je programme avec
des automates finis
en assembleur et en C sur des PIC à 8 bits

Andrew Pratt (Royaume-Uni)

Saviez-vous qu'il existait une *autre* façon de programmer les PIC ? C'est ce qu'explique le livre *Programming the Finite State Machine — with 8-Bit PICs in Assembly and C* en adoptant une approche pratique. Les outils de programmation, dont l'assembleur gpasm pour PIC, peuvent être installés sur Windows ou sur Linux. Les extraits reproduits ici sont tirés du deuxième chapitre intitulé The Assembly Program *as a Finite State Machine*.

Un clignotant à LED plus complexe

L'assembleur a la réputation d'être difficile à lire. Il est vrai que même le code que l'on vient juste d'écrire semble parfois abscons. Outre la mise en forme elle-même de ce code, le diagramme d'état favorise aussi la lisibilité d'un programme en fournissant une vue d'ensemble de son déroulement. États et transitions entre états y sont représentés au moyen de blocs et de flèches. Un tel diagramme facilite la lecture du code assembleur.

Pour vous en convaincre, étudiez la **figure 2-5**. Le diagramme est à nouveau celui d'une LED, mais cette fois-ci sa séquence de clignotement est plus complexe [que celle de la section précédente du livre] : la LED clignote lentement 3 fois, clignote ensuite rapidement 10 fois, puis revient au cycle lent. Notez que le diagramme a quatre états et six transitions. Les quatre états sont les deux états **on** et **off** pour le cycle rapide, et les deux états **on** et **off** pour le cycle lent. Le diagramme peut être divisé en quatre parties au moyen de deux lignes : une séparant les états **on** et **off**, l'autre séparant les cycles lent et rapide.

La **figure 2-6** montre l'oscillogramme de la tension aux bornes de la LED.

De la compréhension du diagramme d'état devrait maintenant découler celle du programme (**listage 1**). Nous n'avions pas encore rencontré le signe **$**. Il représente l'adresse de la ligne actuelle, ce qui permet de brancher le programme sur une certaine ligne sans utiliser d'étiquette. Par exemple **GOTO $+3** effectue un saut en avant de 3 lignes.

Exécuter plus de deux automates dans un programme

Nous allons voir ici une façon d'exécuter simultanément plus d'un automate fini dans un même programme. La méthode ressemblera à celle que nous adoptons lorsque nous prétendons faire deux choses à la fois, alors qu'en réalité nous procédons par multiplexage temporel : nous effectuons une petite partie de la première tâche, passons à la seconde, revenons à la première, et ainsi de suite. Certaines unités distinctes du PIC telles que les temporisateurs (*timers*) peuvent effectivement fonctionner simultanément, mais le processeur du microcontrôleur ne peut traiter qu'une seule instruction à la fois. Si un programme requiert plus d'une tâche, celles-ci devront donc être exécutées dans autant de fils,

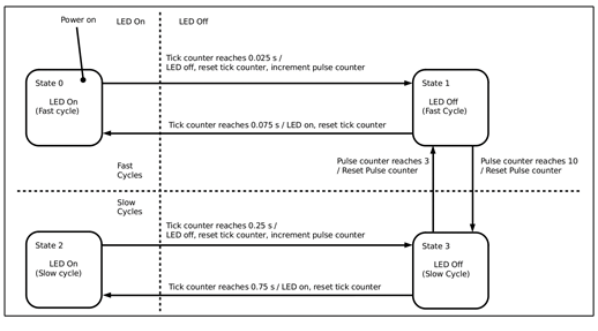

Figure 2-5. Diagramme d'état du programme de clignotement de LED de cycle.

