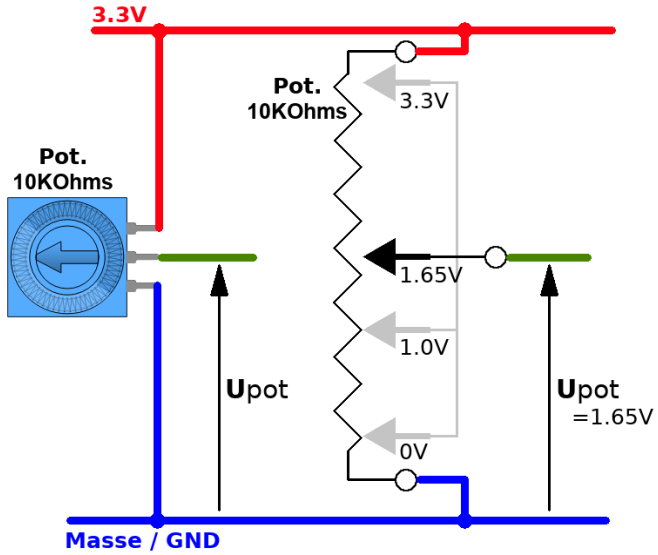
```
>>> pwn.freq( 50 )
>>> # Fixer le cycle utile entre 0 et 65535
>>> # Cycle utile à 50 %
>>> pwn.duty_u16( 32767 )
```

6.2.Entrée analogique ADC

Une entrée analogique, aussi appelée ADC pour « *Analogic to Digital Converter* », permet de convertir une tension analogique arbitraire (comprise entre 0 et 3,3V) en valeur numérique de 0 à 65535 (représentant un entier 16bits non signé).

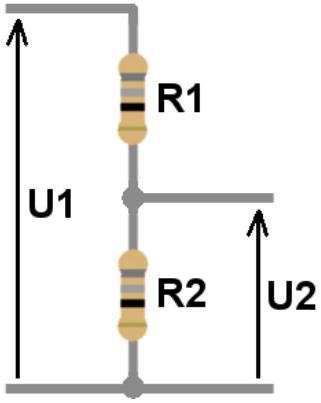
L'entrée analogique est une interface de choix pour fixer une consigne (potentiomètre), une tension de référence, acquérir la tension produite par un capteur analogique comme une photo-résistance ou capteur de température analogique (tmp36).

Le potentiomètre est un composant idéal pour définir une consigne. C'est aussi un composant permettant de produire facilement une tension analogique qui être appliqué sur une entrée analogique du microcontrôleur.



02RI94a – Utilisation d'un potentiomètre

Le potentiomètre fonctionne comme un pont diviseur de tension, un des fondamentaux de l'électronique où la tension de sortie du pont diviseur est un rapport des deux résistances qui le compose. Un potentiomètre est simplement un curseur voyageant le long d'une résistance faisant ainsi varier R1 et R2 dans des proportions inverses.



02RI94b – Pont diviseur de tension

© Tous droits réservés – 2023 – MC Hobby Srl

Selon la formule de calcul du point diviseur, la tension de sortie U2 peut être calculée à partir de la tension d'entrée et des deux résistances R1 et R2.

```
U2 = U1 * R2 / (R1 + R2)
```

Si le potentiomètre est branché entre la masse et 3,3V alors `U1=3,3V` .

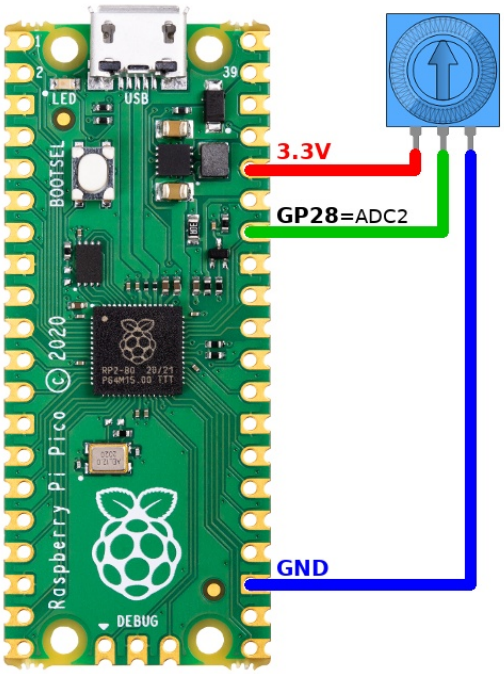
Si le curseur du potentiomètre de 10KOhms est placé dans le premier tiers de son parcours alors les 10 KOhms seront divisés en R1 = 3,3 KOhms (1/3 de 10 KOhms) et R2 = 6,6 KOhms (les 2 dernier tiers de la résistance).

Si `R1 = 3,3 KOhms` et `R2 = 6,6 KOhms`, le pont diviseur présente un rapport de 3300 / (3300 + 6600) = 0,33. La tension d'entrée `U1` est donc divisée par 3.

Dans le cas du potentiomètre branché entre 0 et 3,3V la tension de sortie sera :

```
U2 = 3,3 * 3300 / (3300 + 6600) = 1,1 Volts.
```

Aide mémoire MicroPython



02RI94c – Potentiomètre de 10 KOhms branché sur GP28

Les quelques lignes de code ci-dessous permettent de lire l'entrée analogique présente sur GP28 et convertir cette lecture en tension (volts).

```
>>> from machine import ADC, Pin
>>> adc = ADC( Pin(28) )
>>> # lecture ADC (entre 0 et 65535)
>>> val = adc.read_u16()
>>> val
19140
>>> # tension en volts
>>> v = val * 3.3 / 65535
>>> v
0.963
```

Tension supérieure à 3,3V

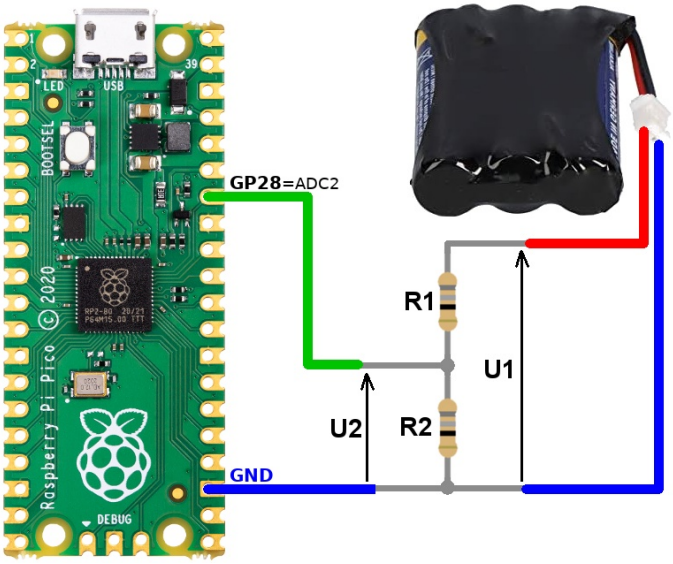
La théorie du pont diviseur de tension qui explique le fonctionnement du potentiomètre peut aussi être utilisée pour mesurer des tensions supérieure à 3,3V.

