HANDLEIDING

AVR-ASSEMBLER PROGRAMMEREN

VERSIE 3.4 α





Ben Kuiper en Jesse op den Brouw De Haagse Hogeschool Opleiding Elektrotechniek 25 september 2018 B.Kuiper@hhs.nl J.E.J.opdenBrouw@hhs.nl

Inhoudsopgave

1	Inleiding	2
2	De ontwikkeling van software voor de AVR2.1AVR development board2.2Software2.3Het assemblertraject	2 2 3 3
3	Tutorial AVR assembler	4
	3.1 Opbouw assemblerprogramma's	4
	3.2 Het programma <i>ledcopy.asm</i>	4
	3.3 Invoeren van het programma ledcopy	5
	3.4 Starten van de debug-omgeving van Atmel Studio met de JTAG-ICE mkII	9
	3.5 Runnen van <i>ledcopy.asm</i>	11
	3.6 Het programma one_hz.asm	11
	3.7 Simulatie van <i>one_hz.asm</i>	12
	3.8 Breakpoints	14
	3.9 Watch	14
	3.10 Werking one_hz.asm	15
	3.11 Uitvoeren van <i>one_hz.asm</i> op het AVR development board	17
4	Atmel Studio sneltoetsen	17
Bil	oliografie	17

We bedanken oud-collega Harry Broeders voor het bijhouden van eerdere versies van dit document.

1 Inleiding

Deze handleiding is gebaseerd op *Introductie AVR-microcontroller* van Burghouwt © 2006 [2]. Deze handleiding is geschreven om via een kort leertraject software te ontwikkelen voor de AVR 8-bit microcontroller van Atmel. Daarbij wordt ervan uitgegaan dat deze handleiding wordt gebruikt in combinatie met het boek: *AVR Microcontroller and Embedded Systems: Using Assembly and C* [3].

2 De ontwikkeling van software voor de AVR

De ontwikkelomgeving bestaat uit de volgende delen:

- AVR development board, bestaande uit een STK500-experimenteerbord met ATmega32A, zelfontwikkeld I/O-board inclusief LCD-display en JTAG-ICE mkII Programmer / In Circuit Emulator (ICE);
- Atmel Studio, AVR-assembler, simulator en programmer van Atmel.

2.1 AVR development board

Tijdens het practicum wordt gebruik gemaakt van een hardwareomgeving die door de opleiding Elektrotechniek zelf is ontwikkeld. Het bestaat uit een STK500 met ATmega32A, een zelfontwikkeld I/O-board en een JTAG-ICE mkII Programmer / In Circuit Emulator (ICE). Zie figuur 1.



Figuur 1: AVR development board.

Bij het practicum INLMIC maken we gebruik van de leds, de schakelaars en de LCD-display. De LCD kan gebruikt worden om tekst weer te geven. Het bestaat uit twee regels van zestien tekens per regel. Er is software beschikbaar voor het aansturen van de LCD. De leds zitten aangesloten op poort B, de drukknoppen op poort A.

Omdat er meer I/O-lijnen nodig zijn dan de ATmega32A aan boord heeft, zijn diverse componenten gemultiplext. Hiervoor is een schakelaar aangebracht. Indien deze in de stand "Display" staat, is poort A aangesloten op de LCD en zijn de drukknoppen niet beschikbaar. Indien de stand "Switches" is gekozen, zijn de drukknoppen aangesloten op poort A en is de LCD niet beschikbaar.

De potmeter levert een analoge spanning tussen 0 V en 5 V op pin 0 van poort A. Deze pin zit ook (via een weerstand) aangesloten op drukknop 0. Let er op dat als je drukknop 0 wilt gebruiken de potmeter in de middenstand moet staan!

De JTAG-ICE mkII is via een USB-aansluiting verbonden met de PC. Het programmeren en debug-

gen van de ATmega32A gaat via deze aansluiting.

2.2 Software

De software is gratis te downloaden en te gebruiken, ook voor commerciële doeleinden. We werken op de Haagse Hogeschool met Atmel Studio 6.2. We adviseren je om thuis ook versie 6.2 te installeren. De installatiebestanden kun je vinden op de Blackboard course van INLMIC onder *Documents > Links en downloads*.

Op de practicumcomputers is de installatie reeds uitgevoerd. Het programmer-deel van Atmel studio is alleen te gebruiken op computers die zijn aangesloten op het AVR development board.

2.3 Het assemblertraject

Assembly is een taal waarmee direct geprogrammeerd wordt in de AVR-machine-instructies. Assembly is vaak moeilijk leesbaar en verschilt per processorfamilie (bijvoorbeeld een Intel Core i7 processor heeft een totaal andere instructieset dan de AVR). Daarom proberen veel programmeurs assembly te vermijden en programmeert men in een hogere programmeertaal (zoals C). Een compiler verzorgt dan de vertaling naar machine-instructies.

Soms is assembly onvermijdelijk, al is het maar om de compiler platformspecifiek te krijgen. In figuur 2 is schematisch het traject aan van brontaal (source) naar doeltaal (target) van de AVR te zien.



Figuur 2: Schematisch overzicht assemblertraject.

Via de editor wordt de broncode (source) ingevoerd. Dit is gewoon leesbare tekst. Het bestand heeft de extensie *.asm*. De assembler zal vervolgens de broncode omzetten naar het doelbestand (target). Dit is de machinetaal voor de AVR. De target heeft de extensie *.hex*. Via de programmer kan het *.hex*-bestand geladen worden in het flash-geheugen van de AVR-microcontroller. Hiervoor bevindt de microcontroller zich bijvoorbeeld op het STK500-experimenteerbord dat communiceert met Atmel Studio via de JTAG-ICE.

Voor het assembleren van het geschreven programma zijn vaak extra bestanden nodig. Dergelijke bestanden bestaan, net als de source, uit leesbare tekst en hebben de extensie *.inc*. Een voorbeeld van zo'n "include-bestand" is *m32def.inc*, die symbolische namen definieert voor de registers van de ATmega32A. Vlak voor het assembleren wordt een *.inc*-bestand automatisch op de gewenste plaats in de source geplakt. Hierdoor wordt de programmeur de moeite bespaard om zelf steeds allerlei registers te benoemen met symbolische namen.

Om de software te kunnen "debuggen" wordt door de assembler een extra bestand gegenereerd met de extensie *.obj.* De simulator kan hiermee de uitvoer van het programma simuleren. Tijdens het simuleren van de instructies kunnen gelijktijdig de I/O, de processor, geheugen, en registers van de "virtuele" AVR-microcontroller worden weergegeven.

Atmel Studio genereert nog een aantal (minder belangrijke) bestanden. Deze zijn in de voorgaande tekst niet genoemd:

- *.eep* bestand met code voor de EEPROM¹ van de microcontroller (wordt gelezen door de programmer);
- .lss bestand met een listing van de geassembleerde code (voor de programmeur);
- *.map* bestand met koppeling symbolische namen en waardes;
- .atsln bestand die de projectinstellingen bevat.