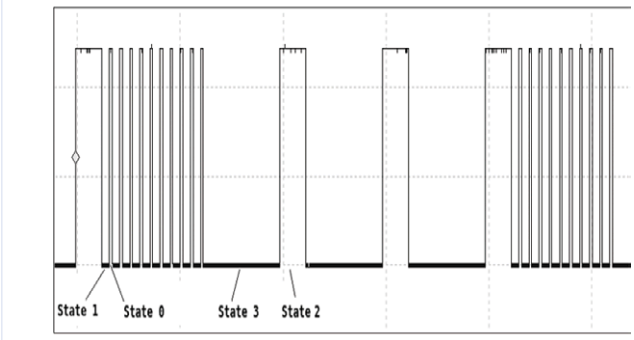

Figure 2-6. Oscillogramme du programme de cycle.

Listage 1. Programme Cycle de Cycle (prog_02_02.asm).

```
; prog_02_02.asm
; Page numbers refer to the data sheet DS40001413E.
; Tables refer to the book.

LIST     P=12f1822
#INCLUDE <p12f1822.inc>

    RADIX DEC                    ; Default numbers are to base 10.

    __CONFIG 0X8007, (_FOSC_INTOSC & _WDTE_OFF & _PWRTE_OFF & _CP_OFF & _BOREN_OFF & _IESO_OFF & _FCMEN_
   OFF )
    __CONFIG 0X8008, (_LVP_ON)


    CBLOCK 0x70                  ; In commom RAM, accessible in all banks.
    TICK_COUNTER
    PULSE_COUNTER
    ENDC

    ORG 0X00
    GOTO START

    ORG 0X04                     ; Interrupt vector.
    BCF INTCON, TMR0IF           ; Clear the tmr0 overflow interrupt flag. INTCON is in all banks.
    INCF TICK_COUNTER, F         ; Increment by one counter_off.
    RETFIE                       ; Return from interrupt
START
    MOVLB 1                      ; To access TRISA, OSCCON, OPTION_REG.
    BCF  TRISA, 2                ; Configure PORTA bit 2 (chip pin 5) as an output
    MOVLW 0x70
    MOVWF OSCCON                 ; Page 65 32MHz. See Table 2-1 in the book.
    MOVLW 0xD7
    MOVWF OPTION_REG             ; Page 164 in the data sheet and Table 2-2 in the book.
    MOVLW 0xE0
    MOVWF INTCON                 ; Page 86 in the data sheet and Table 2-3 in the book.
    MOVLB 0                      ; To access PORTA for the rest of the program.
    BSF PORTA, 2                 ; Inialise LED on;
;-----------Machine States.
S0                               ; LED on. Fast cycle.
    MOVLW 3
    SUBWF TICK_COUNTER, W        ; C in STATUS is set when TICK_COUNTER >= to 3.
    BTFSS STATUS, C              ; Skip the next instruction if C in STATUS is found set.
    GOTO S0                      ; C in STATUS found not set. Stay in this state.
    BCF PORTA, 2                 ; Turn LED off
    CLRF TICK_COUNTER            ; Reset the tick counter.
    INCF PULSE_COUNTER, F        ; Add 1 to PULSE_COUNTER and put the result in PULSE_COUNTER.
                                 ; Continue to S1

S1                               ; LED off. Fast cycle.
    MOVLW 10
    SUBWF PULSE_COUNTER, W       ; C in STATUS is set when PULSE_COUNTER >= to 10.
    BTFSS STATUS, C              ; Skip the next instruction if C in STATUS is found set.
    GOTO $+3                     ; Have not reached 10 pulses jump to ticks check.
    CLRF PULSE_COUNTER           ; 10 Pulses have been counted, reset PULSE_COUNTER.
    GOTO S3                      ; Transition to S3.
    MOVLW 9                      ; Load W with 9 for ticks check.
    SUBWF TICK_COUNTER, W        ; C in STATUS is set when TICK_COUNTER >= to 9.
    BTFSS STATUS, C              ; Skip the next instruction if C in STATUS is found set.
    GOTO S1                      ; C in STATUS found set, 9 ticks counted.
    CLRF TICK_COUNTER
    BSF PORTA, 2                 ; Turn LED on.
    GOTO S0

S2
    MOVLW 31
    SUBWF TICK_COUNTER, W        ; C in STATUS is set when TICK_COUNTER >= to 3.
    BTFSS STATUS, C              ; Skip the next instruction if C in STATUS is found set.
    GOTO S2                      ; C in STATUS found not set.
    BCF PORTA, 2                 ; Turn LED off
```

```
    CLRF TICK_COUNTER           ; Reset the tick counter.
    INCF PULSE_COUNTER, F       ; Add 1 to PULSE_COUNTER and put the result in PULSE_COUNTER.
                                ; Continue to S3

S3                              ; LED on. Slow cycle.
    MOVLW 3
    SUBWF PULSE_COUNTER, W      ; C in STATUS is set when PULSE_COUNTER >= to 3.
    BTFSS STATUS, C             ; Skip the next instruction if C in STATUS is found set.
    GOTO $+3                    ; ave not reached 3 pulses jump to ticks check.
    CLRF PULSE_COUNTER          ; Reset PULSE_COUNTER.
    GOTO S1                     ; Transition to S1.
    MOVLW 91
    SUBWF TICK_COUNTER, W       ; C in STATUS is set when TICK_COUNTER >= to 9.
    BTFSS STATUS, C             ; Skip the next instruction if C in STATUS is found set.
    GOTO S3                     ; C in STATUS found set.
    CLRF TICK_COUNTER
    BSF PORTA, 2                ; Turn LED on.
    GOTO S2

    END
```