© Tous droits réservés – 2023 – MC Hobby Srl

En effet, en utilisant un pont diviseur avec résistances fixes `R1 = 3,3 KOhms` et `R2 = 6,6 KOhms` epermet de mesurer une tension supérieure à l'entrée du pont diviseur U1 tout en restant sous la tension maximale de 3,3V en sortie du pont diviseur (U2).

Nous savons déjà que le rapport de ce pont est de `3300/(3300+6600)` = 0,33 , soit divisé par 3. `U2=U1*0,3333` .

Prenons l'exemple d'une batterie 5V dont la tension est actuellement de 4,9V. En utilisant le pont diviseur approprié, il est possible de mesurer cette tension sur le Pico.



02RI94d – Mesurer une tension supérieure à 3,3V

Soit U1 = 4,9V, U2 = 4,9 * 3300 + (3300 + 6600) = 1,61 V

La lecture sur l'ADC2 avec `read_u16()` retournera la valeur numérique 31973.

Pour retrouver la tension de la batterie, il faut appliquer la formule suivante :

```
Vbat = ( 3,3/65535 ) * read_u16 * ( 1/ rapport_diviseur )
Vbat = ( 3,3/65535 ) * 31973 * ( 1/0,33 )
Vbat = 4,878 Volts
```

Le fait de diviser une tension d'entrée augmente également l'erreur sur la mesure. En effet, lors de la lecture sur l'ADC la tension (U2) de 1,61V devra être multipliée par 3,33 pour obtenir la tension à l'entrée du pont diviseur (U1). Cette multiplication démultiplie également les erreurs et imprécisions.

6.3.Sortie analogique DAC

Les sorties analogiques sont à l'opposé des entrées analogiques. Un DAC (*Digital to Analog Converter*) permet de produire une tension analogique entre 0 et 3,3V à partir du microcontrôleur.

Le Pico ne dispose pas de DAC intégré dans son microcontrôleur RP2040 mais cela n'empêche pas d'utiliser un composant externe comme un breakout MCP4725 qui dispose déjà d'un pilote MicroPython.

Une sortie analogique peut se montrer particulièrement utile dans certaines situation comme le rendu audio, le génération de signal ou l'asservissement moteur. Un DAC est tout à fait capable de produire un signal sinusoïdal, triangle, rampe. Le DAC

© Tous droits réservés – 2023 – MC Hobby Srl

peut également générer un signal totalement arbitraire comme un contenu audio (ex : fichier audio .WAV) avec un qualité tout à fait acceptable. La principale limitation d'un DAC externe réside dans les capacités de transfert de données du microcontrôleur, le débit du bus de communication et l'optimisation du programme sur le microcontrôleur.

Il existe de nombreux modules de contrôle utilisant une tension analogique comme consigne de référence pour l'asservissement d'un système. C'est le cas d'un nombre non négligeable de contrôleurs moteurs qui utilisent une consigne analogique pour asservir la vitesse ou la puissance de celui-ci.

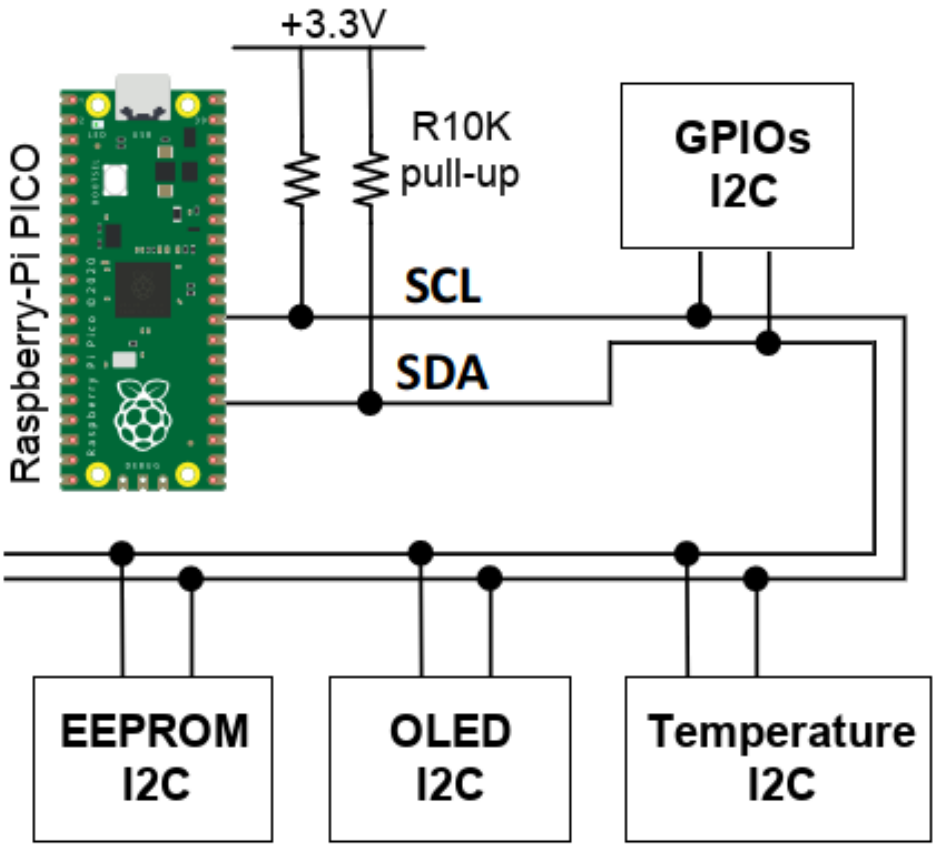
Le lien suivant propose une implémentation du DAC MCP4725 pour MicroPython :

https://github.com/mchobby/esp8266-upy/tree/master/mcp4725

6.4.Bus I2C

Le bus I2C (I²C) est un bus de données deux fils permettant de connecter plusieurs capteurs ou actionneurs sur un microcontrôleur. Ce bus est très répandu et parfois ignoré des nouveaux venus dans le monde des microcontrôleurs.

