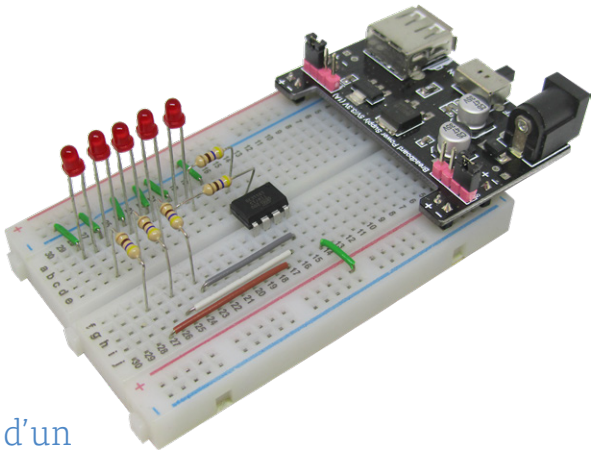
Explore ATtiny Microcontrollers
using C and Assembly Language

Extrait : ports d'E/S d'ATtiny



Warwick A. Smith (Afrique du Sud)

Les ports d'E/S contrôlent les broches d'un microcontrôleur et leur permettent d'être configurées un à un en entrée ou en sortie. C'est une notion si générale, valable dans n'importe quel cours de débutant ou d'introduction à la programmation des microcontrôleurs. Cependant, afin de comprendre et d'exploiter les capacités d'E/S d'un microcontrôleur, vous devez étudier plus profondément le circuit. Dans cet article, Warwick Smith, auteur d'un livre publié par Elektor, en fait une démonstration avec des exemples de programmation en assembleur du célèbre micro ATtiny. Intéressé ? Jetons un œil !

Note de l'éditeur. Cet article est un extrait du livre de 376 pages *Explore ATtiny Microcontrollers using C and Assembly Language* (W. Smith, Elektor, 2021) formaté et légèrement modifié pour correspondre aux normes éditoriales et à la mise en page du magazine Elektor. Puisque cet article est extrait d'une publication plus vaste, certains termes peuvent faire référence à des passages du livre d'origine situés ailleurs. L'auteur et l'éditeur ont fait de leur mieux pour l'éviter et seront heureux de répondre aux questions. Pour les contacter, voir l'encadré « **Des questions, des commentaires ?** ».

Les ports d'E/S sont configurés et contrôlés en utilisant un ensemble de quatre registres dans ATtiny13(A) et ATtiny25/45/85. Il est possible de lire le niveau logique d'une broche configurée en entrée à l'aide d'un programme exécuté sur l'AVR. Si un interrupteur y est connecté, le niveau logique déterminera s'il est ouvert ou fermé. Lorsqu'une broche est configurée en sortie, elle peut être utilisée pour commuter le niveau logique de la broche vers le haut (niveau logique 1) ou vers le bas (niveau logique 0). Une broche de sortie peut être utilisée pour piloter une LED, comme dans le projet de clignotement de LED décrit ailleurs dans ce livre.

Configuration des broches en sorties en assembleur

Dans cette section, nos configurons plus qu'une broche en sortie en utilisant un ATtiny 8 broches avec cinq LED connectées avec des résistances en série. Ensuite, nous créerons un code pour réaliser un compteur binaire de 5 bits qui compte de zéro avec ces LED.
La figure 1 montre un schéma de circuit d'un microcontrôleur AVR ATtiny13(A) ou ATtiny25/45/85 avec cinq LED connectées aux broches E/S PB0 à PB4. La broche PB5 du microcontrôleur ATtiny dans le circuit est utilisée comme broche debugWIRE pour la programmation et le débogage. Notez que pour utiliser cette configuration, il faut utiliser un programmateur/débogueur, tel que Atmel-ICE, ou AVR Dragon.

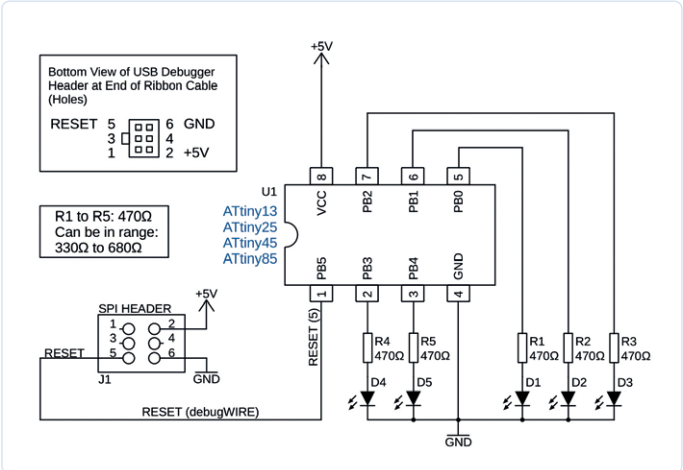


Figure 1. Schéma du circuit du compteur à cinq LED.

