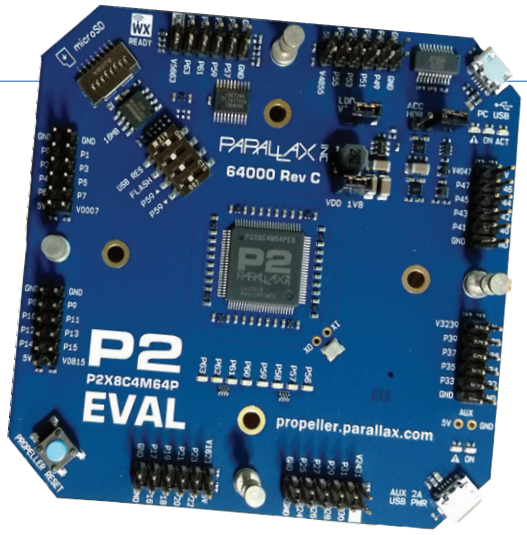
Propeller 2 de Parallax (3)

Faire clignoter une LED



Mathias Claussen (Elektor)

Dans les articles précédents, nous avons allumé une LED. Passons au clignotement. L'idée est de poursuivre la découverte des broches « intelligentes » pour créer un UART (émetteur-récepteur série universel asynchrone) capable de transmettre des caractères. Commençons par établir la communication avec le Propeller 2 de Parallax !

Dans les deux premiers articles de cette série, nous avons commencé à écrire du code pour le microcontrôleur Propeller 2 et allumé une LED avec un programme écrit en Spin2. Maintenant faisons clignoter cette LED et continuons à découvrir les broches d'E/S.

Une solution simple

Vous pouvez suivre un schéma simple : « allumer la LED, puis l'éteindre ». Pour ce faire, vous avez besoin des fonctions suivantes :

- `pinwrite()`
- `repeat()`
- `waitms()`

Les étapes successives sont :

- Allumer la LED
- Attendre 500 ms
- Éteindre la LED
- Attendre 500 ms
- Répéter

La **figure 1** montre le code. Vous pouvez le télécharger dans la page web de l'article [1].
Après avoir utilisé les broches en tant que sorties, l'étape suivante serait de montrer comment les utiliser en entrée. Cependant, nous n'allons pas le faire à ce stade et nous reviendrons sur ce sujet plus tard. Le fait de disposer d'une sortie de données en série pour communiquer avec un PC fait de l'affichage d'un état lu sur une broche d'E/S quelque chose de beaucoup plus intéressant que le simple clignotement d'une LED. De plus, nous pouvons utiliser cette sortie pour envoyer des données d'état et faire un peu de débogage du code. La prochaine étape de cette série va concerner les broches « intelligentes ».

Broches intelligentes

Aujourd'hui, les équipes de marketing adorent qualifier les produits d'« intelligents » (par ex. « ville intelligente », « données intelligentes », « contrats intelligents »). Avec les broches dites intelligentes, c'est une autre histoire, car elles vont bien au-delà des broches d'E/S habituelles à usage général. Sur d'autres microcontrôleurs, vous avez parfois la possibilité de sélectionner plusieurs fonctions pour une broche spécifique. Certains, comme l'ESP32, possèdent une matrice d'E/S qui peut acheminer n'importe quel signal d'E/S des périphériques internes vers n'importe quelle broche. Dans ce cas, une broche d'E/S ne sera toujours qu'une broche d'E/S et les fonctions UART, SPI ou CA/N se trouveront dans un bloc de matériel spécialisé, simplement connecté à la broche elle-même. Avec les broches intelligentes du Propeller 2, c'est différent car les périphériques spécialisés ne sont plus des blocs de fonctions spécifiques à l'intérieur du microcontrôleur, acheminés par une matrice d'E/S, mais ils sont plutôt intégrés, au moins partiellement, à chaque broche d'E/S. D'où le terme de *broche intelligente*.

Figure 1. Code pour faire clignoter une LED.