Une fois découvert, cette interface de communication devient vite un incontournable de très nombreux projets. Une vaste gamme de composants supporte cette norme de communication, composant que l'on retrouve sur de nombreuses carte *breakout* pour Maker (cfr : gamme de produits M5Stack, Adafruit Industries, DFRobot, SparkFun, Pololu, etc).



02RI95 – topologie d'un bus I2C

© Tous droits réservés – 2023 – MC Hobby Srl

De nombreux capteurs sont distribués sous forme de carte breakout incluant déjà les résistances pull-up (résistance de rappel à 3,3v). Il n'est donc pas nécessaire de les ajouter sur votre montage).

L'intérêt du bus I2C est de pouvoir ajouter facilement une multitude de capteurs sur le bus en utilisant toujours les mêmes deux fils (SDA=données et SCL=horloge) pour établir la communication entre le microcontrôleur et les différents capteurs. Chaque capteur dispose une adresse 7 bits (0..127) unique sur le bus, la communication se fait avec un seul capteur à la fois, communication toujours initié par le maître du bus (le microcontrôleur). Les échangent sur le bus inclus l'adresse I2C du capteur cible sous la forme : `requête_microcontrôleur = réponse_capteur` (plus communément appelé topologie maître-esclave ou contrôleur-cible ou leader-suiveur).

Le bus I2C n'a donc rien de comparable avec un réseau Ethernet où tous les intervenants peuvent tenter de dialoguer quand bon leur semble sur le réseau.

Sur le bus I2C, seul le microcontrôleur (maître) est aux commandes du bus et les capteurs répondent exclusivement aux sollicitations du microcontrôleur.

Aide mémoire MicroPython

Les quelques lignes suivantes indiquent comment créer un bus I2C sur base de son numéro de bus (cfr : Raspberry-Pi Pico, Pico en détails – brochage du Pico).

```
>>> from machine import I2C
>>> i2c = I2C( 0 ) # SDA=gp8, SCL=gp9
>>> i2c.scan() # détection des adresses I2C utilisées
```

Il est possible de ralentir le bus I2C en ajoutant le paramètre `freq` (en baud), ce qui peut être utile situation de prototypage.

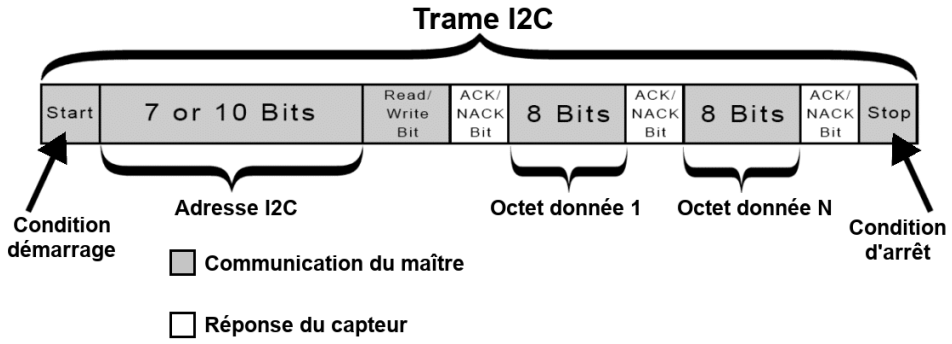
```
>>> from machine import I2C
>>> i2c = I2C( 0, freq=50000 )
```

Il est possible de trouver plus de documentation sur le bus I2C dans la documentation MicroPython.

https://docs.micropython.org/en/latest/library/machine.I2C.html

Trame de communication

Bien qu'il ne soit pas essentiel de dispose d'une connaissance très pointue sur ce sujet, une compréhension plus fine des échanges sur le bus I2C permet parfois de se sortir d'une mauvaise situation.



02RI95b – Trame type d'un échange I2C.

© Tous droits réservés – 2023 – MC Hobby Srl

Pour commencer, les conditions de démarrage et d'arrêt d'une communication sur le bus I2C se fait par le maître en imposant une combinaison particulière de l'état des signaux SDA (donnée) et SCL (horloge).

En cours de transfert, le maître attend des confirmations de la part du capteur. Le seul moment où le capteur peut prendre le contrôle du bus pour transmettre un bit ACK (*acknowledge*=OK) ou NACK (*not acknowledge*=PAS OK) en manipulant les signaux SDA et SCL.

La raison pour laquelle les adresses sont limitées à 7 bit c'est parce que le 8ieme bit est utilisé pour indiquer le type d'opération (lecture ou écriture) effectué par le maître sur le capteur.

Parmi les adresses 7 bits, certaines combinaisons sont réservées pour usage spécifique. Une d'entre-elles permet le passage à un mode d'adressage 10 bits (mode encore jamais rencontré par l'auteur).

Pour les lecteurs intéressés par les détails de la communication I2C, il existe un tutoriel didactite reprenant, entre autre, le concept des registres largement utilisé dans l'implémentation de capteurs I2C.

https://wiki.mchobby.be/index.php?title=Arduino_I2C_Intro

Avantages et inconvenients

Bien que le bus I2C soit très pratique et très apprécié, il souffre de quelques limitations :

1.**Le débit du bus est limité** à 400 Kibits par secondes. Cela représente 48 Kio/sec, suffisant pour des données télémétriques mais pas pour la vidéo et l'audio.

2.**La bande passante est partagée** par tous les capteurs chacun étant interrogé à son tour, le débit de données maximal par capteur est grossièrement équivalent au débit du bus divisé par le nombre de capteurs présents sur le bus.

3.**La longueur du bus est limitée à 1 mètre maximum**, 50 cm pour un bus prototype. A noter qu'il existe des composants actifs comme le LTC4311 permettant d'augmenter la longueur d'un bus I2C (voir https://wiki.mchobby.be/index.php?title=Accueil#Bus_I2C pour plus d'informations).