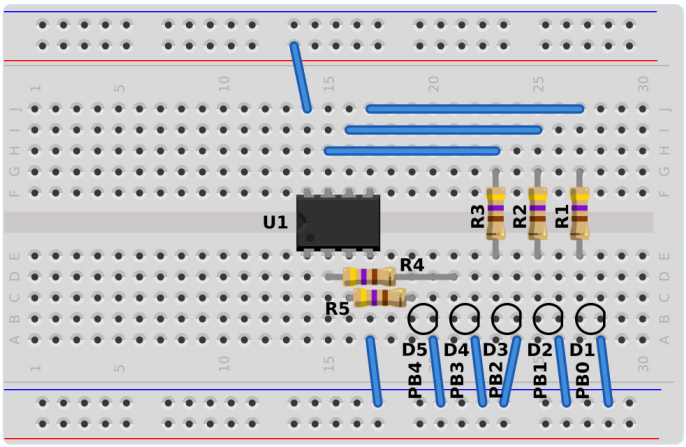


Figure 2. Disposition du circuit de compteur à cinq LED sur la platine à essai.

Un programmateur USB (program-only) ne fonctionnera pas en mode debugWIRE, et ne pourra pas effectuer de débogage.

Note : **N'activez pas le fusible `DWEN` avec un programmateur USB (program-only), car l'AVR ne pourra pas quitter le mode debugWIRE.** L'utilité du mode debugWIRE avec cet exemple de circuit est de libérer les autres broches de l'AVR qui sont normalement utilisées en mode ISP/SPI.

Programmateurs program-only

Les programmateurs USB avec uniquement des capacités de programmation peuvent être utilisés avec le circuit de la **figure 1** pour charger le programme d'exemple qui suit et voir la valeur du compteur affichée sur les LED.

Connectez les LED comme indiqué dans la **figure 1** et la **figure 2.** Le design original du USBasp et celui de USBtinyISP sont munis de résistances de protection sur les lignes qui sortent de ces programmateurs, pour les protéger, ainsi que la puce AVR cible. L'Arduino Uno programmé comme un ArduinoISP n'a pas de résistances de protection, mais il est possible d'en ajouter sur la ligne MOSI et la ligne SCK de l'Arduino Uno dédié à l'AVR cible. Des résistances de protection de 270 Ω sont utilisées dans la conception originale de l'USBasp, et sont placées sur les lignes MOSI, SCK et RESET. Des résistances de 1,5 kΩ sont utilisées sur les lignes MOSI et RESET de la conception originale de l'USBtinyISP.

Les résistances de protection empêchent un court-circuit si l'une des broches de l'AVR cible conduit en même temps une tension de sortie de polarité inverse que celle du programmateur.

Périphériques interférant avec la programmation

Bien que le circuit de la **figure 1** puisse être programmé à l'aide de l'interface ISP/SPI avec les LED et les résistances en série, d'autres circuits peuvent avoir du matériel qui interfère avec la programmation. Il existe différentes solutions à ce problème. Bien sûr, les lecteurs qui disposent d'un programmateur/débogueur USB doté de debugWIRE, tel que l'Atmel-ICE, peuvent simplement mettre l'AVR cible en mode debugWIRE, et ainsi utiliser une seule broche pour la programmation. Une autre solution est de programmer l'AVR sur une autre platine d'essai en utilisant ISP/SPI, et de le brancher ensuite sur le circuit cible.

On peut aussi utiliser un AVR avec plus de broches, mais celles du port libre ne correspondent pas toujours à celles du même port d'un ATtiny PDIP 8 broches. Par exemple, si l'on utilise les broches de port PB0 à PB4 comme dans le circuit de la **figure 1**, on ne dispose pas de cinq broches libres consécutives sur la gamme d'AVRs ATtiny24/44/84 à 14 broches, parce que les broches ISP/SPI utilisent celles du port A et du port B. Sur la gamme ATtiny26/261/461/861 20 broches, le port A est entièrement libre avec un programmateur ISP/SPI connecté, mais cela signifie que le logiciel doit être modifié pour utiliser le port A au lieu du port B. Heureusement, la gamme ATtiny2313/4313 20 broches a les broches PB0 à PB4 libres avec un programmateur ISP/SPI connecté.

