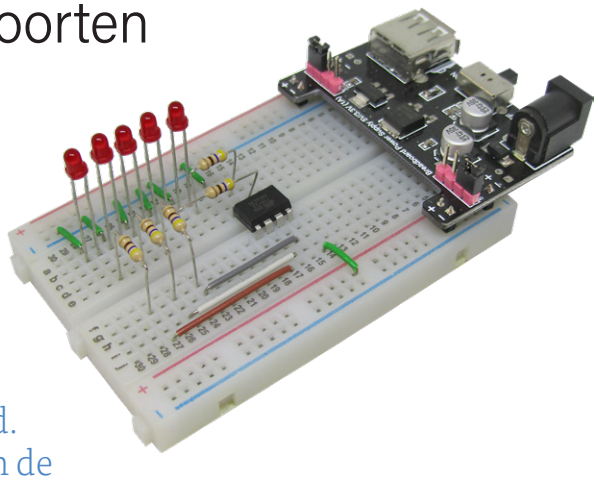
Ontdek ATtiny-microcontrollers met behulp van C en assembler

voorbeeldhoofdstuk: ATtiny I/O-poorten



Warwick A. Smith (Zuid-Afrika)

De I/O-poorten van een microcontroller besturen de aansluitingen ervan en bieden de mogelijkheid om ze individueel te configureren als in- of uitgang. Dit geeft zoveel mogelijkheden dat hier in elke beginnerscursus of inleiding tot het programmeren aandacht aan hoort te worden besteed. Je moet echter werkelijk diep in de materie duiken om de I/O-mogelijkheden van een microcontroller écht te begrijpen en te gebruiken. Hier doet Elektor-auteur Warwick Smith dat aan de hand van een klein assembler-programma op de populaire ATtiny-microcontroller. Nieuwsgierig? Laten we aan de slag gaan!

__Opmerking van de redactie:__ dit artikel is een deel uit het 376 pagina's tellende boek Explore ATtiny Microcontrollers using C and Assembly Language (W. Smith, Elektor 2021) dat we enigszins hebben bewerkt om te voldoen aan de redactionele standaarden en paginaopmaak van Elektor Magazine. Omdat het een gedeelte is van een grotere publicatie, verwijzen sommige aanduidingen in dit artikel naar passages elders in het origineel. Auteur en uitgever hebben hun best gedaan om dat te voorkomen, en zijn graag bereid om eventuele vragen te beantwoorden. Contactgegevens staan in het kader __Vragen of opmerkingen?__.

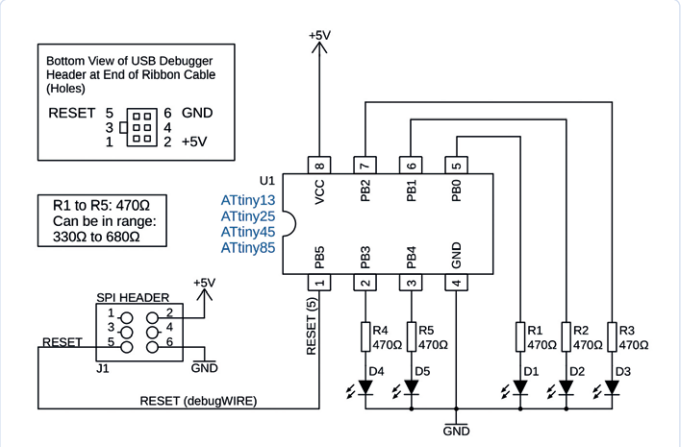
In de ATtiny13(A) en ATtiny25/45/85 worden de I/O-poorten geconfigureerd en aangestuurd met behulp van een set van vier registers. Indien een pin wordt geconfigureerd als ingang, dan kan een programma dat op de AVR wordt uitgevoerd, het logische niveau op die pin lezen. Wanneer er een schakelaar op die pin is aangesloten, dan kan het logische niveau worden gelezen om te zien of die schakelaar open of gesloten is. Wanneer een pin is geconfigureerd als een uitgang, dan kan deze worden gebruikt om het logische niveau ervan 'hoog' (logisch niveau 1) of 'laag' (logisch niveau 0) te maken. Een uitgangspin kan bijvoorbeeld worden gebruikt om een LED aan te sturen, zoals in het knipper-LED-project dat elders in het boek is beschreven.