chaque fil d'exécution partageant le « temps » du processeur. Pour obtenir ce *multi-threading*, le premier automate exécutera le bloc de code associé à son état en cours, puis ce sera au tour du second, et le programme reviendra au premier. Ces automates sont indépendants, ils ne font qu'utiliser le même processeur en temps partagé. Ils peuvent aussi partager la mémoire pour accéder aux données de l'autre.

Jusqu'à présent, l'état d'un automate était défini par le bloc de code dans lequel il se trouvait. Cet état sera maintenant défini par la valeur stockée dans un registre. Appelons **STATE_M0** le registre contenant l'état de l'automate 0, et **STATE_M1** celui de l'automate 1 (**M** pour « machine »). La **figure 2-7** montre le flux du programme. Nous utiliserons l'instruction **BRW** pour contrôler le passage d'un état à l'autre. **BRW** ajoute au compteur de programme la valeur contenue dans le registre W, autrement dit permet d'effectuer un saut déterminé. D'où son nom : **BR***anch with* **W**.

Le programme exécute le bloc de code d'état d'un des automates, puis se branche sur l'instruction de changement d'automate

Listage 2. Programme fictif ; l'automate M0 a deux états, M1 en a trois.

```
SWITCH_M0
     MOVFW STATE_M0         ; Moves the value of STATE_M0 to W.
     BRW                    ; The program will jump from here depending on W.
    GOTO M0_S0              ; The jump will be to this line if W = 0.
     GOTO M0_S1             ; The jump will be to this line if W = 1.

SWITCH_M1
     MOVFW STATE_M0         ; Moves the value of STATE_M0 to W.
     BRW                    ; The program will jump from here depending on W.
    GOTO M1_S0              ; The jump will be to this line if W = 0.
     GOTO M1_S1             ; The jump will be to this line if W = 1.
    GOTO M1_S2              ; The jump will be to this line if W = 1.
M0_S0
     -------------          ; code to decide if a change of state is needed if so change STATE_M0. GOTO
     SWITCH_M1              ; Go to the other machine's switch.

 M0_S1
     -------------          ; code to decide if a change of state is needed  if so change STATE_M0.
    GOTO SWITCH_M1          ; Go to the other machine's switch.

 M1_S0
     -------------          ; code to decide if a change of state is needed  if so change STATE_M1.
    GOTO SWITCH_M0          ; Go to the other machine's switch.

 M1_S1
     -------------          ; code to decide if a change of state is needed  if so change STATE_M1.
    GOTO SWITCH_M0          ; Go to the other machine's switch.

M1_S2
     -------------          ; code to decide if a change of state is needed  if so change STATE_M1.
    GOTO SWITCH_M0          ; Go to the other machine's switch.
```