Ces limitations sont néanmoins contrebalancées par un avantage indéniable qui est son **état d'erreur**. Un état qui apparaît en cours de communications avec un capteur lorsque celui-ci rencontre un problème d'ordre technique ou de configuration. Cette caractéristique propre au bus I2C devient vite un avantage crucial lorsque la configuration des paramètres du capteur est incompatible ou lorsqu'il est nécessaire d'écrire du code pour supporter un nouveau capteur. En effet, un message « cela ne marche pas » est préférable au silence total.

En conclusion

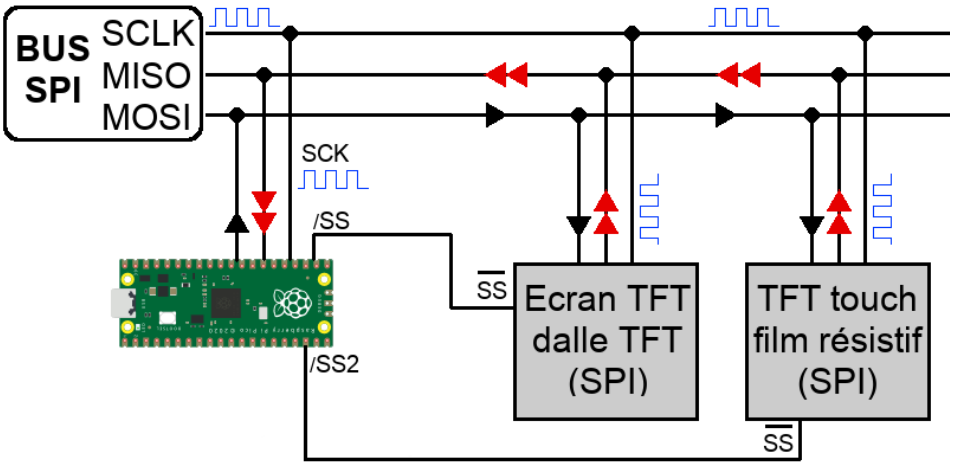
Le bus I2C est un bus flexible idéal pour la capture de données télémétriques où il fera des merveilles. Sa topologie permet d'insérer facilement des nouveaux capteurs sans nécessiter de revoir l'organisation des entrées/sorties sur le microcontrôleur.

Du fait de son débit limité, le bus I2C n'est pas recommandé pour la transmission massive de données (image haute résolution, dalle d'affichage graphique, acquisition de données en haut débit).

© Tous droits réservés – 2023 – MC Hobby Srl

6.5.Bus SPI

Le bus SPI (*Serial Peripheral Interface*) est un bus 4 fils utilisé pour les transferts de gros volumes de données. Il permet, par exemple, d'afficher le contenu d'une dalle graphique de 640x480 pixels en une fraction de seconde. Le bus SPI est également très apprécié pour l'acquisition de données en temps réel.



02RI96 – Topologie d'un bus SPI

Par exemple, un afficheur 640 x 480 pixels en couleurs 16 bits (donc 2 octets par pixel) nécessite un transfert de 640 * 480 * 2 = 614400 octets (600 Kio) pour rafraîchir l'image sur l'afficheur. Sur un bus I2C à plein débit (400 Kbits/sec), cela nécessiterait pas moins de 12 secondes alors que sur un bus SPI à 20 Mhz (20 millions de bits par seconde) le temps de transfert est réduit à (640*480*2) / (20000000/8) = 0,25 secondes !

Le bus SPI est présent sur presque tous les microcontrôleurs du marché. Il demande cependant un savoir faire un peu plus pointu, ce qui le rends moins accessible aux Makers.

Un bus SPI fonctionne en Full-Duplex, ce qui signifie que le microcontrôleur, comme le périphérique, peut communiquer dans les deux sens en même temps (transmettre et recevoir des données en une seule opération).

Un bus SPI est composé de 4 signaux fondamentaux :

•**SCK** : Signal d'horloge (*Serial CloK*), permet d'ordonnancer les échanges entre le microcontrôleur et le périphérique.

•**MOSI** : (*Master Out Slave In*) est le canal qui transmet des informations du microcontrôleur (le maître) vers le périphérique. Ce signal est branché sur la broche du même nom sur le périphérique.

•**MISO** : (*Master In Slave Out*) est le canal de communication du périphérique vers les microcontrôleurs. Ce signal est également branché sur la broche du même nom de l'autre côté du bus.

•**CSn** : (*Chip Select*) est aussi appelé SS (*Slave Select*). C'est un signal inversé qui est donc « actif » lorsqu'il est au niveau bas, c'est pour cela qu'il est souvent accompagné d'un « n » minuscule ou d'un « / ». Le signal CSn permet d'activer le périphérique SPI pour qu'il écoute et traite les informations communiquées sur le bus. Ce signal est aussi utilisé par certains périphériques SPI comme signal de transaction. Il n'est donc pas rare de le voir changer plusieurs fois d'états durant une communication entre le microcontrôleur et le périphérique SPI cible.

© Tous droits réservés – 2023 – MC Hobby Srl

Aide mémoire MicroPython

Les quelques lignes suivantes indiquent comment exploiter bus SPI sur base de son numéro de bus (cfr : Raspberry-Pi Pico, Pico en détails – brochage du Pico).

```
>>> from machine import SPI, Pin
>>> # MISO=gp4, MOSI=gp7, SCK=gp6
>>> spi = SPI( 0, miso=4, mosi=7, sck=6 )
>>> cs = Pin( 5, Pin.OUT, value=1 )
```

Il est important de noter que le signal CSn est géré séparément du bus à l'aide d'un objet `Pin`. Par ailleurs, le signal CSn est initialisé au niveau haut pour éviter de démarrer une nouvelle transaction sur le bus SPI.

Il est possible de ralentir le bus SPI en ajoutant le paramètre `baudrate` (en baud, bits par seconde), ce qui peut être utile situation de prototypage.

```
>>> from machine import SPI, Pin
>>> spi = SPI( 0, baudrate=100000 )
>>> cs = Pin( 5, Pin.OUT, value=1 )
```

Pour plus d'informations sur bus SPI, voir la documentation MicroPython sur le lien suivant :

https://docs.micropython.org/en/latest/library/machine.SPI.html

Plusieurs périphériques SPI

Le bus SPI peut être utilisé pour communiquer avec plusieurs périphériques. Dans ce cas de figure, il faut un signal CSn complémentaire par périphérique SPI additionnel branché sur le bus. Il va de soit qu'un seul périphérique est activé à la fois.