Mettre l'AVR en mode debugWIRE

Pour que le microcontrôleur puisse être programmé à l'aide d'une seule ligne debugWIRE comme le montre la **figure 1**, il est nécessaire de connecter d'abord toutes les lignes du connecteur ISP au microcontrôleur à partir du programmateur/débogueur USB, puis de configurer le fusible `DWEN` de l'AVR pour le mettre en mode debugWIRE. Votre AVR est maintenant en mode debugWIRE. Si ce n'est pas le cas, connectez votre programmateur/débogueur, tel qu'un Atmel-ICE ou un AVR Dragon, et assurez-vous que vous êtes prêt et capable de régler le fusible `DWEN`. Une fois le que le fusible `DWEN` est programmé, toutes les connexions du connecteur ISP peuvent être démontées du circuit à l'exception de RESET, +5 V (Vcc) et GND comme le montre la **figure 1**.

Construire le circuit ATtiny 5-LED sur une platine d'essai

Si vous disposez du matériel nécessaire, construisez le circuit de la **figure 1** sur une plaque d'essai. Assurez-vous que les cinq LED sont connectées en une ligne avec les broches PB0 à PB4 connectées dans l'ordre en commençant par PB0 à droite. La LED D1 sera donc à droite de la rangée et D5 à gauche. Nous voulons avoir une seule rangée de LED avec PB0 connecté à la LED de droite, PB1 connecté à la LED voisine de gauche, PB2 connecté à la troisième LED en commençant par la droite, et ainsi de suite, comme on peut le voir sur le schéma de la **figure 2**. L'image ne montre que les contours des LED afin qu'elles ne masquent pas les connexions des fils et des résistances. Si vous ne disposez pas du matériel nécessaire, suivez les programmes à l'aide d'un simulateur.

Code assembleur pour le circuit du compteur à 5 LED

Créez un nouveau projet en assembleur AVR sur Microchip Studio nommé *led_count_asm*. Tapez le code montré dans le **Listage 1** dans le fichier *main.asm* du projet en remplaçant le programme de base. Si vous utilisez le matériel de la **figure 1**, sélectionnez l'outil matériel (votre débogueur), par exemple Atmel-ICE sous Microchip Studio, avec debugWIRE comme interface. Pour ce faire, cliquez sur l'icône *hammer* dans la deuxième barre d'outils supérieure. Si vous utilisez le simulateur, sélectionnez *simulator* comme outil. Si vous utilisez un programmateur « maison », poursuivez la lecture pour voir comment télécharger le programme sur l'ATtiny cible.

Le programme *led_count_asm* configure les broches PB0 à PB4 en

sortie de sorte que les LED associées puissent être commandées par le code. Il affiche un nombre binaire croissant, ou compte, sur les LED en commençant par 0 (représenté par toutes les LED éteintes). Lorsque le compteur atteint sa valeur maximale (toutes les LED allumées), il revient à zéro et recommence à compter. Chaque LED « éteinte » représente un chiffre binaire 0 ou un niveau bas, et chaque LED « allumée » représente un chiffre binaire 1 ou un niveau haut. Il est important de disposer les LED comme indiqué sur la **figure 2** afin que le compte s'affiche correctement avec PB0/D1 comme LSB (bit de poids faible) et PB4/D5 comme MSB (bit de poids fort) de la valeur du compteur.

Si vous utilisez le matériel de la **figure 1** et de la **figure 2**, créez le programme et téléversez-le sur l'AVR. Si vous utilisez un Atmel-ICE ou un AVR Dragon, utilisez l'icône Start Without Debugging dans la barre d'outils supérieure de Microchip Studio ou le raccourci clavier *Ctrl + Alt + F5* pour télécharger le programme dans l'AVR. Si l'interface du débogueur est correctement configurée et que la puce est en mode debugWIRE, le programme sera chargé et commencera à fonctionner. Le compte binaire incrémentiel sera indiqué par les LED. Si vous utilisez un programmateur USB *program-only*, ou un que vous avez conçu vous-même, chargez alors le programme sur l'AVR cible en utilisant la fonction appropriée. Si le circuit a été câblé convenablement, et le programme a été saisi sauvegardé et compilé correctement, vous verrez la valeur du compte binaire s'incrémenter sur les LED et vous pourrez utiliser le simulateur dans la suite.