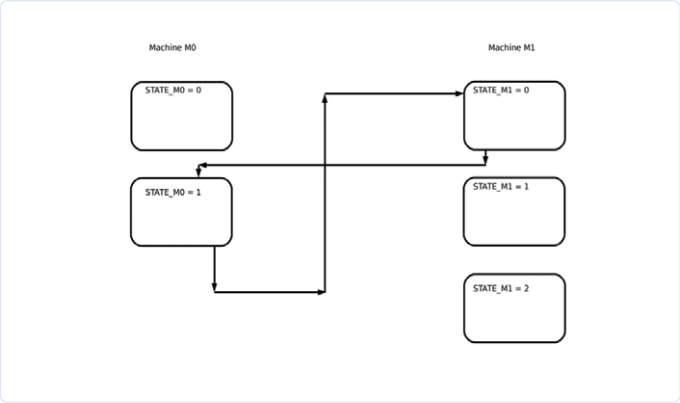

Figure 2-7. Exemple de flux de programme entre deux automates d'un même programme.

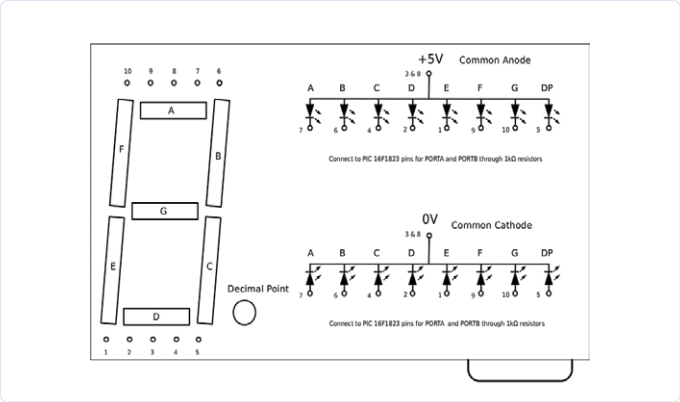

Figure 2-8. Connexions d'un afficheur à 7 segments standard.

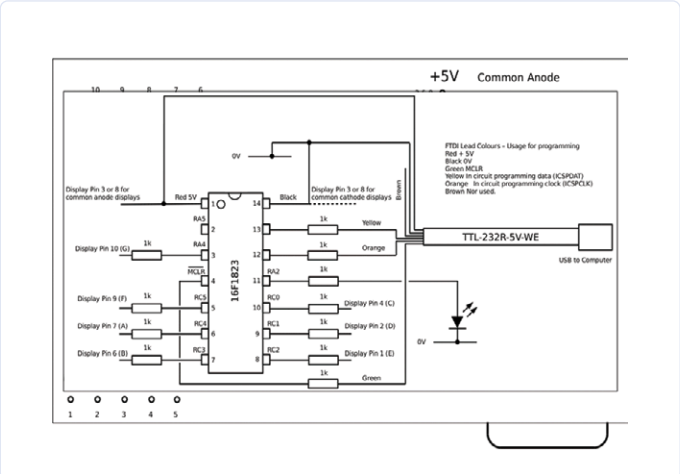

Figure 2-9. Câblage du PIC 16F1823 et de l'afficheur à LED.

(*SWITCH_Mx*) ; de là, l'instruction **BRW** passe à la ligne suivante si la valeur d'état est 0, saute une ligne si elle vaut 1, ou en saute deux si elle vaut 2.

Les « commutateurs » *SWTICH_Mx* assurent le flux continuel du programme entre les blocs d'état. Pour faire tourner deux automates sur deux fils d'exécution, nous avons donc eu besoin de deux commutateurs et de deux variables d'état, *STATE_M0* et *STATE_M1*. Référez-vous au code simplifié du **listage 2** pour comprendre le flux de ce programme fictif ; notez la façon dont il alterne entre les automates lorsqu'il quitte un bloc de code d'état. L'automate *M0* a deux états, *M1* en a trois.

