Programmeren van **de eindige-toestandsmachine**

met 8-bit PIC's in assembler en C

Andrew Pratt (Groot-Brittannië)

In deze aflevering van Elektor Boeken presenteren we drie 'al doende leren' secties en twee 'leuk om te weten' paragrafen uit het boek "Programming the Finite State Machine – with 8-Bit PICs in Assembly and C". Gericht op Microsoft Windows en Linux-gebruikers, is het boek gebaseerd op eenoudige hardware inclusief een seriële FTDI-kabel voor het programmeren en de open source-software 'gpasm' als assembler. Hier laten we de PIC-assembler-versies zien van een LED-flitser, meerdere FSM's in één programma en een 7-segment LED-driver.

Een meer gecompliceerde LED-flitser

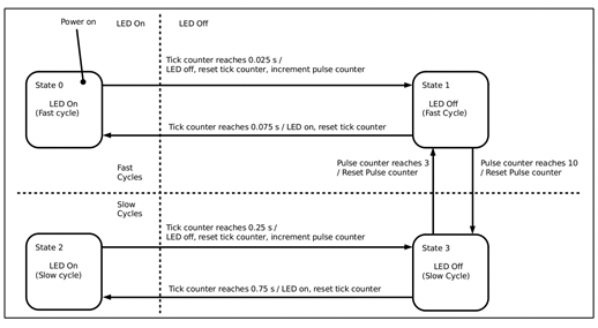
Een van de punten van kritiek op assemblerprogramma's is dat ze moeilijk leesbaar kunnen zijn. Dat kan zelfs het geval zijn als u ze nog maar net hebt geschreven! Als u een standaard-layout voor uw programma gebruikt, wordt het al een stuk gemakkelijker. Ook geeft het eindige-toestandsdiagram een overzicht van de hele operatie. De assemblercode bestaat uit afzonderlijke blokken die de toestanden representeren. U zult zien dat door te verwijzen naar het toestandsdiagram, het volgen van de assemblercode geen problemen oplevert.

Als voorbeeld van een meer gecompliceerd programma geeft **figuur 2-5** het toestandsdiagram voor dezelfde LED; deze knippert nu aan en uit met dezelfde intervallen als hiervoor, maar dan gedurende 3 cycli (heeft betrekking op een programma in een eerder deel van het boek – *redactie*). Daarna knippert de LED sneller gedurende 10 cycli voordat het programma terugkeert naar de tragere snelheid. Merk op dat er vier toestanden zijn met zes overgangen. Er zijn uit- en aan-toestanden voor snel knipperen en uit en aan-toestanden voor langzaam knipperen. Het hele diagram kan worden verdeeld in vier kwadranten met twee scheidslijnen: één tussen snel en langzaam en één tussen aan en uit.

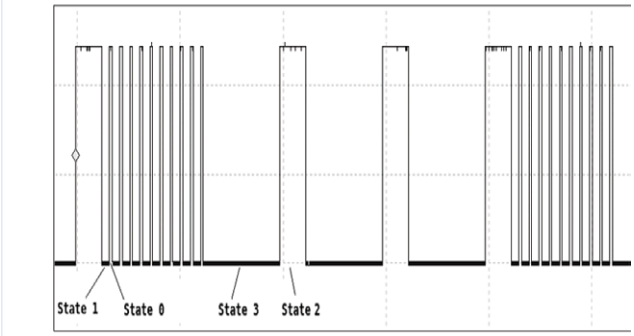
Figuur 2-6 toont de LED-spanning op het oscilloscoopscherm. Het programma is afgedrukt in **listing 1**. Dankzij de verwijzingen naar het toestandsdiagram zou u moeten kunnen begrijpen wat er gebeurt. Alleen het gebruik van het $-teken moet nog worden uitgelegd. Dit staat voor het adres van de huidige regel, dus in plaats van naar een label te springen kunnen over een aantal regels heen springen. Als u GOTO $ + 3 schrijft , springt u bijvoorbeeld 3 regels vooruit.

Meerdere 'machines' in één programma laten draaien

De twee FSM-programma's (*Finite State Machine*) hadden tot dusver een klein codeblok voor elke toestand en voerden die code keer op keer uit totdat een invoer-event een sprong naar een andere toestand veroorzaakte. Ik zal nu een manier voorstellen om meer dan één machine tegelijk in een programma uit te voeren: dit is hetzelfde als twee dingen tegelijk doen door eerst een deel van de ene taak uit te voeren, dan een deel van een andere taak en dan terug naar de eerste – enzovoort. Dit is tijdmultiplexing, de CPU in de microcontroller kan slechts één instructie tegelijk uitvoeren, dus als taken samen moeten worden uitgevoerd, moeten ze de processortijd delen. In tegenstelling tot timers en andere periferie van de PIC, die afzonderlijke hardware-entiteiten zijn die werkelijk gelijktijdig kunnen functioneren, lijkt het alleen maar zo dat meerdere taken in een programma gelijktijdig worden uitgevoerd. Ze worden uitgevoerd als *threads*, waarbij elke thread de processortijd deelt. Bij dit *multi-threading* voert de processor het codeblok



Figuur 2-5. Toestandsdiagram van het LED-knipperprogramma 'Cycles of Cycles'.