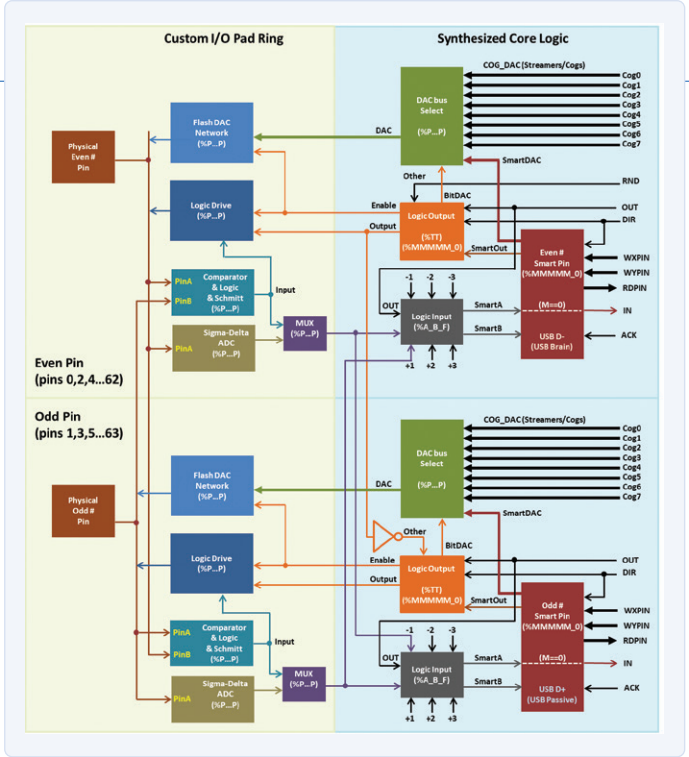


Figure 2. Présentation des broches intelligentes. (Source : Rayman/Parallax, http://bit.ly/parallax-smartpin)

L'utilisateur 'rayman' a fort à propos fait un excellent travail sur le forum Parallax [2] en apportant à la communauté un aperçu du fonctionnement interne des broches intelligentes. Vous pouvez trouver sa publication sur le Propeller 2 et l'image en [3], ou voir la **figure 2**. Dans l'encadré « Vue d'ensemble des fonctions des broches intelligentes », j'ai incorporé un extrait de la fiche technique du Propeller 2. Comme vous pouvez le voir, une broche peut servir à plusieurs fonctions. Pour le moment, c'est la configuration *11110** = *émission série asynchrone* pour envoyer des données à des fins de débogage ultérieur qui nous intéresse. La première étape consiste à déterminer comment définir le mode approprié de la broche, et si c'est possible avec Spin2 ou si nous devons y ajouter un peu de langage assembleur.

Configuration de l'UART

C'est là que le mal de tête peut arriver si c'est la première fois que travaillez avec Spin2 et le Propeller 2. La fiche technique actuelle est relativement complète. Mais tout lire et comprendre peut nécessiter beaucoup de temps. Notre objectif est d'obtenir une simple fonction, `tx()`, qui émet un caractère sur une broche, comme un UART. Nous choisissons les paramètres suivants : 115200 bauds, 8 bits de données, pas de parité et un seul bit d'arrêt. La première étape consiste à configurer la broche.

Pour ce faire, procédez comme suit :

- Configurer la broche en émission série asynchrone
- Définir le débit en bauds et les bits de données
- Activer la broche intelligente

Ces trois étapes simples permettent de paramétrer une broche sous la forme d'un UART en mode émission. Comme par la suite nous modifierons le code et le réutiliserons, nous le plaçons dans une fonction. Une fonction contient simplement du code ou des fragments de code souvent utilisés dans un programme. Vous évitez ainsi un copier-coller dans le code et vous limitez par ailleurs la maintenance à cet emplacement unique. Nous allons donc utiliser une fonction et l'appeler `serial_start`. Celle-ci ne comporte aucun argument et se résume aux trois étapes indiquées ci-dessus. La broche utilisée est actuellement codée « en dur » comme broche 57 (l'une des broches de la LED également accessible sur l'un des connecteurs périphériques, comme le montre la **figure 3**). La fonction commence par le préfixe `PUB` suivi de son nom. L'accolade finale est vide, car il n'y a pas d'arguments.

```
PUB serial_start()
WRPIN( 57, %01_11110_0  ) 'définit le mode émission
   asynchrone pour la broche tx
```