Si vous ne disposez pas des composants nécessaires, vous pouvez toujours voir la valeur du comptage en utilisant le simulateur de Microchip Studio. Le programme peut être exécuté progressivement en utilisant les icônes *Start Debugging* et *Break*. Après avoir démarré le simulateur, ouvrez la fenêtre I/O dans le menu supérieur en cliquant sur *Debug Windows I/O*, puis cliquez sur l'élément *I/O Port (PORTB)*. Exécutez le programme en utilisant l'icône *Step Over* ou la touche *F10* du clavier. Regardez l'élément PORTB au bas de la fenêtre I/O pour voir la valeur de compte qui aurait été affichée sur les LED si elles étaient connectées. Le compteur est mis à jour dans PORTB chaque fois que l'instruction `OUT` est exécutée dans la boucle principale.

Comment fonctionne le programme en assembleur LED Count

La moitié du code *led_count_asm* est constituée du sous-programme de délai qui a été utilisé dans le programme *LED blink* situé ailleurs dans ce livre. Ce sous-programme est appelé une fois dans la boucle principale afin que le comptage sur les LED soit visible à l'œil sans un clignotement trop rapide. Lancez le projet *led_count_asm* dans Microchip Studio, avec le code de *main.asm* également ouvert, tout en suivant l'explication du code qui suit.

Les deux premières instructions du programme sont utilisées pour configurer les broches PB0 à PB4 en sortie pour piloter les LED en configurant les bits du registre DDRB. La **figure 3** montre le registre DDRB en haut. Chaque bit de ce registre correspond à une broche du microcontrôleur. Par exemple, le bit DDB0 correspond à la broche PB0, DDB1 à la broche PB1, et ainsi de suite.

Lorsqu'un bit de DDRB est mis à 1 logique, la broche correspondante est configurée en sortie. Si un bit dans DDRB est mis à 0 logique, la broche correspondante est configurée en entrée qui est la configuration par défaut de toutes les broches à la mise sous tension ou à la réinitialisation. Au début du programme, 0b0001_1111 est écrit dans le registre R16, puis écrit à partir de R16 dans le registre DDRB en utilisant la commande `OUT`. Il est nécessaire de charger d'abord la valeur constante dans R16 car il n'y a pas d'instruction pour le faire directement dans un registre d'E/S tel que DDRB. L'écriture de 0b0001_1111 dans DDRB active les bits DDB0 à DDB4 ce qui définit les broches PB0 à PB4 en sortie.

Listage 1 : led_count_asm : main.asm

```
    ; Set up pins PB0 to PB4 as output pins
     ldi     r16, 0b0001_1111
     out     DDRB, r16
     clr     r18              ; Clear count register
loop:
     out     PORTB, r18       ; Display count on LEDs
     rcall   delay
     inc     r18              ; Increment count
     andi    r18, 0b0001_1111 ; Clear unused bits
     rjmp    loop
; Delay subroutine
delay:
     ldi     r16, 0xff
delloop1:
     ldi     r17, 0xff
delloop2:
     dec     r17
     brb     SREG_Z, delloop2
     dec     r16
     brbc    SREG_Z, delloop1
     ret
```

DDRB : Port B Data Direction Register. Configures pin direction: 0 = Input, 1 = Output.
Address: 0x17 **Initial Value:** 0b0000_0000 **Bits 5 to 0:** Read/Write (R/W)

7	6	5	4	3	2	1	0
–	–	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0

PORTB : Port B Data Register. Drives output pins. Enables input pin pull-ups with logic 1.
Address: 0x18 **Initial Value:** 0b0000_0000 **Bits 5 to 0:** Read/Write (R/W)

7	6	5	4	3	2	1	0
–	–	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0

PINB : Port B Input Pins Register. Read input pin state. Write logic 1 to toggle output pin.
Address: 0x16 **Initial Value:** Depends on level on pin. **Bits 5 to 0:** Read/Write (R/W)

7	6	5	4	3	2	1	0
–	–	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0

Figure 3. Registres des ports d'E/S d'ATtiny13(A) et ATtiny25/45/85.