Figuur 2-6. Oscillogram voor het programma 'Cycle of Cycles'.

Listing 1. Het 'Cycle of Cycle'-programma (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
```

```
BSF PORTA, 2 ; Inialise LED on;
 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
```

van de eerste machine voor de huidige toestand uit en vervolgens het codeblok voor de huidige toestand van de andere machine, om vervolgens terug te keren naar de eerste machine. Deze afzonderlijke machines zijn onafhankelijk. Ze gebruiken toevallig dezelfde CPU door middel van time-sharing. Ze kunnen ook geheugen delen om toegang te krijgen tot elkaars gegevens.

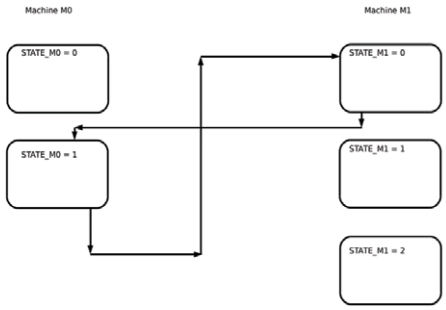
In de eerdere voorbeelden werd de toestand gedefinieerd door de aanwezigheid in een bepaald codeblok, nu wordt de toestand bepaald door de waarde die is opgeslagen in een register. We noemen het register dat de status van machine 0 bevat *STATE_M0* en van machine 1 *STATE_M1*. **Figuur 2-7** toont hoe dit programma zal worden uitgevoerd.

Om dit schakelen tussen toestanden te regelen, kunnen we de instructie *BRW* gebruiken. De *BRW*- instructie is een *branch-with-W*, wat betekent dat de waarde in W bij de programmateller (program counter) wordt opgeteld. Met andere woorden: u kunt een berekende sprong maken.

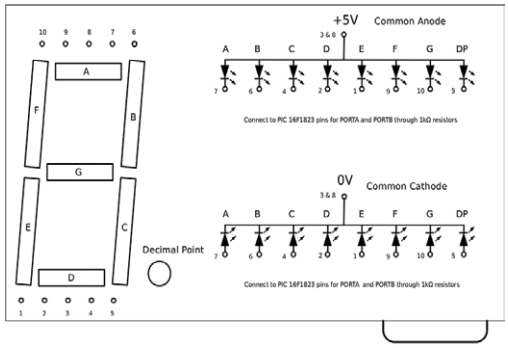
Nadat een blok toestandscode is uitgevoerd, springt het programma terug naar de *switch* voor de volgende machine waar de *BRW*-instructie naar de volgende regel gaat als de toestandswaarde 0 is, een regel overslaat als die 1 is en twee regels overslaat als de waarde 2 is. De uitvoering van het programma is niet opgesloten in de toestandsblokken maar wordt rondgevoerd via de switch-code. Om twee machines als threads te laten draaien, zijn er twee

Listing 2. Fictief programma met twee-toestands-machine 0 en drie-toestands-machine 1.

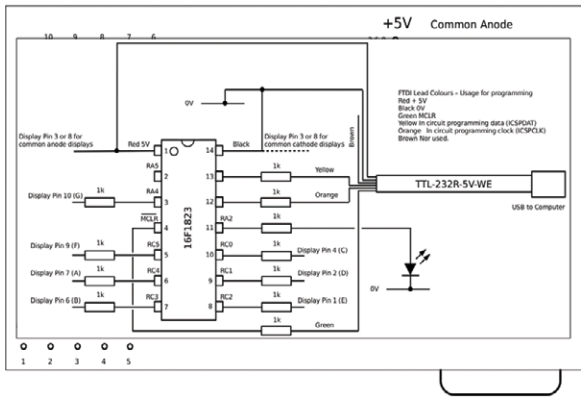
```
SWITCH_M0
     MOVFW STATE_M0      ; Moves the value of STATE_M0 to W.
     BRW                 ; The program will jump from here depending on W.
    GOTO M0_S0           ; The jump will be to this line if W = 0.
     GOTO M0_S1          ; The jump will be to this line if W = 1.