3 Tutorial AVR assembler

In dit hoofdstuk worden twee eenvoudig programma's uitgewerkt die geschreven zijn in assemblertaal. Het eerste programma leest poort A (drukknoppen) en kopieert de waarden naar poort B (leds). Hiervoor wordt poort A geprogrammeerd als ingang en poort B als uitgang. Het tweede programma zorgt voor een 1 Hz puls op poort B (leds). Hiervoor wordt een wachtlus gebruikt die getimed wordt op de klokfrequentie van het AVR development board.

3.1 Opbouw assemblerprogramma's

Een assemblerprogramma bestaat uit vier velden per regel gescheiden door één of meerdere tabs met de volgende structuur:

label: opcode(mnemonics) operanden(adressen, data) ;commentaar

- Het *label*-veld wordt meestal gebruikt om een bepaalde plaats in het programmageheugen van een label te voorzien. Hierdoor kan vanuit andere plekken in het programma naar deze plek worden gesprongen. De assembler rekent tijdens assembleren de feitelijke sprong uit. Let erop dat het *label*-veld wordt afgesloten met een dubbele punt.
- In het *opcode*-veld staat het eigenlijke commando. Zo'n commando bestaat uit een kort woordje dat meestal een afkorting is die bedoeld is om eenvoudig te kunnen onthouden en snel kan worden ingetypt. De opcode LDI is bijvoorbeeld een afkorting van LoaD Immediate. Een opcode wordt ook wel "mnemonic"² genoemd.
- In het *operanden*-veld staan één of meer operanden. Een operand is een waarde of een adres (geheugen, register of I/O) of een indirecte verwijzing naar een geheugenadres. Meestal wordt gewerkt met symbolische namen en labels die door de assembler tijdens assembleren worden omgezet naar werkelijke waarden. Bovenaan het programma of in een zo genoemd include-bestand worden de symbolische namen gekoppeld aan de werkelijke waarden. Een aantal standaard symbolen is reeds gedefinieerd in het include-bestand *m32def.inc*.
- Het *commentaar*-veld start met een puntkomma. Het commentaar dient als uitleg van het programma. De assembler neemt dit commentaar niet mee tijdens het assembleren.

3.2 Het programma *ledcopy.asm*

Het onderstaande programma *ledcopy.asm* kopieert de data van poort A naar poort B. Doordat de drukschakelaars zijn aangesloten op poort A en de leds zijn aangesloten op poort B zorgt het programma ervoor dat de toestand van de drukschakelaars zichtbaar gemaakt (gekopieerd) wordt op de leds. Het grootste stuk van de code dient ter initialisatie van de I/O-poorten, poort A wordt geconfigureerd als input en B als output. Tenslotte wordt in een lusje steeds eerst de waarde van poort A gelezen in register R0 en daarna wordt de waarde van R0 naar poort B geschreven.

¹ EEPROM staat voor Electrically Erasable Programmable Read-Only Memory. Dit type geheugen zullen we bij het practicum INLMIC nog niet gebruiken. Zie eventueel [3, blz. 19].

² Het Engelse woord mnemonic betekent geheugensteuntje.

```
ledcopy.asm - Simply copies switches to LEDs
;
    Author:
               <your name>
;
   Date:
                <date>
;
   AVR type:
               ATmega32
;
   Target:
               STK 500 (AVR development board)
;
        LDI
                R16,0x00
        OUT
                DDRA.R16
                                ; configure Port A as 8 inputs
        I D T
                R16,0xFF
                DDRB,R16
        OUT
                                ; configure Port B as 8 outputs
                                ; read Port A (switches)
L00P:
        IN
                RØ, PINA
                PORTB,R0
        OUT
                                ; output to Port B (leds)
        RJMP
                LOOP
                                ; and again
```

Listing 1: ledcopy.asm

De code in listing 1 is eenvoudig te lezen, maar lastig te begrijpen. Wat is de betekenis van LDI, R16, 0x00 enzovoort. Om een en ander duidelijker te maken heeft de programmeur commentaar achter bepaalde regels geplaatst.

In het boek [3] kun je informatie vinden die je zal helpen om de werking van dit programma te begrijpen.

- In [3, blz. 56–57] worden de general purpose registers van de AVR besproken, R16 is één van deze registers. In dezelfde paragraaf wordt ook de LDI instructie behandeld. *Lees deze twee pagina's uit het boek voordat je hier verder leest!* Na het lezen van deze paragraaf zal je begrijpen waarom de programmeur R16 en niet het meer voor de hand liggende register R0 heeft gebruikt.
- In [3, blz. 140–144] wordt het gebruik van de I/O-registers DDRA, DDRB, PINA en PORTB besproken. Het is op dit moment voldoende als je begrijpt dat het DDRA register met allemaal nullen moet worden geladen zodat de drukknoppen via het PINA register kunnen worden ingelezen en dat het DDRB register met allemaal enen moet worden geladen zodat de leds via het PORTA register kunnen worden aangestuurd.
- De RJMP instructie zorgt ervoor dat teruggesprongen wordt naar de instructie die voorzien is van het label LOOP. Hierdoor wordt de kopieeractie steeds herhaald. Zie eventueel [3, blz. 117].

3.3 Invoeren van het programma ledcopy

We gaan nu het programma invoeren. Start de Atmel Studio door op het pictogram te klikken, zie figuur 3.



Figuur 3: Het pictogram van Atmel Studio.

We gaan een nieuw assemblerproject aan maken met behulp van de wizard. Na het starten verschijnt het volgende scherm (figuur 4):



Figuur 4: Het beginscherm van Atmel Studio.

Je krijgt nu een scherm te zien waar je een nieuw project aan kan maken. Als je al projecten hebt gemaakt, kan je die selecteren onder *Recent Projects* of via de *Open* knop. Klik op *New Project*. Er wordt een nieuw scherm geopend (figuur 5):

New Project		? 🔀
Recent Templates	Sort by: Default	Search Installed Templates
Installed Templates C/C++ Assembler Atmel Studio Solutio	n	Type: Assembler Creates an AVR 8-bit Assembler project
Name:	AssemblerApplication1	
Location:	H:\Mijn Documenten\Atmel Studio\6.2	Browse
solution name:	азепшенаррисацон	OK Cancel

Figuur 5: Het scherm waarin nieuwe projecten kunnen worden aangemaakt.

Kies nu onder Installed templates voor Assembler. Doe nu het volgende:

- vul in *Name* de projectnaam in (zonder extensies);
- vink Create directory for solution aan;
- zet *Location* op de juiste map; neem hiervoor een map op de H-schijf.

Klik nu op *OK*. Je krijgt nu een nieuw scherm te zien waarin je het AVR-type kan kiezen. Kies *ATmega32A*, zie figuur 6.