Avantages et inconvénients

Dans le rang des avantages, il y a le débit élevé et le transfert simultané dans les deux sens.

Le développement autour d'un bus SPI s'accompagne de deux inconvénients :

1.Un bus SPI ne dispose pas d'état d'erreur comme sur le bus I2C. Il n'y a pas, non plus, de LED d'erreur qui clignote sur les périphériques. Par conséquent, il est impossible (ou presque) de détecter une erreur de communication ou de configuration durant la transmission des données vers le périphérique SPI.

2.Un bus SPI fonctionne selon 4 modes différents identifié par la combinaison des paramètres de polarité (`polarity`) et de phase (`phase`) du signal d'horloge. Information à fournir lors de l'initialisation du bus SPI.

Le développement de ces points dépasse largement le cadre de cet ouvrage. Il sera cependant facile de trouver des informations complémentaires sur Internet.

Electronique, histoire des USA et législation

Voici une bien étrange association ! Des considérations qui n'ont, à vrai dire, normalement rien à faire dans un tel ouvrage… pourtant, le passé lointain des États-Unis à ressurgit plus récemment en provoquant de terribles émeutes raciales. Cela n'est pas sans un impact surprenant de ce côté ci de l'océan.

© Tous droits réservés – 2023 – MC Hobby Srl

Durant les années 2020, les problèmes raciaux aux États-Unis ont conduit à la création du mouvement « Black Live Matter » que l'on traduit pas « la vie des noirs compte ». L'esclavagisme ayant aussi fait partie de l'histoire Américaine, l'évocation même d'esclavage, et toutes notions attachées, est aujourd'hui devenu un tabou chez nos amis anglo-saxons.

En résulte aujourd'hui un négationnisme d'une histoire pourtant bien réelle, négationnisme qui éradique l'existence même des notions et termes d'esclavage au plus profond de la société américaine. Il est aujourd'hui très mal venu de faire référence à ce passé.

Cela impacte aussi le monde de la programmation et de l'électronique où les notions et topologies maîtres/esclaves sont aujourd'hui renommées pour les faire disparaître. La langue anglaise étant omniprésente dans ces domaines, cela à aussi un impact par chez nous.

Ainsi, le MOSI (*Master Out Slave In*) d'un microcontrôleur s'appelle aussi DO (*Data Out*) ou TX (transmission) tandis que le MISO côté périphérique s'appelle DI (*Data In*) ou RX (réception).

Le but final étant de réaliser les connexions croisée TX → RX dans les deux sens comme pour les liaisons séries/UARTs qui portent déjà la terminologie RX/TX (et avec lesquels il ne faudra pas confondre les bus SPI).

Dans le même ordre d'idée, le signal SS (*Slave Select*) est aussi communément appelé CS (*Chip Select*), le maître est appelé *Leader* et les esclaves *Follower* (suiveur).

A contrario, en Europe, le négationnisme est généralement interdit par force de loi dans de nombreux pays. De sorte, la terminologie MISO, MOSI, SS reste présent et usité.

En conclusion

Le bus SPI sera plutôt destiné à la mise en œuvre de périphériques nécessitant un débit important mais reste cependant plus difficile à mettre en œuvre pour un nouveau périphérique SPI.

6.6.UART (port série)

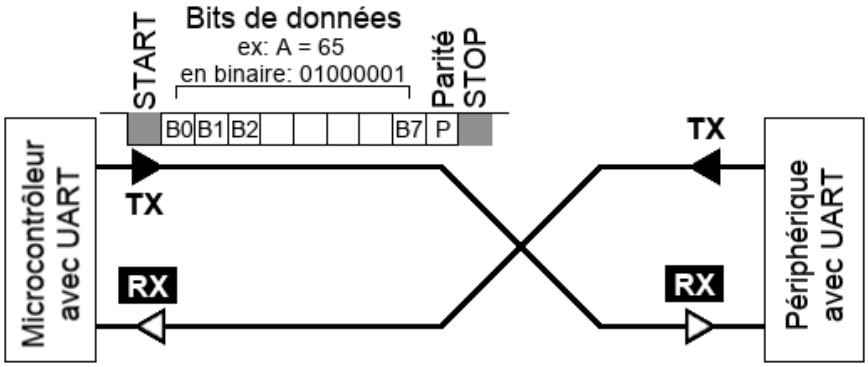
Abusivement appelé port série, l'UART (*Universal Asynchronous Receiver-Transmitter*) est un périphérique permettant d'établir une liaison de type série.

Le port série fait référence à l'interface RS232 qui, elle, exploite des tensions de -12V à +12V (même si, au final, ce port RS232 est lui même raccordé sur un UART).

Pour sa part, l'UART fonctionne au niveau logique du microcontrôleur, donc entre 0 et 3,3V pour le Raspberry-Pi Pico.

La liaison série fait référence au fait que l'information est transmise bit à bit sur un fil de transmission.