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.
```



Figuur 2-7. Voorbeeld van de programmaflow tussen twee machines in hetzelfde programma.



Afbeelding 2-8. Pinning van typische 7-segment LED-displays.



Figuur 2-9. Aansluiting van het 7-segment-display op de PIC 16F1823.

toestandsvariabelen (*STATE_M0*, *STATE_M1*) en twee switches. **Listing 2** toont een vereenvoudigd fragment uit dit fictieve programma, waarbij machine 0 twee toestanden heeft en machine 1 drie toestanden.

Merk op hoe de programmaflow tussen de machines wisselt bij het verlaten van een blok toestandscode.

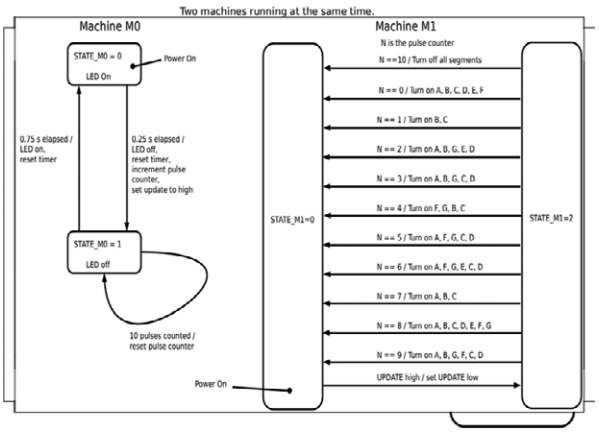
Aansturing van een 7-segment LED-display

Voor een praktische demonstratie van het gebruik van de switches om twee machines in hetzelfde programma aan te sturen, geven we hier als voorbeeld een knipperende LED en een 7-segment-display dat telt hoevaak de LED knippert. Men kan kiezen tussen een LC- of een LED-display. LED-displays zijn eenvoudiger omdat ze kunnen worden aangestuurd met gelijkstroom, terwijl LC-displays wisselspanning nodig hebben. LED's hebben weer het nadeel dat ze meerdere mA per segment nodig hebben. Er zijn ontelbaar veel 7-segment LED-displays verkrijgbaar. Ik gebruik een display met één cijfer. **Figuur 2-8** toont de aansluitingen van een typisch display. Deze zijn verkrijgbaar in twee verschillende uitvoeringen: met gemeenschappelijke kathode en met gemeenschappelijke anode. Bij gemeenschappelijke kathodes worden alle de kathodes aan 0 V gelegd. Bij gemeenschappelijke anode moeten alle anodes op +5 V worden aangesloten. De segmentaansluitingen worden telkens verbonden met de uitgangen van de PIC. De logica van de uitgangscode moet zo worden geschreven dat deze past bij de polariteit van het display. Het is het eenvoudigste om een display met dezelfde aansluitingen te gebruiken; zo niet, dan moet u uw de bedrading zelf uitvogelen. Het LA-601VB common anode-display van RS Components is vergelijkbaar met het display dat ik heb gebruikt. Farnell levert een soortgelijk exemplaar (bestelnummer SA52-11SRWA).

Omdat er in totaal 8 LED's moeten worden aangestuurd, gebruiken we de PIC 16F1823. Deze heeft een extra I/O-poort, PORTC. **Figuur 2-9** toont het 'schema' voor de aansluiting op de PIC.

Het volgende deel van het ontwerpproces is het tekenen van het diagram van de eindige-toestandsmachine (**figuur 2-10**). Machine M1 is de displaydriver. Bij het inschakelen wordt de LED in de aan-toestand geïnitialiseerd, waarna machine M0 in toestand 0 begint. Na 0,25 s loopt de timer af en gaat machine 0 over naar toestand 1. De timer wordt gereset, de LED gaat uit, de pulsteller wordt met 1 verhoogd en het signaal UPDATE wordt hoog gemaakt. Dit UPDATE-signaal wordt gebruikt door machine 1. In toestand 1 gaat de machine, wanneer de timer na 0,75 s afloopt, over naar toestand 0, wordt de timer gereset en gaat de LED aan. Dit gaat eindeloos door. In toestand 1 is er, als de pulsteller 10 heeft bereikt, een overgang naar dezelfde toestand. Dit doen we om de pulsteller op nul te zetten. Intussen draait machine M1 onafhankelijk, maar deelt de CPU-tijd. Machine M1 start in toestand 0 en het display is leeg. Wanneer het UPDATE-signaal hoog wordt, gaat de machine over naar toestand 2 en wordt het UPDATE-signaal naar laag gereset. Afhankelijk van de waarde van de variabele N gaat de machine terug naar toestand 1, waarbij de vereiste segmenten van het display worden in- en uitgeschakeld. De machine blijft in toestand 0 wachten op een ander update-signaal. De uitvoer-details die voor machine M1 worden weergegeven, zijn voor een display met gemeenschappelijke anode. De opmerkingen in de code tonen de alternatieven voor een display met gemeenschappelijke kathode.