Device Family:	All 🗸					Search for device	۶
Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (b)		Device Info:		
ATmega325P	32	2048	1024		Device Name:	ATmega32A	
ATmega325PA	32	2048	1024		Sneed	0	
ATmega328	32	2048	1024		Speed.	27/55	
ATmega328P	32	2048	1024		VCC:	2,7/3,5	
ATmega329	32	2048	1024		Family:	megaAVR	
ATmega3290	32	2048	1024		Datashee	ts	
ATmega3290A	32	2048	1024				
ATmega3290P	32	2048	1024		Supported To	ols	
ATmega3290PA	32	2048	1024		Atmel-ICE		
ATmega329A	32	2048	1024		AVR Drag	on	
ATmega329P	32	2048	1024		• mining		
ATmega329PA	32	2048	1024		P AVRISP m	<u>ikll</u>	
ATmega32A	32	2048	1024		AVR ONE	!	
ATmega32C1	32	2048	1024				
ATmega32HVB	32	2048	1024				
ATmega32HVBrevB	32	2048	1024		JTAGICE	mkll	
ATmega32M1	32	2048	1024		The Simulator		
ATmega32U2	32	1024	1024	_	STK500		
T 20114		25.00	1001		-		
•			,	_	*** <u>STK600</u>		

Figuur 6: Het scherm waarin het AVR-type geselecteerd moet worden.

Klik nu op *OK* om de wizard te beëindigen. Er wordt nu een map aangemaakt onder de opgegeven locatie; daarin wordt nu automatisch het initiële bestand (in dit voorbeeld *ledcopy.asm*) aangemaakt. In figuur 7 zie je het nu geopende scherm. Rechtsboven is het projectscherm waarin je alle bestanden kan vinden, ook bestanden die door de assembler worden aangemaakt of die via een *.include* binnenhaalt. Links is het invoerscherm waarin je de broncode invoert en onder is het venster waarin je de uitvoer van de assembler ziet³.

Vul nu de code in uit listing 1. In figuur 7 is niet alle code zichtbaar. In listing 1 zijn alle instructies in hoofdletters weergegeven (zoals in het boek). Als je een programma intypt, is het vervelend om steeds hoofdletters te gebruiken. Om deze reden zijn de instructies in figuur 7 in kleine letters weergegeven (zoals in de PowerPoint presentaties die bij de theorielessen worden gebruikt).



Figuur 7: Atmel Studio na het invoeren van *ledcopy.asm*.

Druk nu op $F7^4$ of kies via het menu *Build* > *Build Solution*. Als alles goed gaat, verschijnt onderaan de melding *Build*: 1 succeeded or up-to-date, 0 failed, 0 skipped, zie figuur 8. Zo niet, verbeter dan eventuele fouten en herhaal het assembleren.

³ De schermindeling van Atmel Studio kan helemaal naar eigen smaak worden ingericht. Het kan zijn dat de indeling van de op de PC geïnstalleerde software afwijkt van datgene wat hier afgebeeld is.

⁴ Hoofdstuk 4 bevat een lijst met veelgebruikte sneltoetscombinaties.

Output	
Show output from: Build 🔹 🖓 🧔	🔍 🔍 🖻
AVRASM: AVR macro assembler 2.1.57 (build 16 Aug 27 2014 16 Copyright (C) 1995-2014 ATMEL Corporation [builtin](2): Including file 'C:\Program Files (x86)\Atmel\ "ATmega32A" memory use summary [bytes]: Segment Begin End Code Data Used Size Us	:39:43) Atmel Toolchain\AVR Assembler\Native\2.1.1175\avrassembler\Include\m32Adef.inc′ :e%
[.cseg] 0x000000 0x00000e 14 0 14 32768 0. [.dseg] 0x000060 0x000006 0 0 0 2048 0. [.eseg] 0x000060 0x000000 0 0 0 1024 0. Assembly complete, 0 errors. 0 warnings Done executing task "RunAssemblerTask". Done building target "CoreBuild" in project "LedCopy.asmproj". Target "PostBuildEvent" skipped, due to false condition; ('\$(PostBu Target "Build" in file "C:\Program Files (x86)\Atmel\Atmel Studio 6 Done building target "Build" in project "LedCopy.asmproj". Done building target "Build" in project "LedCopy.asmproj".	 0% 0% 0% nildEvent)' != '') was evaluated as ('' != ''). .2\Vs\Avr.common.targets" from project "H:\Mijn Documenten\Atmel Studio\6.2\Demol\Demo
Build succeeded. ======== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==	
III Error List III Output	

Figuur 8: Atmel Studio assembler output na het succesvol assembleren van ledcopy.asm.

Er is nu een aantal bestanden door de assembler aangemaakt, zoals beschreven in paragraaf 2.3. De meest relevante zijn *ledcopy.hex* en *ledcopy.obj* en zijn te vinden in de map *Output Files* in de *Solution Explorer*. Het *.hex*-bestand bevat de eigenlijke target code (machinecode in Intel-hex formaat) en kan geladen worden in de microcontroller. Het *.obj*-bestand kan worden gebruikt voor het debuggen van de software door middel van de Atmel Studio-simulator.

3.4 Starten van de debug-omgeving van Atmel Studio met de JTAG-ICE mkll

Het foutloos assembleren van het bestand wil nog niet zeggen dat de software correct werkt. Om na te gaan of het programma doet wat het zou moeten doen, kan de werking op het AVR development board (zie paragraaf 2.1) worden gevolgd. Dit wordt het tracen van een programma genoemd. Het opsporen van fouten wordt debuggen genoemd.

Kies het menu *Debug* > *Continue* of klik op de play button; die eruit ziet als de play-knop op een mediaspeler. Atmel Studio zal nu een verbinding proberen op te bouwen met een programmer, maar deze is nog niet ingesteld. Hierdoor krijg je een foutmelding te zien. Vervolgens krijg je de mogelijkheid om deze verbinding in te stellen. Selecteer bij *Selected debugger/programmer* JTA-GICE mkII en bij *interface JTAG*. De rest van de instellingen kan ongemoeid worden gelaten. Druk nu nogmaals op de play button. Als het goed is, zal je programma nu gaan draaien op de micro-controller. Je kunt het programma onderbreken door op de Pause button te klikken (Break All, Ctrl+F5). De schermindeling verandert dan enigszins. Zo wordt aan de rechterkant het *Processor*-scherm geopend en is een gele pijl zichtbaar. Deze pijl geeft aan welke instructie uitgevoerd gaat worden, zie figuur 9.