I/O-pinnen als uitgang configureren in assembler

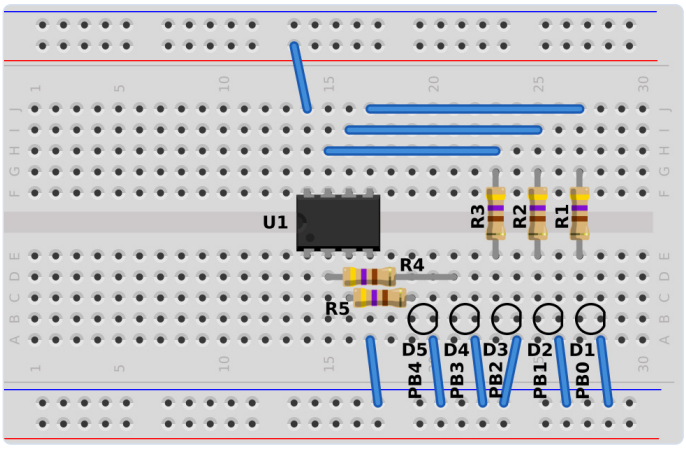
In dit gedeelte bekijken we hoe we bij een 8-pins ATtiny meer dan één pin als uitgang kunnen configureren; we sluiten vijf LED's met serieweerstanden op deze pinnen aan. De code configureert de LED's als een 5-bit binaire teller die vanaf nul telt.

Figuur 1 toont het schema van een ATtiny13(A) of ATtiny25/45/85 AVR met vijf LED's die zijn aangesloten op I/O-pinnen PB0...PB4. Ten behoeve van het programmeren en debuggen van de microcontroller wordt pin PB5 van de ATtiny microcontroller in dit ontwerp gebruikt als debugWIRE-pin. Om deze configuratie te kunnen gebruiken, is een programmer/debugger zoals een Atmel-ICE of AVR Dragon vereist.

Een program-only USB-programmer werkt overigens niet in



Figuur 1. Schema van de 5-LED-teller.



Figuur 2. Breadboard-layout van de 5-LED-teller.

de debugWIRE-modus en kan niet debuggen. **Let op: activeer de DWEN-fuse niet met een program-only USB-programmer want deze kan de AVR niet terug uit de debugWIRE-modus halen.** De reden van het gebruik van de debugWIRE-modus bij deze voorbeeldconfiguratie is om eventueel andere pinnen van de AVR vrij te maken die normaal worden gebruikt in de ISP/SPI-programmeermodus.

Program-only programmers

Voor de configuratie van figuur 1 kunnen program-only USB-programmers worden gebruikt om het voorbeeldprogramma (zie verderop) te laden en de tellerstand op de LED's te bewonderen. Sluit de LED's aan zoals getekend in figuur 1 en **figuur 2**. In het originele USBasp-ontwerp en de originele USBtinyISP-configuratie waren begrenzingsweerstanden in de aansluitingen opgenomen om de programmer en de target-AVR-chip te beschermen. De Arduino Uno, geprogrammeerd als ArduinoISP, heeft geen begrenzingsweerstanden maar deze kunnen worden toegevoegd aan de Arduino Uno MOSI- en SCK-pinnen die naar de target-AVR gaan. In het originele USBasp-ontwerp worden begrenzingsweerstanden van 270 Ω gebruikt die op MOSI, SCK en RESET zijn aangesloten. Bij het originele USBtinyISP-ontwerp worden 1k5-weerstanden gebruikt aan MOSI en RESET.
Begrenzingsweerstanden voorkomen kortsluiting indien een van de AVR-uitgangen een bepaalde uitgangsspanning heeft terwijl op hetzelfde moment de programmer een uitgangsspanning van tegengestelde polariteit voert.