Om plaatsredenen kan het programma 'prog_02_03.asm' hier niet worden afgedrukt, maar u kunt het vrijwel direct van de webpagina van het boek ophalen [1]. Het grootste deel ervan is al uitgelegd, behalve één ding waar u op moet letten: het READ MODIFY WRITE-probleem dat kan optreden bij het gebruik van de *BSF*- en *BCF*-instructies. Tot nu toe hebben we slechts één uitgangspin ingesteld door de bitwaarde op *PORTA* in te stellen. Dat gaat goed zolang u slechts met één uitgangspin van een poort werkt, maar er gebeurt iets vreemds wanneer u een uitgang probeert te wijzigen kort nadat u een andere pin heeft gewijzigd. Om het probleem te begrijpen, moeten we kijken naar de elementaire theorie met betrekking tot het laden en ontladen van condensatoren, en bekijken hoe de PIC omgaat met het veranderen van de waarde van één pin tegelijk. Wanneer de PIC naar een uitgang schrijft, kan hij alleen naar een complete byte (dus acht bits) schrijven, dus om één bit te veranderen moet hij de status van de pinnen lezen, degene die hij



Figuur 2-10 Toestandsdiagram voor programma 02-03 (hier niet afgedrukt maar beschikbaar op [1]).

wil wijzigen aanpassen en dan terugschrijven naar de uitgangen. Het probleem is dat bijvoorbeeld *PORTA* bit 5 (fysieke pin 2) is gewijzigd van 0 in 1, en dat u in de volgende instructie de waarde van *PORTA* bit 4 (fysieke pin 3) wilt wijzigen. De PIC leest de waarden op de *PORTA*-pinnen – u zou dan verwachten dat de waarde op bit 5 hoog is zoals u die zojuist hebt ingesteld – NEE dus omdat in de echte wereld de elektrische schakeling capacitief kan zijn en het even duurt voordat de spanning is gestegen. De waarde van bit 5 wordt dan als laag gelezen. Wanneer de uitgangen worden teruggeschreven naar poort, wordt bit 5 (pin 2) weer op laag gezet. Dit is een voorbeeld van een situatie waar een test in een simulator u in de steek kan laten. Over het algemeen moet u bij het gebruik van *BCF* en *BSF* naar de latchregisters voor de poorten (*LATA* en *LATC*) schrijven. Als u echter naar de hele poort als register schrijft, bijvoorbeeld met *MOVWF PORTA*, dan gaat het wel goed.

De verschillen tussen de PIC 12F1822 en de 16F1823

Het in het oog springende verschil is dat de 12F1822 8 pinnen heeft, terwijl de 16F1823 er 14 telt. Dit komt doordat de 16F1823 twee I/O-poorten heeft. Afgezien van dit verschil verloopt het programmeren nagenoeg hetzelfde. U moet het juiste include-bestand bovenaan uw broncode vermelden en de juiste controller voor uw chip vermelden. Dit zijn de relevante regels:

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

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

Het voordeel van een kleinere chip is dat deze minder ruimte inneemt op een print en dat er minder pinnen gesoldeerd moeten worden – als u die extra poort niet nodig hebt. U moet de datasheet raadplegen, want er zijn altijd valstrikken. Er zullen situaties zijn waarin u naar uw code staart en probeert te begrijpen wat er mis is, terwijl het antwoord in de datasheet te vinden is.

Interrupts en toestandsdiagrammen

Ik heb op geen enkele manier geprobeerd interrupts in de toestandsdiagrammen weer te geven, aangezien deze de toestand van de machine niet veranderen. Als er een interrupt optreedt, wordt het hoofdprogramma onderbroken (zoals uitgelegd in een eerder hoofdstuk in het boek). Daarom vindt er geen toestandsverandering plaats tot na de terugkeer uit de interrupt en dan nog uitsluitend als de interrupt de invoer naar de toestandsmachine heeft gewijzigd (zoals een verhoging van een teller die het resultaat van een vergelijking verandert). ⏮.

200447-04

Een bijdrage van

Tekst en beeld: **Andrew Pratt**
Redactie: **Jan Buiting**
Vertaling: **Eric Bogers**
Layout: **Giel Dols**

Vragen of opmerkingen?

Hebt u vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur of naar de redactie van Elektor, via redactie@elektor.com.

WEBLINKS

[1] **Drie .asm-programmabestanden (scroll op de webpagina omlaag naar downloads):**
www.elektor.nl/programming-the-finite-state-machine