_													
-	LedCopy	(Debugging)	- AtmelStudio										×
Fi	le Edit	View VAss	istX ASF Project	t Build Debug Tools V	/indow	Help							_
1	1 - 4	🕮 * 🜈 🔓	🖉 🕺 🗅 🖻	9 - (2 - 2 - 3 2	20	DII	Debug	- 🌁			- 🛛 🖓 🔗	🖹 🍋 🔜 🔄	;:∵
-	🔁 🗁 🕉	* 8 8	O≥ abc 🛱 📮 🗄	🔊 🖡 🖬 🗢 🖬 🖌	, ™ ⊒ (,⊒	Č_	*≣ 🚹 Hex	🛛 🕶 📮 🗄		2 💷 🐺 🛄 🗄	9 👑 📥 🚽	Ŧ	
	edCopy	m32def.ir	nc LedCopy.as	m X					+	Processor		~ 4	x
Г	/*								÷	Name	Value	2	
L	* L	edCopy.asi	n						*	Program Counter	0x00000004		
L	*									Stack Pointer	0x0000		
L		Author:	1-9-2015 9:45:1 okuiner	4						X Register	0x006C		
L	*/	Auction .	okuiper							7 Register	0x085F		
L										Status Register	กุกคุดง	NZC	-
		ldi	R16,0x00						=	Cycle Counter	0		
		out	DDRA,R16	; configure Port A	as 8 i	nput	S			Frequency			
		101	RID, UXFF	, configure Dont P		.+	+-			Stop Watch			
L		out	DDRD, KIO	, configure Port b	as o 0	rcpu				Registers			
	loop	: in	RØ,PINA	; read Port A (swi	tches)					R00	0x2F		
		out	PORTB,RØ	; output to Port B	(leds)					R01	0x00		
		rjmp	LOOP	; and again						R02	0x40		
										R03	0x4B		
										R04	0x21		
										R05	0x20		
										R06	A0x0		
I.								_	*	R07	0x47	_	Ψ.
	00 % 👻	<					-		*	🔍 ASF E 🏢 Pi	oce 💐 Soli	uti 🚰 Prop	e
۷	/atch 2				- 1	X	Memory 1					* Q	×
	Name		Value		Туре	^	Memory: p	rog FLASH		-			
H							prog 0x0000	00 e0 0	a bb	0f ef 07 bb 09 b	2 08 ba .à	.».ï.»º	
							prog 0x0000	fd cf f	f ff	ff ff ff ff ff f	fffffýÏ	yyyyyyyyyy	
							prog 0x0018	ff ff f	fff	ff ff ff ff ff ff	fffff ÿÿ	ууууууууууу	
							prog 0x0024		1 11	TT TT TT TT TT T	t tt tt yy	ууууууууууу	
							prog 0x0030		T TT	ff ff ff ff ff ff ff	f ff ff oo	уууууууууу	
							prog 0x0030	ff ff f	f ff	ff ff ff ff ff ff	f ff ff yy	000000000000000000000000000000000000000	
						-	prog 0x0054	ff ff f	f ff	ff ff ff ff ff i	f ff ff yy	VVVVVVVVVVV	-
	🙀 Autos	👼 Locals 💡	🛒 Watch 1 🛛 д W	atch 2			Breakp	🔲 Memo	r 🧦	Call Sta 🗾 Com	ım 🖅 İmm	ied 🧧 Out	put
Re	ady												

Figuur 9: Atmel Studio in JTAG-ICE mkII debug-mode.

In bovenstaande figuur zijn de afzonderlijke bits van de I/O-registers die bij poort A en B horen niet te zien. Er kan een *I/O View* geopend worden waarin dit wel te zien is. Klik middenboven op I/O view, zoals ook te zien is in figuur 10.



Figuur 10: Openen van het *I/O View*-scherm.

Klik vervolgens in het *I/O View*-scherm op *PORTA* of op *PORTB*. Hierdoor worden alle betrokken I/O-registers getoond.

3.5 Runnen van ledcopy.asm

We gaan de software nu runnen en volgen (tracen) via de JTAG-ICE mkII.

De gele pijl staat bij de eerstvolgende instructie die uitgevoerd gaat worden. Je kunt de code nu stap voor stap doorlopen door herhaaldelijk op F11 (Step Into) te drukken. Dit heet stepping. Bekijk alle veranderingen in het *I/O view-* en *Processor-*scherm. Controleer in het broncode-scherm (midden) of de software uiteindelijk in de lus belandt.

Verander nu een ingang op poort A door een drukknop op het AVR development bord (zie figuur 1) in te drukken. Houdt deze ingedrukt en stap steeds door de software heen. Je ziet nu in het *I/O View*-scherm bepaalde bits van *PINA* en *PORTB* veranderen.

Laat de drukknop los en doorloop de cyclus nogmaals. Je kan natuurlijk ook meerdere drukknoppen tegelijk gebruiken.

Met F5 kan de software ook real-time draaien; de software draait nu op ware snelheid in de AVRcontroller. Om te pauzeren (niet te stoppen) moet je Ctrl+F5 gebruiken of de pauze knop. Met het menu *Debug* > *Stop Debugging* of Ctrl+Shift+F5 keer je terug naar de edit-mode.

Figuur 11 laat het gebruik van twee drukknoppen zien (let op de *I/O View*).

Sluit het project af via het menu *File > Close Solution*.



Figuur 11: Tracen met twee knoppen ingedrukt.

3.6 Het programma *one_hz.asm*

We gaan nu een tweede programma invoeren. Dit programma zal de ledjes op poort B met een frequentie van 1 Hz laten knipperen. In listing 2 is het programma weergegeven.

Maak een nieuw project aan via het menu File > New > Project, kies weer voor Assembler. Vul daarna de code uit listing 2 in het geopende assemblerbestand in.

```
one_hz.asm - Simple 1 Hz pulse generator
;
;
    Author:
                <your name>
;
    Date:
                 <date>
;
    AVR type:
                ATmega32
;
                STK 500 (AVR development board)
    Target:
;
    Frequency: 3.6864 MHz
        LDI
                 r16.0xFF
        OUT
                 DDRB,r16
                              ; configure Port A as 8 inputs
        LDI
                 r17,0xFF
                              ; initialize LED state
START:
        COM
                 r17
                              ; complement (invert) this state
        OUT
                 PORTB, r17
                              ; show them on the LEDs
        LDI
                 r20,0x00
        LDI
                 r21,0xA0
        LDI
                 r22,0x05
                              ; initialize 3 byte counter.
WAIT:
        SUBI
                 r20,1
        SBCT
                 r21,0
        SBCI
                 r22,0
                              ; wait 0.5 secs by decrementing the
                              ; 3 byte counter until zero
        BRNF
                 WAIT
                 START
        RJMP
                              ; and again
                                Listing 2: one_hz.asm
```

3.7 Simulatie van one_hz.asm

Start de debugger door op F5 of de Play knop te drukken. Je krijgt weer een foutmelding en daarna de mogelijkheid om een debugger/programmer te selecteren. Dit keer selecteren we niet JTAGICE, maar de simulator. Dit heeft als effect dat het geassembleerde programma niet op de ATMega32A via de JTAGICE wordt gedraaid, maar nu dus op een simulator.

Je kunt trouwens op ieder moment switchen tussen de simulator en JTAGICE door bovenaan *Project* > $OneHz^5$ Properties te kiezen. Selecteer vervolgens de tabblad *Tool*.

Druk nadat je de simulator hebt geselecteerd weer op F5 of de Play knop. Je programma wordt nu gedraaid op de simulator en om deze te onderbreken kun je op de pauze knop (Break All, Ctrl+F5) drukken.

Zoals je aan de broncode kan zien, worden registers R16, R17 en R20 tot en met R22 gebruikt. In het *Processor*-scherm kun je de inhoud van deze registers bekijken. Dit *Processor*-scherm kun je bekijken door in het rechter-scherm op het tabblad *Processor* te klikken. Zorg er met behulp van de scrollbar voor dat de register R16 tot en met R22 zichtbaar zijn. Stap nu weer door de software heen met F11. In figuur 12 is te zien dat de simulatie al een paar stappen aan de gang is en dat R20 veranderd is (rode kleur).