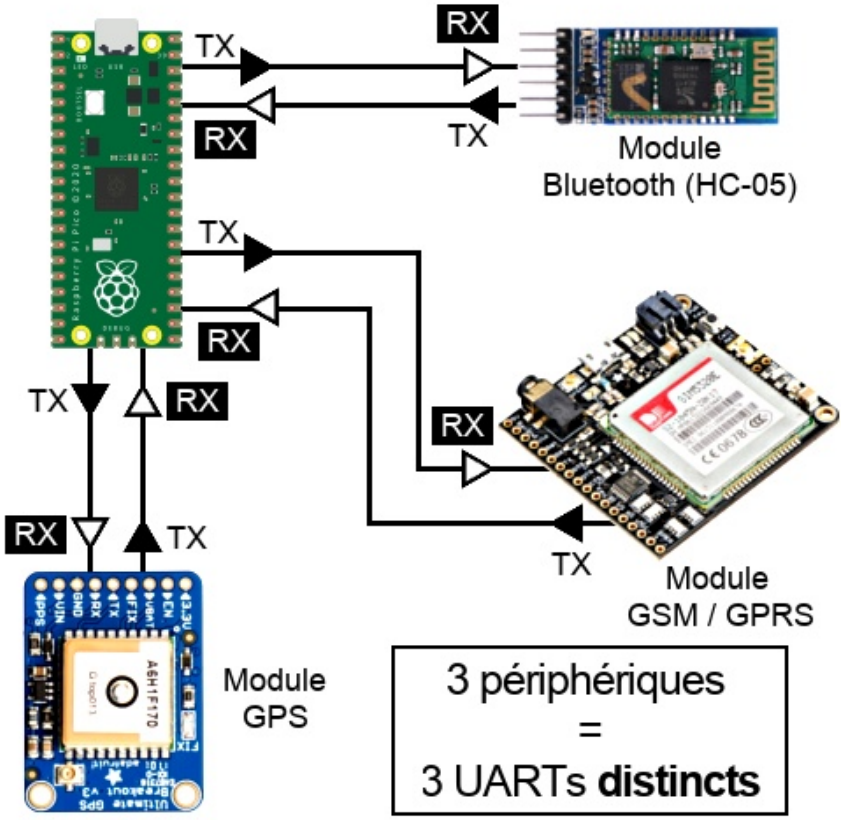
© Tous droits réservés – 2023 – MC Hobby Srl



02RI84 – Principe d'une liaison série

Une connexion UART s'établit en croissant les signaux TX (Transmission) et RX (Réception) entre les deux périphériques. Il s'agit d'une communication point-à-point en full-duplex (dans les deux sens en même temps). Sur un port série, tout comme un UART, la communication peut débuter (ou être suspendue) à l'importe quel moment par le microcontrôleur ou le périphérique.

L'UART s'utilise avec des périphériques tels que modem GSM, interface WiFi/Bluetooth ou encore un module GPS.



02RI85 – utilisation type d'UARTs

Il faut autant d'UART disponible sur le microcontrôleur qu'il n'y a de périphériques à connecter.

UART et terminal

© Tous droits réservés – 2023 – MC Hobby Srl

Un UART peut être utilisé avec logiciel Terminal (comme Putty et WinTerm) afin d'établir une communication homme-machine entre le microcontrôleur et un ordinateur.

Cette fonctionnalité permet de prendre le contrôle à distance depuis un ordinateur.

Ces dernières années ont vu l'apparition de microcontrôleurs disposant d'un support USB natif. Cela permet, entre autre, d'établir une **liaison série par l'intermédiaire de la connexion USB** (donc sans réserver des broches sur le microcontrôleur pour y établir une liaison UART).

Cette particularité est largement exploitée par MicroPython pour communiquer avec la carte.

Aide mémoire MicroPython

Les quelques lignes suivantes indiquent comment créer un UART sur base de son numéro de bus (cfr : Raspberry-Pi Pico, Pico en détails – brochage du Pico).

La configuration par défaut est 8 bits de données, pas de parité, 1 bit de stop (8N1, la plus communément utilisée). Ne reste plus qu'à lui adjoindre le débit parmi les valeurs usuelles 9600, 19200, 28800, 115200 bauds pour ne citer que les valeurs les plus courantes.

```
>>> from machine import UART
>>> ser0 = UART( 0, 9600 ) # 8N1, TX=gp0, RX=gp1
>>> # Modifier le parametrage de l'UART
>>> ser0.init(4800, bits=1, parity=None, stop=2)
```

Il est possible de trouver plus d'informations sur l'UART dans la documentation de MicroPython.

https://docs.micropython.org/en/latest/library/machine.UART.html

6.7.Horloge RTC

Le Pico dispose d'une horloge RTC interne (RTC pour *Real Time Clock* signifiant Horloge Temps Réel). Cependant, le microcontrôleur doit rester sous tension pour qu'elle fonctionne.

Equipée d'une pile bouton, une horloge RTC continue d'égrainer le temps alors même que le microcontrôleur est hors tension.

Cependant, le Pico ne permet pas de brancher une pile bouton et, par conséquent, il ne peut pas rester à l'heure lorsqu'il est mis hors tension !

Il y a deux solutions à cet épineux problème :

D'une part, il est possible de maintenir le Pico sous tension à l'aide d'une paire de pile AA pour garder sa RTC en fonctionnement. Cela implique également de garder le Pico sous tension, ce qui limite la durée de vie de l'ensemble

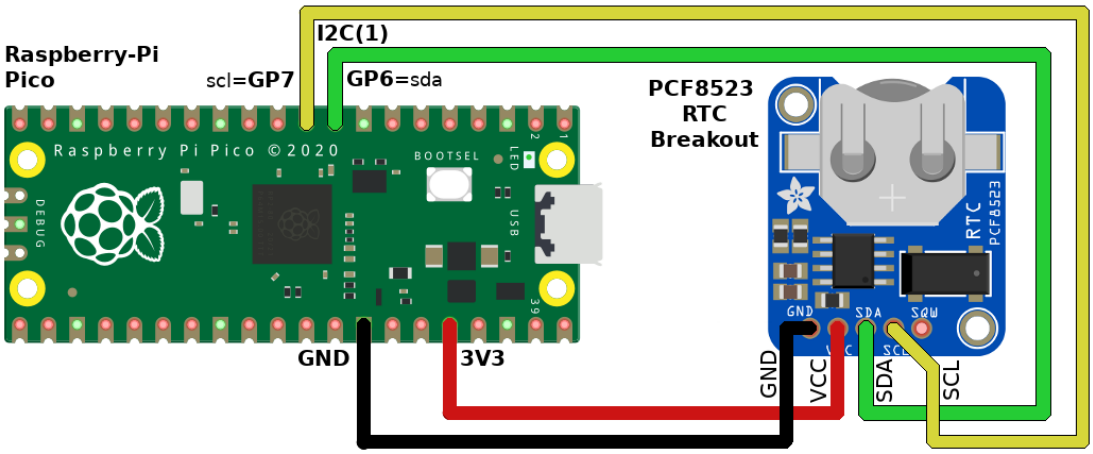
Si ces informations ci-dessous semblent denses et potentiellement confuses en ce début d'ouvrage elles pourront, un jour, se montrer fort utiles.

D'autre part, en adjoignant un module RTC externe au Pico (ex : PCF8523, Adafruit 5189, voir le schéma de raccordement ci-dessous), il devient possible de maintenir l'heure en toutes circonstances puisque ce module RTC dispose d'une pile bouton.