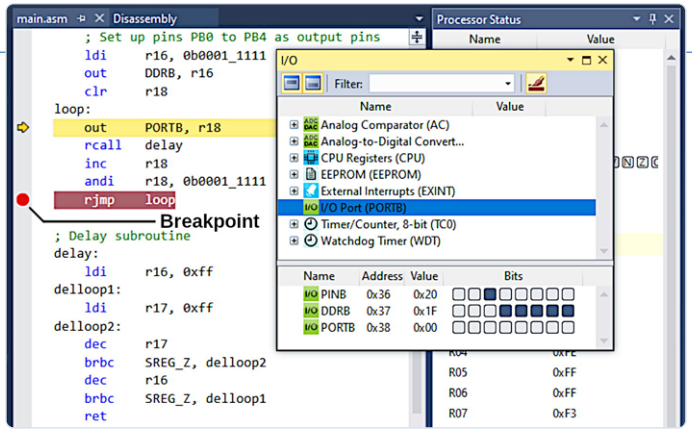


Figure 4. Insertion d'un point d'arrêt dans le débogueur de Microchip Studio.

Bien qu'il y ait quatre registres pour contrôler le port d'E/S B, seuls trois sont représentés sur la **figure 3**. Le quatrième registre, MCUCR, n'a qu'un seul bit utilisé par le port B. Ce bit est un bit global de pull-up que nous n'utilisons pas dans ce chapitre. Référez-vous à la partie description des registres de la fiche technique dans la section ports d'E/S pour voir ce registre - consultez la fiche technique de l'ATtiny13, ATtiny13A, ou ATtiny25/45/85.

R18 est utilisé dans le programme pour maintenir une valeur de compte incrémentielle qui est affichée sur les LED. R18 est mis à 0 à l'aide de `CLR` avant d'entrer dans la boucle principale afin que le compte commence à partir de 0.

Dans la boucle principale, la commande OUT permet d'envoyer le contenu de R18 au registre PORTB. Ce dernier est représenté au milieu de la **figure 3**. Encore une fois, chaque bit dans ce registre correspond à une broche sur le microcontrôleur. Les niveaux logiques écrits dans le registre PORTB apparaissent sur les broches qui ont été configurées en sortie en utilisant DDR. Pour les broches qui sont configurées en entrée, 1 logique dans PORTB active une résistance de rappel haut interne sur ces broches. 0 logique désactive la résistance de rappel haut sur la broche correspondante.

Comme les broches PB0 à PB4 sont configurées en sortie, la valeur de compte écrite au PORTB est représentée par les valeurs logiques qui allument et éteignent les LED connectées. Le niveau logique 1 dans la valeur de comptage allume la LED correspondante, et le niveau logique 0 dans la valeur de comptage éteint la LED correspondante.

Après que la valeur de compteur soit écrite sur PORTB en utilisant la commande `OUT`, le sous-programme de délai est appelé pour la maintenir affichée sur les LED pendant une courte durée. La valeur sur le registre R18 est alors incrémenté de 1 en utilisant la commande `INC`. `ANDI` est utilisée pour effacer les trois bits supérieurs de la valeur de compte dans R18, en utilisant une valeur de masque de 0b0001_1111. Cela garantit que ces bits de poids fort sont toujours à zéro. De retour au début de la boucle, lorsque R18 est à nouveau écrit dans PORTB, 0 est toujours écrit dans les trois bits de poids fort, car ils ont été effacés en utilisant `ANDI`. Lorsque la fin de la boucle est atteinte à l'instruction `RJMP`, l'exécution du programme recommence, avec la nouvelle valeur du compteur écrite dans PORTB, et affichée sur les LED.

Notez que dans ce programme : les registres d'E/S DDRB et PORTB sont tous deux utilisés pour configurer et contrôler le port B du microcontrôleur dans les deux programmes. Le programme LED blink n'a besoin de contrôler qu'une seule broche, il utilise donc les instructions `SBI` et `CBI` pour activer et désactiver un seul bit dans les registres d'E/S directement — aucun des registres de travail R0 à R31 n'est nécessaire pour cela. Le programme de compteur à LED a besoin d'accéder à cinq LED ou broches en même temps et utilise donc l'instruction `OUT` pour écrire sur plusieurs bits dans un registre en même temps.