⁵ Hier staat de naam van het project. Als je een andere naam aan het project hebt gegeven, dan staat deze naam er in plaats van OneHz.

Idd copy (Debugging) - AmetStudio Image: State of the state of			_					_				_						
File Edit View VAsist ASF Project Build Debug Tools Window Help Image: Construction of the state of the	🏶 L	edCopy (Del	bugging)	- AtmelStudio												_		×
<pre> i i i i i i i i i i i i i i i i i i i</pre>	File	Edit View	w VAssi	stX ASF Proje	ect Build Debug Tools Windo	w Help												
Image: Construction Image: Construction<	: 1	- 🗄 🔠	- 🞽 🔒	1 🖉 🕺 🖬 🕻	🛃 🔊 • (* - 📮 • 🖳 🔣 🔩	De Di De	ebug	- 12	9					•	🔜 😤 🛃	📴 繘 💌 •	- 1 💷	₹
one_bzasm × Processor * 3 × /* LedGopy.asm Name Value * Created: 2-9-2015 18:46:00 Name Value * Author: bkulper */ Name Value */ Idi r16.9xFF out DOB8,r16 ; configure Port A as 8 inputs Idi r17,9xFF ; initialize LED state R1 000 R1 000 R14 000 R15 000 R2 000 R1 000 Idi r17,9x8FF ; initialize 3 byte counter. R1 000 R2 000 R2 000 Idi r22,0x80 ; initialize 3 byte counter. R2 000 R2 000 R2 000 R2 000 R2 000 R2 000 R3 000 R3 <t< th=""><th>: 🔁</th><th>। 🗁 🌄 🕫</th><th>6 8 8</th><th>Oz ábc 🛱 📮</th><th>🛛 🕹 🖬 🗘 🖬 🕹</th><th>(I°I ∗I</th><th>1 Hex</th><th>-</th><th>÷ i 💐</th><th>ox.</th><th>💷 🙀</th><th></th><th>. : 🖄</th><th>**</th><th>🔺 🚽 i 📟</th><th>ATmega32A</th><th></th><th></th></t<>	: 🔁	। 🗁 🌄 🕫	6 8 8	Oz ábc 🛱 📮	🛛 🕹 🖬 🗘 🖬 🕹	(I°I ∗I	1 Hex	-	÷ i 💐	ox.	💷 🙀		. : 🖄	**	🔺 🚽 i 📟	ATmega32A		
/* Name Value /* LedCopy.ass Created: 2-9-2015 10:46:00 Author: bkuiper Author: bkuiper // Idi r16,0xFF out DDR8,r16 ; configure Port A as 8 inputs ldi r17,0xFF ; initialize LED state start: com r17 out PORTB,r17 ; complement (invert) this state out PORTB,r17 ; show them on the LEDs ldi r21,0x80 ; initialize 3 byte counter. wait: subi r21,0 idi r21,0x80 ; initialize 3 byte counter. wait: subi r21,0 sbci r21,0 5 idi r21,0 5 sbci r21,0 ; byte counter until zero rjmp start ; and again 100 % * * Mame Value Type // Prog 8x0000 red 50 e0 40 e0 e1 r7 f6 c1 red 20 e0 e1 prog 8x0000 red 50 e0 40 e0 e0 e1 r7 f6 c1 red 20 e0 e1 sol r22,0 0x00 r	one	_hz.asm ×											Proces	sor			-	Ψ×
LedCopy.asm Created: 29-2015 19:46:00 Author: bkuiper Author: bkuiper // Idi r16,0xFF out 00085,r16 ; configure Port A as 8 inputs Idi r17,0xFF ; initialize LED state start: com r17 ; complement (invert) this state out PORTs,r17 ; show them on the LEDs Idi r20,0x60 Idi r21,0x40 Idi r20,0x60 Idi r22,0x40 Idi r22,0x40 Idi r22,0x40 Idi r22,0x65 ; initialize 3 byte counter. wait: subi r20,1 sbci r22,0 start ; and again ID % * * Memory1 Memo		/*										÷		Nam	e	Value		
* Created: 2-9-2015 18:46:00 Author: bkuiper */ */ Idi r15,0xFf out DDRB,r15 idi r17,0xFF initialize LED state start: com out PORTB,r17 is model ldi idi r21,0x6 idi r22,0x85 idi r21,0x6 idi r21,0x6 idi r21,0x6 idi r21,0x6 sbci r22,0 sbci r22,0 idi r21,0 sbci r22,0 sbci r22,0 igit j wait: ; and again 100 % - * * Watch2 * * Memory1 * * * Memory1 * * * * * Memory1 * * * * * * <t< th=""><th></th><th>* LedC</th><th>opy.asm</th><th>1</th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th></th><th>R08</th><th></th><th>0:</th><th>00</th><th></th><th></th></t<>		* LedC	opy.asm	1									R08		0:	00		
Created: 2-9-2035 10:46:00 R10 0.000 Autor: bkuiper R10 0.000 '/ Idi r16, 0xFF configure Port A as 8 inputs Idi r17, 0xFF ; initialize LED state R1 0.000 start: com r17 ; complement (invert) this state R1 0.000 out PORT8,r17 ; show them on the LEDs R2 0.000 Idi r22,0 0.000 R2 0.000 Idi r22,0 ; show them on the LEDs R2 0.000 Idi r22,0 ; initialize 3 byte counter. R2 0.000 wait: subi r22,0 ; wait 0.5 secs by decrementing the brine R2 0.000 sbci r22,0 ; wait 0.5 secs by decrementing the brine R2 0.000 R2 0.000 R00 R2 0.000 R2 0.000 R2 0.000 R2 0.000 Name Value Type Memory1 Call Sack Proc. X X Prog 0x0000 eff ff		*											R09		0:	00		
<pre>Author: bkuiper */ Idi r16,0xFF out DDRB,r16 ; configure Port A as 8 inputs Idi r17,0xFF ; initialize LED state Start: com r17 ; complement (invert) this state out PONTB,r17 ; show them on the LEDs Idi r20,0x00 Idi r21,0x40 Idi r21,0x40 Idi r22,0x05 ; initialize 3 byte counter. wait: subi r20,1 sbci r21,0 sbci r21,0 sbci r21,0 sbci r22,0 ; wait 0.5 secs by decrementing the brne wait ; 3 byte counter until zero rjmp start ; and again ** Watch 2 *** Watch 2 *** Watch 2 *** Watch 2 *** Watch 1 Watch 2 *** Watch 2 *** *** *** *** *** *** *** *** *** *</pre>	* Created: 2-9-2015 10:46:00												R10		0:	00		
Idi r16,0xFF out DDBB,r16 ; configure Port A as 8 inputs Idi r17,0xFF ; initialize LED state start: com r17 ; complement (invert) this state out PORTB,r17 ; show them on the LEDs Idi r20,0x00 initialize 3 byte counter. Wait: subi r20,1 sbci r22,0 ; wait 0.5 secs by decrementing the prog sbci r22,0 y * Wemoy1 Watch2 * Wemoy1 Prog 0x0000 ef 07 bb if ef 10 95 18 bb 40 e05 9 .i.m.i@BP prog 0x0000 ef 07 bb if ef 10 95 18 bb 40 e05 9 .i.m.i@BP Prog 0x0000 ef 07 bb if ef 10 95 18 bb 40 e05 9 .i.m.i@BP Prog 0x0000 ef 07 bb if ef 10 95 18 bb 40 e05 9 .i.m.i@BP Prog 0x0000 ef 07 bb if ef 10 95 18 bb 40 e0 59 .i.m.i@BP Prog 0x0000 ef 07 bb 1f ef 10 95 18 bb 40 e0 59 .i.m.i@BP Prog 0x0000 ef 07 bb 1f ef 10 95 18 bb 40 e0 59 .i.m.i@BP Prog 0x0000 ef 07 bb 1f ef 10 95 18 bb 40 e0 59 .i.m.i@BP		* Au	thor: b	okuiper									R11		0:	00		
ldi r16,0xFF out DDR8,r15 ; configure Port A as 8 inputs ldi r17,0xFF ; initialize LED state start: com r17 ; complement (invert) this state out PDR8,r17 ; show them on the LEDs ldi r20,0x60 ; initialize 3 byte counter. wait: subir r20,0x60 ; initialize 3 byte counter. wait: subir r20,0 ; wait 0.5 secs by decrementing the bree wait ; 3 byte counter until zero rjmp start ; and again wato vote /22.0 Value Value Vexth2 4 Memory prog PLASH prog 0x0034 fff fff fff fff fff fff fff fff fff ff		-/											R12		0:	00		
Autors Outors configure Port A as 8 inputs Idi r17, 0xFF ; initialize LED state start: com r17 ; complement (invert) this state out PORTB,r17 ; show them on the LEDs Idi r20,0x8 ; initialize 3 byte counter. wait: subi r20,1 sbci r22,0 ; subi sbci r22,0 ; wait 0.5 secs by decrementing the brne wait ; 3 byte counter until zero r28 rjmp start ; and again wood Watch 2 Value Value prog 0x0000 eff ef of bb if ef 10 95 18 bb 40 e0 50 .i.m.i			145	CIG OVER									R13		0:	00		