Aangesloten hardware kan het programmeren mogelijk storen

Hoewel de configuratie van figuur 1 kan worden geprogrammeerd met behulp van de ISP/SPI-interface terwijl de LED's plus serieweerstanden zijn aangesloten, kunnen andere ontwerpen wellicht hardware hebben die het programmeren hinderen. Indien periferie die op een te programmeren AVR is aangesloten het programmeren van de AVR inderdaad bemoeilijkt, dan zijn daar wel enkele oplossingen voor. Lezers die een debugWIRE-compatibele USB-programmer/debugger hebben (zoals de Atmel-ICE) kunnen dan natuurlijk de target-AVR eenvoudig in de debugWIRE-modus zetten; dan is slechts één pin voor het programmeren nodig. Een andere oplossing is om de AVR eerst op een ander breadboard te programmeren met behulp van ISP/SPI en pas daarna in de doelschakeling te prikken.
Als alternatief kan ook een AVR met meer pinnen worden gebruikt. Maar de beschikbare poortpinnen zijn niet altijd dezelfde als van een 8-pins PDIP-ATtiny. Als u bijvoorbeeld pinnen PB0…PB4 gebruikt (zoals in het schema van figuur 1), dan zijn er geen vijf opeenvolgende vrije pinnen beschikbaar bij de 14-pins ATtiny24/44/84-serie AVR's, omdat voor ISP/SPI zowel pinnen van poort A als van poort B worden gebruikt. Bij de 20-pins ATtiny26/261/461/861-controllers is poort A compleet beschikbaar voor een ISP/SPI-programmer, maar dat betekent dan dat de software moet worden aangepast om poort A te gebruiken in plaats van poort B. Gelukkig zijn bij deze 20-pins ATtiny2313/4313-serie de pinnen PB0…PB4 wel vrij als een ISP/SPI-programmer is aangesloten.

De AVR in de debugWIRE-modus zetten

Om de microcontroller te kunnen programmeren met behulp van een enkele debugWIRE-lijn, zoals geschetst in figuur 1, moeten eerst alle verbindingen van de ISP-header vanaf de USB-programmer/debugger naar de microcontroller worden gemaakt. Zet vervolgens de DWEN-fuse van de AVR. De AVR is dan in de debugWIRE-modus gezet. Zo niet, sluit dan uw programmer/debugger aan, bijvoorbeeld een Atmel-ICE of een AVR Dragon, en overtuig u ervan dat u de DWEN-fuse kunt zetten. Zodra de DWEN-fuse is geprogrammeerd, kunnen alle aansluitingen van de ISP-header weer worden weer verwijderd, behalve RESET, +5 V (Vcc) en GND, zoals in figuur 1 getekend.

Bouw van de schakeling op breadboard

Nadat u de hardware hebt verzameld, bouwt u de schakeling van figuur 1 op een breadboard op. Zorg ervoor dat de vijf LED's op één rij én in de juiste volgorde op het breadboard verbonden zijn met de pinnen PB0…PB4. LED D1 komt dan aan de rechterkant van de rij en D5 aan de linkerkant. Anders gezegd: de LED's komen naast elkaar, waarbij PB0 naar de meest rechtse LED gaat, PB1 naar de LED links daarnaast, PB2 naar de derde LED van rechts – en zo vervolgens, zoals de breadboard-layout van figuur 2 toont. We zien daar alleen de omtrek van de LED's, zodat de draadverbindingen en weerstanden zichtbaar blijven. Als u het zonder de hardware moet stellen, controleer dan de werking van de programma's met een simulator.

De assembler-code voor de teller met 5 LED's

Start een nieuw Microchip Studio AVR Assembler Project met de naam *led_count_asm*. Neem de code van **listing 1** over in *main.asm* van het project (komt in plaats van het codeskelet dat daar stond). Indien u de hardware van figuur 1 gebruikt, neem dan uw hardware-tool (de debugger) zoals de Atmel-ICE in Microchip Studio, met debugWIRE als interface. Hiertoe klikt u op het hamer-pictogram op de tweede menubalk van boven. Wanneer u de simulator gebruikt, kies dan *Simulator* als tool. Bij gebruik van een 'hobby'-programmer kunt u verder lezen tot gevraagd wordt het programma in de target-ATtiny te laden.
Het *led_count_asm* programma configureert PB0…PB4 als uitgangen zodat de LED's die hierop zijn aangesloten, door het programma kunnen worden in- en uitgeschakeld. Het programma toont een

oplopende binaire waarde (tellerstand) op de LED's, beginnend bij 0 (alle LED's uit). Wanneer de teller zijn hoogste waarde heeft bereikt (alle LED's aan) springt de teller naar nul en begint de telling opnieuw. Elke uitgeschakelde LED staat voor een binaire (logische) 0, en elke ingeschakelde LED is een binaire (logische) 1. Het is belangrijk de LED's op te stellen zoals in figuur 2: alleen dan wordt de tellerstand correct weergegeven (PB0/D1 als LSB en PB4/D5 als MSB).