© Tous droits réservés – 2023 – MC Hobby Srl

Le Pico dispose des éléments logiciels (classe RTC) permettant de gérer l'écoulement du temps dès sa mise sous tension. Par défaut, le temps s'écoule à partir du 1ier janvier 2021.

Au démarrage du Pico il sera nécessaire de transférer l'heure depuis l'horloge RTC externe vers l'horloge RTC interne du Pico.



02RI98 – Brancher un module RTC externe sur un Pico

L'exploitation de ce module passe par le pilote MicroPython PCF8523. Un pilote, c'est un script python (`pcf8523.py` en l'occurrence) contenant une ou plusieurs classes (ce sera `PCF8523` dans ce cas) prenant en charge les détails de la communication avec le périphérique matériel pcf8523.

Une fois le pilote (`pcf8523.py`) copié sur la carte MicroPython, il est possible d'exécuter quelques lignes de code python pour obtenir l'heure stockée dans le module PCF8523 et transférer celle-ci dans l'horloge du Pico.

Les opérations de copie de fichiers/scripts et exécution seront abordés plus loin dans cet ouvrage.

L'exemple ci-dessous interroger l'horloge externe puis transfère l'heure dans l'horloge interne du Pico.

Par la suite, il sera bien plus efficace d'interroger l'horloge interne du Pico. En effet, interroger l'horloge externe implique de communiquer sur le bus I2C plus lent que les bus au coeur du microcontrôleur.

```
>>> from pcf8523 import PCF8523
>>> from machine import I2C
>>> import time
>>> i2c = I2C(0)
>>> rtc_ext = PCF8523( i2c ) # Horloge externe
>>> rtc_ext.datetime         # temps en secondes
1650759490
>>> time.localtime( rtc_ext.datetime ) # Date Horloge externe
>>> # annee, mois, jour, heure, min, sec, jour_semaine, jour_année
>>> # jour_semaine : 0=Lundi, 1=Mardi, ... 6=Dimanche
(2022, 4, 24, 0, 26, 6, 6, 114)
>>>
>>> # Transférer la date vers la RTC interne du Pico
>>> # format (year, month, day, weekday, hours, minutes, seconds,
>>> #   subseconds)
>>> from machine import RTC
>>> _now = time.localtime( rtc_ext.datetime ) # Date Horl. externe
```


© Tous droits réservés – 2023 – MC Hobby Srl

```
>>> rtc_mcu = RTC()    # RTC du Pico
>>> # Injecter la date dans la RTC du Pico
>>> rtc_mcu.datetime( (_now[0],_now[1],_now[2],_now[5],
                       _now[3],_now[4],_now[5],0) )
>>>
>>> # Maintenant l'horloge Pico est a l'heure
>>> # Interroger l'horloge du Pico doit retourner le bonne heure
>>> rtc_mcu.datetime()
(2022, 4, 24, 6, 0, 43, 18, 0)
```

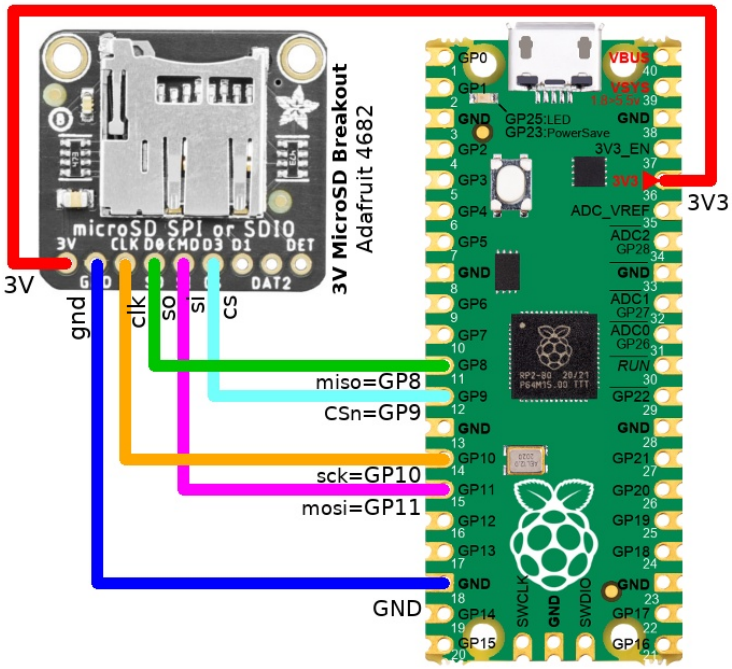
Le lecteur attentif notera la différence de format entre le résultat de la fonction localtime() et l'ordre des paramètres d'initialisation pour l'horloge interne du Pico.

6.8.carte MicroSD

Bien que courant sur les microcontrôleurs plus puissants, le Raspberry-Pi Pico ne dispose pas de connecteur pour carte SD/microSD. D'une façon générale, ce n'est pas un problème car le système de fichiers MicroPython sur la Pico est bien assez grand pour couvrir plus de 95 % des besoins.

A défaut, d'autres cartes RP2040 avec module de mémoire Flash plus spacieux, il est aussi possible de brancher un connecteur microSD sur le Raspberry-Pi Pico.

Les projets nécessitant un stockage plus volumineux (ou amovible) peuvent brancher un connecteur microSD sur le Raspberry-Pi Pico à l'aide d'une carte breakout (Adafruit, 4682) et de quelques fils. Cela permet de lire des cartes au format vFat.



02RI99 – Raccorder un connecteur MicroSD sur le Pico

L'utilisation d'une carte microSD passe par l'utilisation de la bibliothèque `sdcard.py` disponible sur le dépôt GitHub de MicroPython.

https://github.com/micropython/micropython-lib/tree/master/micropython/drivers/storage/sdcard

© Tous droits réservés – 2023 – MC Hobby Srl

Une fois `sdcard.py` copié sur la carte MicroPython, la classe `SDCard` devient accessible. Classe qui permet d'accéder au système de fichiers en montant la carte SD dans le système de fichier MicroPython.

Monter un système de fichier est une opération bien connu du monde Unix/Linux où le contenu d'un média (la carte SD) est associé à un répertoire sur le système de fichiers existant. Accéder à la carte SD se fait donc par l'intermédiaire d'un répertoire où tout le contenu du média est accessible.