Idi r17,0xFF ; initialize LED state start: com r17 ; complement (invert) this state out PORTB,r17 ; show them on the LEDs ldi r22,0x80 jiii r22,0x80 ldi r22,0x85 ; initialize 3 byte counter. wait: suit r22,0 ; wait 0.5 secs by decrementing the brne R25 sbci r21,0 ; and again R26 uo % ord Value Type Memory1 @ ASu; @ Pro Wath2 value Type Memory1 @ CallStack 2 Comman @ Immediat © Output			out	DDRB.r16	: configure Port A as 8 in	outs						=	R14		05	00		
ldi r17, 0xFF ; initialize LED state start: com r17 ; complement (invert) this state out PORTB,r17 ; show them on the LEDs ldi r20,0x00 R18 0x00 ldi r22,0x05 ; initialize 3 byte counter. R23 0x00 wait: subi r20,1 start ; 3 byte counter. R24 0x00 sbci r21,0 ; wait 8.5 secs by decrementing the brne wait ; 3 byte counter until zero R25 0x00 rjmp start ; and again R24 0x00 R25 0x00 100 % Memory prog %x0000 R26 0x00 R27 0x00 R23 0x00 R29 0x00 </td <th></th> <td></td> <td></td> <td></td> <td>,</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>R15</td> <td></td> <td>03</td> <td>00</td> <td></td> <td></td>					,								R15		03	00		
start: com r17 ; complement (invert) this state out PORTB,r17 ; show them on the LEDs ldi r20,0x00 ldi r21,0x00 ldi r22,0x05 idi r22,0x05 sbci r21,0 sbci r21,0 sbci r22,0			ldi	r17,0xFF	; initialize LED state								P17		0	00		
<pre>start: com r17 ; complement (invert) this state out PORTB,r17 ; show them on the LEDs ldi r20,0x00 ldi r21,0xA0 ldi r22,0x05 ; initialize 3 byte counter. wait: subi r20,1 sbci r21,0 /pre>													R18		0	-00		
out PORTB,r17 ; show them on the LEDs ldi r28,0x00 ldi r21,0x00 sbci r22,0x05 ; initialize 3 byte counter. wait: subi r22,0 sbci r21,0 sbci r22,0 sbci r22,0 rjmp start sbci r22,0 sbci r20,0		start:	com	r17	; complement (invert) this	state							R19		0	-00		
Idi r20,8x80 Idi r21,9x80 Idi r22,9x95 initialize 3 byte counter. R21 wait: subi sbci r21,0 sbci r21,0 sbci r21,0 sbci r21,0 sbci r21,0 sbci r21,0 sbci r22,0 sbci r20,0 sbci r20,0 sbci r20,0 sbci r20,0 sbci r20,0 sbci <th></th> <td></td> <td>out</td> <td>PORTB, r17</td> <td>; show them on the LEDs</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>R20</td> <td></td> <td>0</td> <td>(F1</td> <td></td> <td></td>			out	PORTB, r17	; show them on the LEDs								R20		0	(F1		
111 r22,0x00 121 r22,0x00 121 r22,0x00 121 r22,0x00 121 r22,0x00 121 r22,0x00 121 r22,0x00 122 0x00 123 0x00 124 0x00 125 0x00 125 0x00 126 0x00 127 0x00 128 0x00 129 0x00 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120 % 120			1.44	-20 0.00									R21		0	5D		
Idi r.22,0x05 ; initialize 3 byte counter. wait: subi r.20,1 sbci r.21,0 sbci r.22,0 ywait: ; wait 0.5 secs by decrementing the brne brne wait rjmp start rjmp start yaar memory wate: yaar yaar yaar yaaar yaaar yaaar </td <th></th> <td></td> <td>ldi</td> <td>r21 0x40</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>R22</td> <td></td> <td>0:</td> <td>:05</td> <td></td> <td></td>			ldi	r21 0x40									R22		0:	:05		
wait: subi r20,1 sbci r21,0 sbci r22,0 j wait 0.5 secs by decrementing the brne wait j 3 byte counter until zero rjmp start j and again R24 0x00 R25 0x00 R26 0x00 R27 0x00 R28 0x00 R30 0x00 R30 0x00 R31 0x00 R			ldi	r22.0x05	: initialize 3 byte counter	r.							R23		0:	:00		
wait: subi r20,1 sbci r21,0 sbci r22,0 sbci r22,0 sb				,	,								R24		0:	00		-
sbci r21,0 sbci r22,0 sbci r22,0 brne wait rjmp start rjmp start start sadain void sadain void <th></th> <td>wait:</td> <td>subi</td> <td>r20,1</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>R25</td> <td></td> <td>0:</td> <td>00</td> <td></td> <td>-</td>		wait:	subi	r20,1									R25		0:	00		-
sbci r22,0 ; wait 0.5 secs by decrementing the brne P27 0.00 prime wait ; 3 byte counter until zero P28 0.00 rjmp start ; and again P29 0.00 100 % - 4 P P Pro Pro Sol Pro Watch 2 P Memory1 Pro Sol Pro Sol Pro Name Value Type Memory1 Prog 0x0000 e6 54 150 59 40 60 40 e1 77 66 cf. éeàAPP@'éa+01 Prog 0x0000 e6 55 e0 41 50 59 40 60 40 e1 77 66 cf. éeàAPP@'éa+01 Prog 0x0021 ff	\Rightarrow		sbci	r21,0									R26		0:	00		
brne wait ; 3 byte counter until zero rjmp start ; and again 100 % ~ 4 Prome Prome Watch 2 • P × Memory: prog FLASH • P × Memory: prog FLASH • P × Prog 0x0000 e6 f 07 bb 1f ef 10 95 18 bb 40 e0 50 1.1.9.10@P prog 0x0000 e6 f 07 bb 1f ef f ff			sbci	r22,0	; wait 0.5 secs by decreme	nting the							R27		0:	00		
rjmp start ; and again R29 0.00 100 % % % % % % % Watch 2 • % % <th></th> <td></td> <td>brne</td> <td>wait</td> <td>; 3 byte counter until zero</td> <td>0</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>R28</td> <td></td> <td>0:</td> <td>00</td> <td></td> <td></td>			brne	wait	; 3 byte counter until zero	0							R28		0:	00		
R30 0.600 Nome Value Valu			adam	staat	, and again								R29		0:	00		
R31 0.00 100 % * Watch 2 * Watch 2 * Prome * Name Value Prog 0x0000 0f ef 07 bb 1f ef 10 95 18 bb 40 e0 50 Prog 0x0000 e6 5 e0 41 50 50 40 60 40 e1 f7 f6 cf Prog 0x0000 ea 65 e0 41 50 50 40 60 40 e1 f7 f6 cf Prog 0x0001 Aff ff			1. Juib	Start	; and again								R30		0:	00		
100 % • • Image: Im												Ŧ	R31		0:	.00		Ψ.
Watch 2 • • •	100	% - 1										•	I IO		🔍 AS 📴	Pro 💐 S	ol 🚰	Pro
Name Value Type Memory: prog FLASH Image: Constraint of the start of	Wat	ch 2				- ₽	× Mem	ory 1									-	Ψ×
prog 0x0000 0f ef 07 bb 1f ef 10 95 18 bb 40 e0 50 .i.w.i0ghP prog 0x0000 ea 65 e0 41 50 50 40 60 40 e1 7f 6c f eeAAPP@`@4+6I prog 0x001A ff	N	ame		Value		Туре	^ Men	nory: p	rog FLA	SH				-				
prog 0x0000 ea 65 e0 41 50 50 40 60 40 e1 77 f5 cf eabAPP@ @4+0I prog 0x001A ff						_	prog	0x0000	Øf	ef 07	bb 1	f ef	10 95	18	bb 40 e0 5	50 .ï.».ï.	»@àP	-
prog 8x801.A ff							prog	0x0000	ea	65 eØ	41 5	0 50	40 60	40	e1 f7 f6 (f êeàAPP@	`@á÷öÏ	
prog 0x002/ ff							prog	0x001/	ff	ff ff	ff f	f ff	ff ff	ff	ff ff ff t	f yyyyyyy	уууууу	
prog 0x0041 ff							prog	0x002	11 44	TT TT	TT T	T TT	11 11	TT .	TT TT TT 1	T <u>yyyyyyy</u>	уууууу	
Image: Second state of the second s							prog	0x0034	ff	ff ff	ff f	f ff	ff ff	ff	ff ff ff i	f www.www	yyyyyy	
▼ prog 0x0058 ff							prog	0x0048	ff	ff ff	ff f	f ff	ff ff	ff	ff ff ff i	f vvvvvvv	000000	
🐺 Autos 🐺 Locals 🐺 Watch 1 😹 Watch 2 🐻 Breakpoi 🔳 Memory 1 🖧 Call Stack 🗩 Comman 👼 Immediat 🗐 Output							- prog	0x0058	ff	ff ff	ff f	f ff	ff ff	ff	ff ff ff t	f yyyyyyy	ÿÿÿÿÿÿ	-
		Autos 👼	Locals 🌡	🛒 Watch 1 🖉	Watch 2		To B	reakpoi.		Memo	ry 1	Cal	l Stack		Comman	🧰 Immediat	🔳 01	utput
Stopped Ln 21 Col1 Ch1 INS	Stop	ped								Ln	21		Col 1		Ch 1		IN	si