Maak het programma en laad het in de AVR wanneer u de fysieke hardware van figuur 1 en figuur 2 gebruikt. Indien u voor een Atmel-ICE of AVR Dragon hebt gekozen, gebruikt u het pictogram *Start Without Debugging* op de bovenste toolbar van Microchip Studio (sneltoets Ctrl+Alt+F5) om zo het programma in de AVR te laden. Als de debugger-interface correct is ingesteld en de chip zich in de debugWIRE-modus bevindt dan wordt het programma geladen en kan het worden gestart. De oplopende binaire tellerstand is nu te zien op de LED's. Als u een program-only programmer ('hobby'-programmer) gebruikt, laad dan het programma naar de target-AVR met de juiste functie. Als het programma correct is ingetypt, de schakeling juist is bedraad en het programma is opgeslagen en opgestart na het invoeren, ziet u ook de binaire weergave op de LED's oplopen. Desgewenst kunt u voor de rest van de tekst nu de simulator gebruiken.

Als u niet over de fysieke hardware beschikt, dan kunt u het telproces bekijken in de Simulator in Microchip Studio. Het programma kan dan worden doorlopen met behulp van de pictogrammen *Start Debugging* en *Break*. Nadat de simulator is opgestart , opent u het I/O-venster door op *Debug Windows I/O* te klikken (bovenste menu) gevolgd door een klik op *I/O Port (PORTB)*. Speel het programma af via het *Step Over*-pictogram (toets *F10*). Op het PORTB-item onderin het I/O-venster ziet u de telwaarde die op de LED's zou worden weergegeven indien die waren aangesloten. De telling wordt telkens in PORTB bijgewerkt wanneer in de main-loop de `out`-instructie wordt aangeroepen.

De werking van het tellerprogramma

De helft van de *led_count_asm* programmacode bestaat uit een delay-subroutine, die ook werd gebruikt in het knipper-LED-programma dat elders in het boek wordt besproken. Deze subroutine wordt eenmalig in de main-programmalus aangeroepen om ervoor te zorgen dat de tellerstand op de LED's duidelijk zichtbaar is en niet te snel voorbijflitst. Open het *led_count_asm* project in Microchip Studio, met de *main.asm*-code ook geopend, zodat u de nuvolgende bespreking goed kunt volgen.

De eerste twee instructies van het programma worden gebruikt om de pinnen PB0...PB4 in te stellen als uitgang, om er dan de LED's mee aan te sturen; hiertoe moeten de juiste bits in het DDRB-register worden gezet. **Figuur 3** toont bovenin het DDRB-register. Elk bit in dit register komt overeen met een pin van de microcontroller. Bit DDB0 correspondeert bijvoorbeeld met pin PB0, DDB1 met pin PB1 enzovoort.

Wanneer een bit in DDRB logisch 1 wordt gemaakt, wordt de corresponderende pin een uitgang. Indien een bit in DDRB logisch 0 wordt gemaakt, verandert de corresponderende pin in een ingang; dat is overigens de standaard toestand van alle pinnen bij het opstarten of na een reset. Aan het begin van het programma wordt eerst `0b0001_1111` naar register R16 geschreven, en vervolgens van R16 naar het DDRB-register met behulp van de `out`-instructie. Het is noodzakelijk om eerst deze constante waarde in R16 te laden, want er is geen instructie beschikbaar om direct een constante waarde naar een I/O-register zoals DDRB te schrijven. Door `0b0001_1111` naar DDRB te schrijven worden de bits DDB0..DDB4 gezet en worden pinnen PB0...PB4 uitgangen.