Vue d'ensemble des fonctions des broches intelligentes

00000	broche intelligente désactivée (par défaut)
00001	référentiel long (P[12:10] != %101)
00010	référentiel long (P[12:10] != %101)
00011	référentiel long (P[12:10] != %101)
00001	bruit CN/A (P[12:10]= %101)
00010	bruit de dispersion 16 bits CN/A, bruit (P[12:10]= %101)
00011	bruit de dispersion 16 bits CN/A, PWM (P[12:10]= %101)
00100*	sortie impulsion/cycle
00101*	sortie transition
00110*	fréquence oscillateur à commande numérique (NCO)
00111*	service oscillateur à commande numérique (NCO)
01000*	triangle PWM
01001*	dent de scie PWM
01010*	PWM alimentation à découpage, retour V et I
01011	périodique/continu : codeur à quadrature A-B
01100	périodique/continu : aug. sur montée A & B haut
01101	périodique/continu : aug. sur montée A & B haut / dim. sur montée A & B bas
01110	périodique/continu : aug. sur A haut {/ dim. sur montée B}
01111	périodique/continu : aug. sur A haut {/ dim. sur B haut}
10000	durée des états continus sur entrée A
10001	durée des états hauts continus sur entrée A
10010	durée de X états hauts/montées/fronts sur entrée A ou temporisation sur X états hauts/montées/fronts sur entrée A
10011	pendant X périodes, mesure du temps
10100	pendant X périodes, comptage des états
10101	pour des périodes de X cycles d'horloge minimum, mesure du temps
10110	pour des périodes de X cycles d'horloge minimum, comptage des états
10111	pour des périodes de X cycles d'horloge minimum, comptage des périodes
11000	échantillonnage/filtrage/capture CA/N, horloge interne
11001	échantillonnage/filtrage/capture CA/N, horloge externe
11010	oscillo CA/N avec déclencheur
11011*	hôte/dispositif USB (paire de broches paire/impaire = DM/DP)
11100*	émission série synchrone (données A, horloge B)
11101	réception série synchrone(données A, horloge B)
11110*	émission série asynchrone (débit en bauds)
11111	réception série asynchrone (débit en bauds)
* signal OUT forcé	
PWM = modulation de largeur d'impulsion	

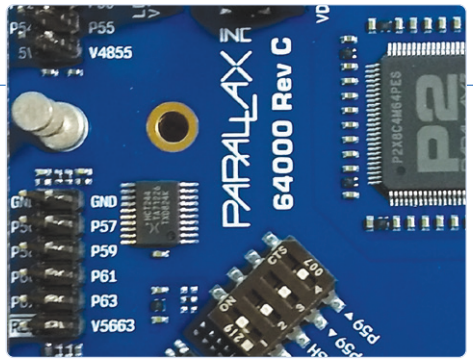


Figure 3. Emplacement de l'embase pour avoir accès aux broches de la LED.

```
PUB serial_start()
  WRPIN( 57, %01_11110_0  )              'set async tx mode for txpin
  WXPIN( 57, ((217<<16) + (8-1 )) )      'set baud rate to sysclock/115200 and word size to 8
  org
   dirh    #57
  end
```

Figure 4. Code de la fonction `serial_start()`.

```
PUB tx(val)

      WYPIN(57,val) 'load output word
org

      WAITX   #1              'wait 2+1 clocks before polling busy
      wait
      RDPIN   val,#57  WC     'get busy flag into C
      IF_C    JMP     #wait           'loop until C = 0

end
```

Figure 5. Code de la fonction `tx`.

```
WXPIN( 57, ((217<<16) + (8-1 )) )
'débit en bauds = sysclock/115200 et taille du mot = 8 bits
org' début de la partie en assembleur
dirh #57
end   'fin de la partie en assembleur
```

À partir de la ligne 1, nous avons la fonction, comme mentionné plus haut – et plus précisément, l'*en-tête de fonction*. La ligne suivante règle la broche 57 en *émission série asynchrone*, ce qu'indique la valeur 11110. Le premier bit est toujours à zéro ; les deux bits les plus significatifs, ici '01', indiquent que la broche doit être pilotée par la fonction GPIO ou Smart Pin (broche intelligente). Nous utilisons ici la fonction Spin2