L'utilisation des registres dans un programme en langage assembleur est indispensable pour suivre son progrès. Par exemple, une valeur immédiate est chargée dans R16 au début du programme. R16 n'est utilisé que temporairement dans ce cas, et peut donc être utilisé plus tard sans avoir besoin de sauvegarder sa valeur au préalable. Il est à nouveau utilisé dans le sous-programme de délai. Comme R16 et R17 sont utilisés dans le sous-programme de retard, R18 a été choisi pour contenir la valeur de comptage uniquement. Cela garantit que la valeur du compteur n'est jamais écrasée. Si un programme devient très large et qu'il n'y a pas de registres libres à utiliser pour des objectifs dédiés, alors les commandes `PUSH` et `POP` peuvent être utilisées au début et à la fin du sous-programme pour maintenir les valeurs dans les registres en cours d'utilisation.

Utilisation d'un point d'arrêt dans le débogueur

En utilisant le simulateur, ou le matériel et un débogueur tel que l'Atmel-ICE , utiliser le débogueur pour progresser dans le programme en cliquant sur *Start Debugging* et *Break* comme nous l'avons fait précédemment. Visualisez les registres du port d'E/S en ouvrant *I/O Window* dans Microchip Studio. Lorsque le débogueur est en cours d'exécution, sélectionnez *Debug Windows I/O*, puis cliquez sur *I/O Port (PORTB)* dans *I/O window*, comme illustré à la **figure 4**.

Lorsque l'exécution du programme entre dans la boucle principale, il est pratique de placer un point d'arrêt sur la commande `RJMP` et d'exécuter la boucle entière en cliquant sur le bouton *Continue* de la barre d'outils (touche F5). Un point d'arrêt peut être placé sur la commande `RJMP` en cliquant sur la zone grise située à l'extrême gauche de l'instruction et de la fenêtre du Microchip Studio. Cela fait apparaître un point rouge dans cette zone, indiquant qu'un point d'arrêt a été placé sur l'instruction, comme on peut le voir sur la **figure 4**. Vous pouvez également cliquer sur la commande sur laquelle vous souhaitez placer un point d'arrêt afin que le curseur soit placé dessus, puis sélectionner *Debug Toggle Breakpoint* dans le menu supérieur ou utiliser la touche F9 du clavier. Une fois le point d'arrêt défini, utilisez le bouton *Continue* de la barre d'outils ou appuyez sur la touche F5 pour parcourir toute la boucle et ne l'interrompre qu'en bas. De cette façon, on peut voir le compte s'incrémenter de 1 dans le PORTB dans la fenêtre I/O ainsi que dans le registre R18 dans la fenêtre *Processor Status*, sans avoir à parcourir individuellement chaque instruction de la boucle.

Supprimez le point d'arrêt de la commande `RJMP` en cliquant sur le point rouge à gauche de l'instruction, ou désactivez-le en utilisant le menu ou la touche F9, nous supposons que le curseur est sur la ligne de la commande `RJMP`. Placez un point d'arrêt sur la commande `INC R18`. Utilisez maintenant F5 pour progresser dans la boucle. Le nouveau point d'arrêt garantit que le registre PORTB et R18 contiennent la même valeur lorsque l'exécution du programme s'arrête. Si l'exécution du programme s'arrête sur `RJMP`, alors R18 est incrémenté avant d'écrire sa nouvelle valeur à PORTB, donc ils sont désynchronisés lors de l'observation de ces valeurs dans les

fenêtres du débogueur dans Microchip Studio.
Le logiciel associé au livre publié par l'auteur peut être téléchargé gratuitement. Visitez [1], descendez jusqu'aux téléchargements, et cliquez sur le nom du fichier : *Software_Explore ATtiny Microcontrollers using C and Assembly Language*. Save the ZIP archive file locally (environ 29 kb) et extrayez-le.

220045-04

Contributeurs

Texte et image : **Warwick Smith**
Éditeur : **Jan Buiting**
Mise en page : **Giel Dols**
Traduction : **Asma Adhimi**

Des questions, des commentaires ?

Envoyez un courriel à l'auteur (warwsmi@axxess.co.za) ou contactez Elektor (redaction@elektor.fr).

LIEN

[1] Ressources et informations sur le livre: www.elektor.fr/20007

PRODUITS

› **Livre en anglais « Explore ATtiny Microcontrollers using C and Assembly Language » de Warwick A. Smith**
Version papier : www.elektor.fr/20007
Version numérique : www.elektor.fr/20008