Figuur 12: Atmel Studio na het simuleren van een deel van one_hz.asm.

Schuif de scrollbar van het *Processor*-scherm nu helemaal naar boven. Hiermee krijg je de details van de processor te zien (figuur 13):

Processor	*	Ψ×
Name	Value	
Program Counter	0x0000009	
Stack Pointer	0x0000	
X Register	0x0000	
Y Register	0x000x0	
Z Register	0x000x0	
Status Register	ITHSVNZC	
Cycle Counter	84639	
Frequency	1,000 MHz	=
Stop Watch	84.639,00 µs	
Registers		
R00	0x00	
R01	0x00	
R02	0x00	
R03	0x00	
R04	0x00	
R05	0x00	
R06	0x00	
R07	0x00	
R08	0x00	
R09	0x00	
R10	0x00	
R11	0x00	
R12	0x00	
R12	0_00	
🔄 10 🔍 AS	📲 Pro 🔍 Sol 😁 I	Pro

Figuur 13: De processor frequentie is zichtbaar in het Processor-scherm.

Hier valt op dat de klokfrequentie van de AVR in de simulator op 1 MHz staat; in werkelijkheid is de frequentie 3,6864 MHz. Dit kan je instellen door te dubbelklikken op de frequentie en deze aan te passen.

Handig is het om vanaf de eerste instructie in je programma de klokfrequentie correct in te stellen. Selecteer daarom *Debug > Start Debugging and Break* (of gebruik sneltoets Alt+F5) en verander daarna de klokfrequentie. Je kunt vervolgens stap voor stap door de software lopen met F11. Na een tijdje zal het je opvallen dat je in een lus terecht bent gekomen en dat steeds dezelfde instructies worden doorlopen.

3.8 Breakpoints

Eigenlijk willen we dit stukje code op volle snelheid laten draaien en de simulator laten pauzeren als de lus is afgelopen. Dit kan je doen door het zetten van een breakpoint.

Verplaats de cursor naar de regel waar de instructie rjmp staat. Voeg nu een breakpoint toe via het menu *Debug > Toggle Breakpoint* of gebruik F9. Er verschijnt een rode stip links van de regel.