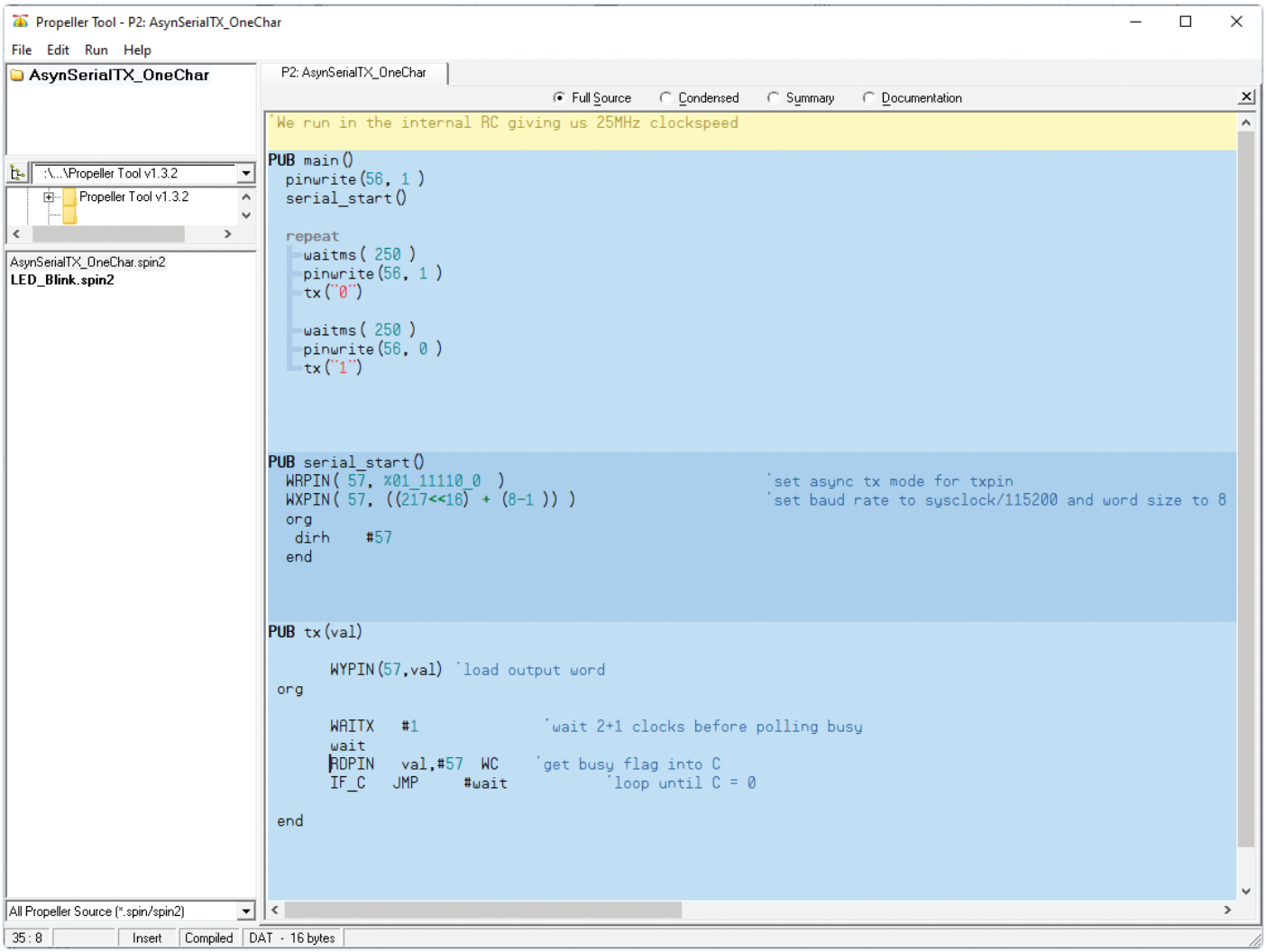


Figure 6. Code complet.