Hoewel er vier registers beschikbaar zijn voor het besturen van I/O-poort B, zijn er slechts drie getekend in figuur 3. Het vierde register, MCUCR, heeft slechts één bit dat van toepassing is op poort B. Dit bit is een globaal pull-up disable bit dat we in dit hoofdstuk

Listing. 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

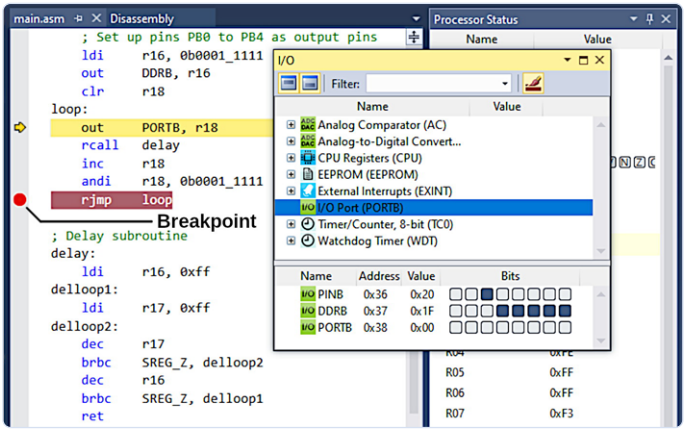
PORTB : Port B Data Register. Drives output pins. Enables input pin pull-ups with logic 1.
Address: 0x18 **Initial Value:** 0b0000_0000 **Bits 5 to 0:** Read/Write (R/W)

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

Figuur 3. I/O-poortregisters van de ATtiny13(A) en de ATtiny25/45/85.



Figuur 4. Een breakpoint invoegen in Microchip Studio Debugger.

niet gebruiken. Raadpleeg het Register Description gedeelte van de datasheet in het deel over de I/O-poorten om over dit register te lezen (ofwel de datasheet van de ATtiny13/ATtiny13A ofwel die van de ATtiny25/45/85).

R18 wordt in het programma gebruikt om de oplopende tellerstand vast te houden die via de LED's wordt weergegeven. R18 wordt met CLR gewist (0 gemaakt) vóór de hoofd-programmalus zodat de telling begint vanaf 0.

In de main-programmalus wordt de inhoud van R18 naar het PORTB-register gestuurd met behulp van de `out`-instructie. Het PORTB-register is te zien in het midden van figuur 3. Nogmaals, elk bit in dit register komt overeen met een pin van de microcontroller. Voor pinnen die met DDRB zijn ingesteld als uitgang ziet u de logische niveaus op deze pinnen zoals deze naar het PORTB-register zijn geschreven. Bij pinnen die zijn ingesteld als ingang activeert een logische 1 in PORTB een interne pull-up weerstand op de betreffende pin. Een logische 0 schakelt de pull-up weerstand van de corresponderende pen uit.

Omdat pinnen PB0...PB4 zijn ingesteld als uitgang verschijnt de tellerstand die naar PORTB is geschreven hier als logische niveaus die de LED's in- en uitschakelen. Een logische 1 schakelt de corresponderende LED in en een logische 0 schakelt de LED uit.

Nadat de tellerstand naar PORTB is geschreven met behulp van de `out`-instructie, wordt de `delay`-subroutine aangeroepen om die telwaarde nog een tijdje zichtbaar te houden. Register R18 wordt vervolgens verhoogd met 1 met behulp van de `inc`-instructie zodat de teller één verder telt. `andi` wordt gebruikt om de hoogste drie bits van de waarde in R18 te wissen met behulp van de waarde 0b0001_1111 als masker. Dit zorgt ervoor dat deze hoogste bits altijd nul zijn. Terug aan het begin van de lus, wanneer R18 weer naar PORTB wordt geschreven, wordt altijd een 0 naar de bovenste drie bits geschreven omdat ze immers zijn gewist met `andi`. Zodra aan het einde van de lus de `rjmp`-instructie is bereikt gaat het programma weer naar het begin van de programmalus, waarbij de nieuwe tellerstand naar PORTB wordt geschreven en wordt weergegeven via de LED's.

Een paar opmerkingen over dit programma: de I/O-registers DDRB en PORTB worden beide gebruikt om poort B van de microcontroller in beide programma's in te stellen en aan te sturen. Het eerdergenoemde knipper-LED-programma hoefde slechts één enkele pin aan te sturen; daarvoor werden de `sbi`- en `cbi`-instructies gebruikt om één enkel bit in de I/O-registers direct te zetten en te resetten. Daarvoor was geen van de werkregisters R0...R31 nodig. Ons tellerprogramma moet echter gelijktijdig toegang hebben tot vijf LED's (pinnen); daarom wordt hier de `out`-instructie gebruikt om tegelijkertijd naar meerdere bits in een register te schrijven.