Pilotage d'un afficheur LED à 7 segments

Illustrons l'exploitation simultanée de deux automates finis avec l'exemple d'un programme comptant le nombre de clignotements d'une LED et affichant ce nombre sur un afficheur à 7 segments. Nous avons le choix entre un afficheur à cristaux liquides et un afficheur à LED. Les écrans à LED consomment plusieurs milliampères par segment mais sont plus faciles à piloter car ils peuvent l'être avec un courant continu. Les LCD requièrent quant à eux une tension alternative. J'ai donc opté pour un modèle à LED. La **figure 2-8** montre les connexions d'une unité typique. Un afficheur peut être à cathode commune ou à anode commune. Dans le premier cas la broche commune doit être reliée au 0 V, dans le second elle se connecte au +5 V. La logique de commande des LED, et par suite l'écriture du code, dépendent de cette polarité. J'ai utilisé un modèle basique à anode commune. Si vous n'en disposez pas, le *LA-601VB* de Rohm ou le *SA52-11SRWA* de Kingbright conviendront.

Les LED à piloter étant au nombre de huit, nous utilisons ici le PIC 16F1823 et son port d'E/S supplémentaire (*PORTC*). La **figure 2-9** montre comment relier les broches de l'afficheur à celles du PIC. Le câblage prêt, nous pouvons tracer le diagramme d'état du programme (**fig. 2-10**). **M1** est l'automate pilotant l'afficheur. La LED est initialisée à **on** à la mise sous tension, puis l'automate **M0** démarre dans l'état 0. Après 0,25 s le *timer* (temporisateur) expire et **M0** passe à l'état 1. Le *timer* est alors initialisé, la LED mise hors tension, le compteur d'impulsion incrémenté, et un signal appelé *UPDATE* est mis au niveau haut. L'automate **M1** utilise ce signal. Lorsque le *timer* expire au bout de 0,75 s, **M0** passe de l'état 1 à l'état 0, le *timer* est initialisé, et la LED mise sous tension. Ce cycle se poursuit indéfiniment. À l'état 1, une transition vers ce même état se produit lorsque le compteur d'impulsion atteint 10, ce qui a pour effet de le remettre à zéro. Simultanément et indépendamment tourne l'automate **M1**, mais en temps partagé (du processeur). **M1** démarre à l'état 0, segments éteints. Lorsque le signal *UPDATE* est mis au niveau haut, **M1** passe à l'état 2 et remet *UPDATE* au niveau bas. **M1** revient à l'état 1 et commande les segments de l'afficheur en fonction de la valeur de la variable **N**. L'automate reste à l'état 0 dans l'attente d'un nouveau signal *UPDATE*. La logique pour **M1** sera inversée dans le cas d'un afficheur à cathode commune. Pour adapter le programme à un tel afficheur, référez-vous aux commentaires du code **prog_02_03.asm** que, faute de place, nous ne reproduisons pas ici. Vous pouvez le télécharger depuis la page [1] associée à ce livre.

Ce que j'ai décrit du déroulement du programme devrait suffire à sa compréhension, mais j'aimerais attirer votre attention sur les problèmes que peuvent occasionner les instructions **BSF** et **BCF** lors d'une procédure READ MODIFY WRITE modifiant les bits du registre d'un port. Quelque chose d'étrange peut en effet survenir lorsqu'on modifie la sortie d'une broche **juste après** avoir modifié une autre broche. Ce phénomène vient de la nature capacitive des charges liées aux broches d'E/S et de la façon dont un PIC change la valeur d'une broche. Lorsque le PIC écrit dans le registre d'un port pour modifier une sortie, il doit d'abord lire l'état des broches du port car il ne sait qu'écrire un octet complet. Il lit donc ces états, modifie le bit concerné, et réécrit les autres. Supposons qu'une instruction demande de mettre à 1 le bit 5 du