`WRPIN` pour effectuer notre première étape. La fonction suivante, `WXPIN`, définit le diviseur d'horloge et les bits de données à utiliser pour une broche intelligente en mode série asynchrone. Pour simplifier, nous ne tenons pas compte pour l'instant de la partie concernant le diviseur de débit d'émission. Il est possible de calculer la valeur de ce débit en appliquant la formule horloge système / débit en bauds – ici 25 MHz/115200 bauds – ce qui donne env. 217. Ce résultat doit être décalé de 16 vers la droite. Pour le nombre de bits à transférer, nous utilisons la formule (bits souhaités – 1), ce qui nous laisse (8 – 1) bits. Toute la magie de la configuration du débit d'émission et des bits de données est là. Les trois lignes suivantes sont différentes du code précédent. Comme il s'agit de montrer une petite section d'assembleur, quelques mots d'explications supplémentaires sont nécessaires. Avec l'indication `org`, vous pouvez incorporer une section d'instructions en assembleur, nécessaires ici. La commande `dirh` activera les fonctions de broche intelligente comme prévu pour notre étape 3. Ce qui est différent, c'est la façon d'indiquer le numéro de la broche sur laquelle nous travaillons, car il doit commencer par un '#'.

La dernière ligne ferme la section en assembleur. Dans ce cas particulier, elle correspond également à la fin de la fonction elle-même. Il aurait été préférable d'éviter l'assembleur, mais il n'y a actuellement aucun équivalent Spin2 pour l'instruction `dirh`. La **figure 4** montre le code formaté.

La broche étant dans le bon mode, nous pouvons continuer et mettre en place une fonction pour émettre un caractère et attendre jusqu'à ce que ce soit fait. Il est possible de récupérer cette fonction dans la fiche technique [4], page 91. Nous avons vu comment créer des fonctions sans arguments, ce qui signifie qu'aucune information ne leur est transmise. Pour émettre un caractère, il serait utile de pouvoir le passer à la fonction. Comme nous essayons d'éviter autant que possible l'assembleur, nous utiliserons, le cas échéant, les fonctions Spin2 plutôt que l'assembleur intégré.

Émission

Pour l'émission, nous créons une fonction `tx` presque identique à la fonction `serial_start()` comme le montre la **figure 5**.

La différence visible, outre le nom, est l'argument `val` entre parenthèses. `val` contiendra le caractère à imprimer. À l'intérieur de la fonction, nous allons d'abord écrire la valeur dans le registre de transmission de la broche 57 avec la commande `WYPIN`. La section suivante contient à nouveau quelques lignes de code en assembleur. Nous devons attendre que le drapeau 'occupé' de l'émetteur ne soit plus levé et que l'émission soit terminée. D'après la fiche technique, il faut patienter pendant trois cycles de l'unité centrale pour lire le drapeau en toute fiabilité. Cette attente est réalisée par l'instruction `WAITX` avec le paramètre `#1`, car son exécution prend deux cycles + la valeur spécifiée pour la fonction (ici un cycle). La ligne suivante est une étiquette appelée `wait`, en langage assembleur. Nous pourrons y accéder ultérieurement. L'instruction `RDPIN` en assembleur, comme écrit ici, sert à lire l'état de la broche avec report. Ce report est signalé par l'indication `WC` à la fin de l'instruction. Le bit de report, qui sert ici de drapeau d'occupation, est important car il indique si l'émission est terminée.

`RDPIN val, #57 WC` lit l'état, y compris le bit de report dans notre valeur `val`. Alors que le contenu est en cours d'émission, nous pouvons réutiliser la mémoire de `val` pour y lire l'état de la broche intelligente. L'ultime commande `IF_C JMP #wait` est un saut conditionnel ; en BASIC, ce serait l'équivalent du fameux `GOTO` combiné à une

instruction `IF`. En clair, cela signifie : *le bit de report est-il levé (ici le drapeau 'occupé') ? Revenir à l'éqtiquette 'wait' et recommencer à partir de là, sinon continuer.* Notre émission est considérée comme terminée si le bit de report n'est plus levé. La fonction s'exécutera donc jusqu'à sa fin et reviendra là où elle a été appelée.

Assemblage des différentes parties

Nous pouvons maintenant assembler le code et insérer, après chaque fonction `pinwrite()`, l'émission d'un « 0 » ou d'un « 1 » en incorporant les fonctions `tx("0")` ou `tx("1")` dans notre code, comme le montre la **figure 6**.

Pour capturer la sortie, connectez un convertisseur série USB. À cet effet, nous avons utilisé notre fidèle Logic 16 et enregistré la sortie de la LED et l'émission en série. Le résultat apparaît dans la **figure 7** et la **figure 8**.

Et pour transmettre une chaîne de caractères ? Peut-on envoyer un simple `print("Hello World")` sur la liaison série comme nous en avons l'habitude dans l'univers Arduino ? Oui, c'est possible et nous le ferons dansle prochain article.