Het is zaak om het registergebruik in een assembler-programma goed bij te houden. Aan het begin van het programma wordt bijvoorbeeld een immediate waarde geladen in R16. R16 wordt in dit geval echter slechts tijdelijk gebruikt, dus kan later in het programma opnieuw worden gebruikt zonder dat eerst de waarde ervan moet worden opgeslagen. Het wordt dan ook opnieuw gebruikt in de delay-subroutine. En omdat R16 en R17 worden gebruikt in de delay-subroutine is R18 gekozen om de tellerstand vast te houden en wordt het verder voor niets anders gebruikt. Dit zorgt ervoor dat de telwaarde nooit wordt overschreven. Mocht een programma erg groot worden en er geen registers over zijn voor speciale doeleinden, dan kunnen `push`- en `pop`-instructies aan het begin en einde van een subroutine worden gebruikt om de waarden in de gebruikte registers te onthouden.

Het gebruik van breakpoints in de debugger

Met de simulator, of met de hardware en een dito debugger zoals de Atmel-ICE, kunt stap voor stap door het programma lopen door op *Start Debugging* en *Break* te klikken, zoals we al eerder hebben gedaan. Bekijk de I/O-poortregisters door het I/O-venster in Microchip Studio te openen. Terwijl de debugger loopt selecteert u daartoe *Debug Window I/O* en vervolgens op *I/O Port (PORTB)* in het I/O-venster, zoals getoond in **figuur 4**.

Het is handig om op het moment dat de uitvoering van het programma de hoofd-programmalus bereikt een breakpoint op de `rjmp`-instructie te zetten, en dan de hele lus uit te voeren door steeds op de knop *Continue* (of *F5*) te klikken. Zo'n breakpoint kan aan de `rjmp`-instructie worden gekoppeld door te klikken op het grijze vlak helemaal links van de instructie en van het Microchip Studio-venster. Er verschijnt dan een rode stip als indicatie dat er een breakpoint op de instructie is ingesteld, zoals te zien in figuur 4. U kunt ook op de instructie klikken waaraan u een breakpoint wilt koppelen zodat de cursor erop wordt geplaatst, en dan *Debug Toggle Breakpoint* selecteren in het bovenste menu (of druk op *F9*). Zodra het breakpoint is ingesteld kunt u met de knop *Continue* (of *F5*) door de hele lus stappen en aan het eind *Break* gebruiken. Zo ziet u de tellerstand steeds met 1 toenemen in PORTB in het I/O-venster, alsmede in register R18 in het *Processor Status*-venster, zonder dat elke instructie in de lus afzonderlijk moet worden doorlopen.

Verwijder het breakpoint weer van de `rjmp`-instructie door op de rode stip links van deze instructie te klikken, of gebruik het menu of *F9* om het breakpoint om te schakelen, ervan uitgaande dat de cursor zich op de `rjmp`-instructie bevindt. Zet nu een breakpoint op de `inc r18`-instructie. Gebruik opnieuw *F5* om de lus te doorlopen. Het nieuwe breakpoint zorgt ervoor dat het PORTB-register en R18 dezelfde waarde behouden wanneer de uitvoering van het programma stopt. Als de uitvoering van het programma stopt op `rjmp` dan wordt R18 verhoogd voordat de nieuwe waarde ervan naar PORTB wordt geschreven, zodat deze waarden niet gesynchroniseerd zijn in de debugger-vensters in Microchip Studio.

De software van de auteur kan als ondersteuning van het boek gratis worden gedownload. Ga naar [1], scroll omlaag naar *Downloads* en klik op de bestandsnaam *Software_Explore ATtiny Microcontrollers using C and Assembly Language*. Sla het zip-bestand lokaal op (ca. 29 kB) en pak het vervolgens uit.

220045-03

Een bijdrage van

Tekst en afbeeldingen: **Warwick Smith**
Redactie: **Jan Buiting**
Vertaling: **Marc Gauw**
Layout: **Giel Dols**

Vragen of opmerkingen?

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

WEBLINK

[1] Book resources/info page: https://www.elektor.com/20007

GERELATEERDE PRODUCTEN

- Boek: W. Smith: Explore ATtiny Microcontrollers using C and Assembly Language (SKU 20007)
  www.elektor.nl/20007
- E-Boek: W. Smith, Explore ATtiny Microcontrollers using C and Assembly Language (SKU 20008)
  www.elektor.nl/20008