Les quelques lignes Python ci-dessous indiquent comment monter la carte micro-SD dans le répertoire `sd` du système de fichiers MicroPython.

```
>>> from machine import Pin, SPI
>>> from os import VfsFat, mount
>>> # sdcard.py téléchargé depuis le dépôt GitHub MicroPython
>>> from sdcard import SDCard

>>> spi = SPI(1, sck=Pin(10, Pin.OUT), mosi=Pin(11, Pin.OUT),
miso=Pin(8, Pin.OUT), baudrate=0x14<<20)
>>> cs = Pin(9, Pin.OUT, value=True)
>>> sd_card = SDCard(spi, cs)
>>> vfs = VfsFat(sd_card)
>>> mount(vfs, "/sd")
```

Le système de fichier MicroPython reprend maintenant un répertoire `sd` qu'il est possible d'explorer pour y découvrir deux fichiers textes.

```
>>> os.listdir()
['sd', 'ili934x.py', 'pcf8523.py', 'pm25.py', 'veram_m15.bin']
>>> os.listdir('sd')
['textual.txt', 'ascii-art.txt']
```

L'ouverture, la lecture et l'écriture des fichiers sur la carte SD se fait à l'identique de n'importe quel autre fichier présent sur le système de fichiers MicroPython.

```
>>> with open( '/sd/ascii-art.txt', 'r' ) as f:
...     s = f.readline()
...     while s != '':
...         print( s.replace('/n','').replace('/r','') )
...         s = f.readline()
...
```

Ce qui dans le cas de cet exemple produit l'affichage suivant :

© Tous droits réservés – 2023 – MC Hobby Srl

02RI100 – Logo MicroPython, Ascii-art

7.Le timer

Il s'agit ici d'une fonctionnalité qui trouve logiquement sa place dans ce chapitre. Il s'agit cependant d'une technique avancée plus rarement exploitée. Ainsi, il sera peut-être opportun de survoler cette section pour y revenir plus tard.

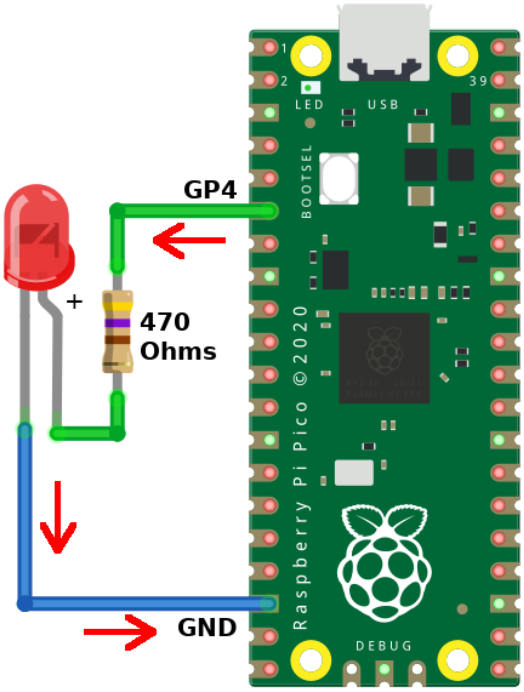
Le `timer` à, proprement parler, n'est pas une interface matérielle vers le monde extérieur comme l'est le bus I2C ou les broches d'entrées/sorties.

Un `timer` est un élément matériel permettant d'interrompre l'exécution du programme principale pour transférer l'exécution à un morceau de code spécifique appelé routine d'interruption (dit *Interrupt Handler*). Après l'exécution de celle-ci, le programme principal reprend le cours de son exécution.

Un `timer` est utilisé pour exécuter la routine à intervalle régulier ou après un délai spécifique (et une seule fois) après l'activation du `timer`. D'autres configurations sont possibles mais sortent largement du cadre de cet ouvrage.

L'exemple ci-dessous exploite un `timer` et une routine d'interruption pour changer l'état d'une LED sur GP4 sans perturber le fonctionnement général du microcontrôleur.

© Tous droits réservés – 2023 – MC Hobby Srl



02RI90 – LED branchée sur GP4

Voici les quelques étapes de mise en place des éléments permettant d'atteindre l'objectif.

```
1: >>> from machine import Pin, Timer
2: >>> p4 = Pin( 4, Pin.OUT )
3: >>>
4: >>> def routine( tim ):
5: >>>     global p4
6: >>>     p4.toggle()
7: >>>
8: >>> mon_tim = Timer( period=1000, mode=Timer.PERIODIC,
callback=routine )
9: >>>
```

Voici les explications concernant les quelques lignes ci-dessus saisies dans une session REPL.

•Ligne1 : importation des classes `Pin` et `Timer`.

•Ligne2 : création d'une instance de `Pin` pour commander le GPIO 4 en sortie.

•Ligne 4 à 6 : définition de la routine d'interruption qui prend le `timer` en paramètre (communiqué par la machine virtuelle MicroPython lors de l'appel).

•Ligne 5 : l'instruction `global` informe la fonction de rappel que la variable `p4` est une variable globale.

•Ligne 6 : la méthode `toggle()` de `p4` permet d'inverser l'état de broche à chaque appel.

•Ligne 8 : Crée une instance de `Timer` et configure le timer en mode périodique pour un appel toute les 1000 millisecondes (soit toutes les secondes). Cette initialisation transmet également un référence vers la fonction de rappel `routine` qui sera appelée à intervalle régulier. Enfin, une référence du `timer` est stockée dans la variable `mon_tim` pour pouvoir y avoir accès si besoin.

© Tous droits réservés – 2023 – MC Hobby Srl

A partir de la saisie de la ligne 8, la LED sur le GPIO 4 se met à clignoter de façon totalement autonome en changeant d'état toutes les secondes. L'utilisateur peut continuer à saisir des instructions ou démarrer un script.

Il est également possible de désactiver le `timer` en le « dé-initialisant ». Cela se fait en appelant `mon_tim.deinit()` .

Une routine d'interruption doit être aussi courte que possible et ne peut, en aucun cas, allouer de la mémoire depuis la routine d'interruption. Il est plutôt recommandé de manipuler des variables globales pour transmettre des informations.

© Tous droits réservés – 2023 – MC Hobby Srl