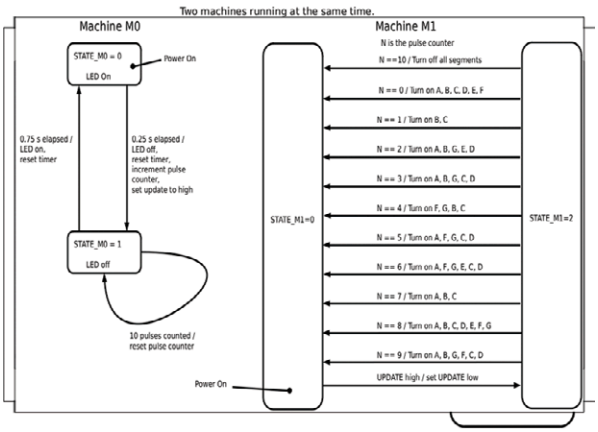

Figure 2-10. Diagramme d'état du programme 02-03.asm (non reproduit ici, cf. [1]).

PORTA (broche physique 2), et que l'instruction suivante demande de modifier la valeur du bit 4 du PORTA (broche physique 3). Le PIC lit les valeurs des broches du *PORTA*, et l'on pourrait s'attendre à ce que la broche 2 soit au niveau haut puisque son bit 5 vient d'être mis à 1. Pourtant rien n'est moins sûr. Le circuit pourrait être capacitif, et dans ce cas il faudra un certain temps avant que la tension atteigne le niveau haut. Si ce temps est trop long, le PIC considérera que la broche 2 est au niveau bas et remettra le bit 5 à 0 lorsqu'il écrira de nouveau le registre du *PORTA*. C'est un exemple de situation dont un logiciel de simulation ne pourra pas deviner l'issue. En général, lorsqu'on utilise *BCF* et *BSF*, il est préférable d'écrire dans les registres de transfert *LATA* et *LATC*. Leur utilisation est toutefois sans risque si l'on écrit le contenu du port entier, pris comme un registre, p. ex. avec *MOVWF PORTA*.

Différences entre les PIC 12F1822 et 16F1823

Vous le savez, le PIC 12F1822 possède 8 broches tandis que le 16F1823 en offre 14 grâce à ses deux ports d'E/S. Ils se programment presque de la même façon. Le contrôleur et le fichier *include* utilisés doivent être déclarés au début du code source de la façon suivante :

```
LIST P=12F1822
#INCLUDE <p12f1822.inc>

LIST P=16F1823
#INCLUDE <p16f1823.inc>
```

L'avantage d'une puce plus petite est qu'elle prend moins de place sur un circuit imprimé et qu'il y a moins de broches à souder si on n'a pas besoin du port supplémentaire. Lisez la fiche technique de votre PIC pour ne pas vous faire piéger par une de ses particularités. Votre code ne fonctionne pas comme prévu ? Là encore la réponse se trouve dans la fiche technique.

Interruptions et diagrammes d'état

Les interruptions n'apparaissent pas sur les diagrammes d'état car elles ne changent pas l'état de l'automate. Nous l'avons vu dans une section précédente, le programme principal est suspendu lorsqu'une interruption se produit. S'il y a changement d'état, ce sera donc uniquement sur reprise, et si l'interruption a modifié l'entrée de l'automate (l'augmentation d'un compteur a p. ex. changé le résultat d'une comparaison).

200447-02

Ont contribué à cet article :

Auteur : **Andrew Pratt**
Rédaction : **Jan Buiting**
Traduction : **Hervé Moreau**
Maquette : **Giel Dols**

Votre avis, s'il vous plaît...
Posez vos questions ou adressez vos commentaires à redaction@elektor.fr

LIEN

[1] **Les programmes du livre, dont les trois décrits ici :** www.elektor.fr/programming-the-finite-state-machine