(200479-C-04)

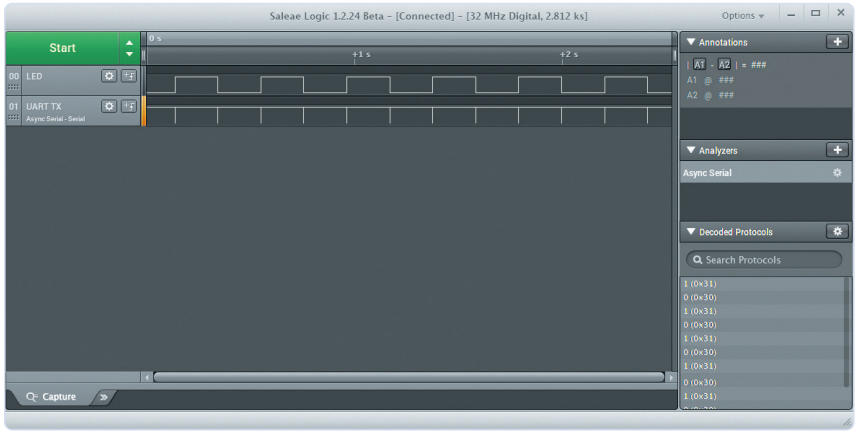


Figure 7. Tracé de l'analyseur logique avec un caractère transmis toutes les 500 ms.

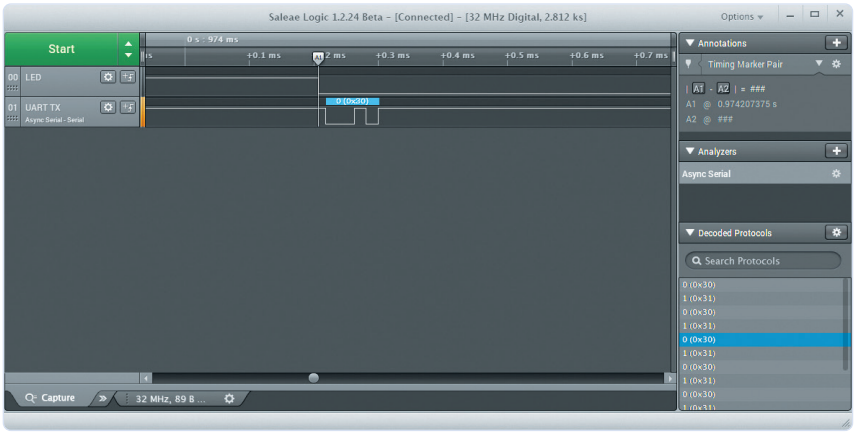
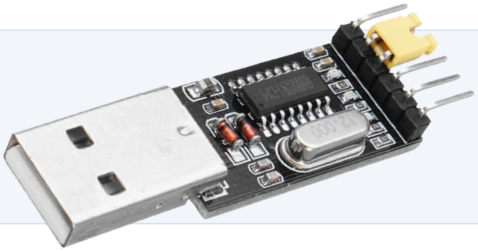


Figure 8. Tracé agrandi montrant le caractère émis.

Des questions, des commentaires ?
Contactez Elektor (redaction@elektor.fr).

Contributeurs
Conception et texte : **Mathias Claußen**
Rédaction : **Jens Nickel et C. J. Abate**
Traduction : **Pascal Godart**
Mise en page : **Giel Dols**

PRODUITS



Convertisseur USB-TTL CH340G (3,3 V/5,5 V)
www.elektor.fr/ch340-usb-to-ttl-converter-uart-module-ch340g-3-3-v-5-5-v

LIENS

[1] **Page de l'article** : www.elektormagazine.fr/200479-C-04
[2] **Forum Parallax** : https://forums.parallax.com
[3] **Vue d'ensemble des broches intelligentes, rayman** : https://forums.parallax.com/discussion/171420/smartpin-diagram-now-with-p-p-bit-mode-table/p8
[4] **Feuille de caractéristiques du Propeller 2 (préliminaire)** : https://docs.google.com/document/d/1gn6oaT5Ib7CytvlZHacmrSbVBJsD9t_-kmvjd7nUR6o/edit#heading=h.1h0sz9w9bl25