Laat de simulator nu op volle snelheid draaien door F5 te drukken. Nadat de lus is afgelopen, zal de simulator pauzeren en de gele pijl ligt over de rode stip, zie figuur 14. Let op: afhankelijk van de gebruikte PC kan het wel even duren voor het breakpoint is bereikt!

ne_hz.asm ×	LedCop	у			Processor		
/*				+	Name	Value	
* Led	opy.asm	1		*	Program Counter	0x0000000C	
*					Stack Pointer	0x0000	
* Cre	ated: 2	-9-2015 10:4	5:00		X Register	0x0000	
* AL	thor: b	kuiper			Y Register	0x0000	
*/					Z Register	0x0000	
					Status Register	ITHSVNZC	
	ldi	r16,0xFF		=	Cycle Counter	1843207	
	out	DDRB, r16	; configure Port A as 8 inputs	1	Frequency	3,686 MHz	-
	1.44	-17 AVEE	, initializa LED state		Stop Watch	500.001,90 µs	
	101	117,0211	; initialize LCD state		Registers		
start:	com	r17	; complement (invert) this state		R00	0x00	
	out	PORTB,r17	; show them on the LEDs		R01	0x00	
					R02	0x00	
	ldi	r20,0x00			R03	0x00	
	Idi	r21,0xA0			R04	0x00	
	101	r22,0x05	; initialize 3 byte counter.		R05	0x00	
unit.	cubi	-20 1			R06	0x00	
wart.	shci	r20,1			R07	0x00	
	shci	r22.0	: wait 0.5 secs by decrementing the		R08	0x00	
	brne	wait	: 3 byte counter until zero		R09	0x00	
			,,		R10	0x00	
	rjmp	start	; and again		R11	0x00	
-					R12	0x00	
				*	R13	0_00	*
0% • 4				,	🔄 IO 🔍 AS	👘 Pro 🔍 Sol 👔	Pro

Figuur 14: De simulator is gestopt bij het breakpoint.

In het *Processor*-scherm kan je nu informatie vinden over de toestand van de processor. Bekijk vooral de *Cycle Counter* en *Stop Watch*.

3.9 Watch

Het is ook mogelijk om een paar registers of I/O-poorten te volgen zonder dat je daarvoor in het *Processor*- of *I/O View*-scherm alles moet uitklappen (soms lukt dat ook niet omdat de onderdelen te ver uit elkaar liggen en het beeldscherm te klein is). Je kan dit doen via het *Watch*-scherm dat onderaan is te vinden.

Ga met de cursor in het broncode-scherm op een registernaam of I/O-registernaam staan en klik dan één keer zodat daar de cursor komt te staan.

Selecteer via het menu *Debug* > *QuickWatch* of type Shift+F9. Je krijgt nu een scherm te zien met daarin de naam van het register of I/O-register dat je geselecteerd hebt. Klik dan op *Add to Watch* om deze toe te voegen (figuur 15):

		<u>R</u> eevaluate
	•	Add Watch
Name	Value	Туре
🧼 r20	0	byte{reg

Figuur 15: Een register toevoegen via het QuickWatch-scherm.

Je kan ook met de rechtermuisknop op een register of I/O-register klikken. Er verschijnt dan een context-menu met daarin een *Add Watch* optie. Klik hierop om het register of I/O-register toe te voegen aan het *Watch*-scherm.

Daarna wordt het toegevoegde register getoond in het onderste *Watch*-scherm. Het is ook mogelijk om een geheugenplaats aan het *Watch*-scherm toe te voegen.

3.10 Werking one_hz.asm

Hoe werkt het programma one_hz.asm nu eigenlijk? In grote lijnen werkt het programma als volgt:

- Poort B wordt geconfigureerd voor output zodat we via I/O-register PORTB de leds kunnen aansturen.
- R17 wordt gevuld met allemaal enen.
- Vervolgens wordt de volgende eeuwig durende lus uitgevoerd.
 - De waarde in R17 wordt geïnverteerd en naar de leds geschreven.
 - Er wordt een (drie bytes brede) teller geladen met een bepaalde waarde.
 - De teller wordt steeds met één verlaagd totdat de waarde 0 wordt bereikt.
 - De waarde waarmee de teller wordt geladen is zodanig gekozen dat het aftellen naar de waarde 0 exact een halve seconde duurt.

Noot: Het op deze manier wachten tot een bepaalde tijd is verlopen wordt busy waiting genoemd. In H3 van het boek wordt een andere manier besproken om een langere tijd te wachten. De in het boek besproken methode is echter veel minder handig dan de in deze handleiding gebruikte methode.

Om de waarde te kunnen bereken die in de (drie bytes brede) brede teller moet worden geladen, moeten we het programma tot in de kleinste details begrijpen. Het eerst deel van het programma tot aan het label WAIT zou je tot in detail moeten begrijpen als je het *ledcopy.asm* programma uit paragraaf 3.2 hebt begrepen, al zullen de waarden die in R20, R21 en R22 geladen worden nog een raadsel voor je zijn. De COM R17 instructie inverteert de waarde in register R17, zie indien nodig [3, blz. 70]. De registers R20, R21 en R22 worden samen gebruikt als één teller van drie bytes, zie figuur 16. Hierbij is R20 het LSB (Least Significant Byte) en R22 het MSB (Most Significant Byte).



Figuur 16: Drie registers vormen samen een teller van drie bytes (= 24 bits).

De drie instructies:

SUBI	R20,1
SBCI	R21,0
SBCI	R22,0

zorgen ervoor dat de teller met één wordt verlaagd. Dit werkt als volgt: de SUBI R20,1 instructie verlaagt register R20 met één. Als register R20 bijvoorbeeld de waarde 3 bevat voordat de SUBI instructie wordt uitgevoerd, dan zal dit register na afloop van deze instructie de waarde 2 bevatten. De carry-flag in het AVR statusregister, zie [3, blz. 71], zal in dit geval gecleard worden omdat er niet "geleend" hoeft te worden om van de waarde 3, de waarde 1 af te trekken. De zero-flag wordt eveneens gecleard omdat het resultaat van deze bewerking (3 - 1) ongelijk aan nul is. Als daarna de instructie SBCI R21,0 wordt uitgevoerd, dan wordt de berekening (R21) – 0 – (carry-flag) uitgevoerd. Als de carry flag de waarde 0 heeft, dan heeft deze instructie dus geen effect.

Als R20 echter de waarde 0 bevat voordat de instructie SUBI R20,1 wordt uitgevoerd, dan zal dit register na afloop van deze instructie de waarde 0xFF bevatten. De carry-flag zal in dit geval geset worden omdat er "geleend" moet worden om van de waarde 0, de waarde 1 af te trekken. Als daarna de SBCI R21,0 instructie wordt uitgevoerd, dan wordt register R21 met één verlaagd (omdat de carry-flag geset was). De één die "geleend" moet worden, wordt dus "geleend" bij het meer significante byte van de teller. Als er geleend moet worden om de instructie SBCI R21,0 uit te kunnen voeren, dan wordt bij R22 geleend (door de instructie SBCI R22,0).

Omdat de SBCI instructie de zero-flag alleen cleart en nooit set, zie [1, blz. 149], is de zero-flag alleen maar geset als alle drie registers na afloop van de drie subtract instructies nul geworden zijn. Zie voor verdere uitleg van de SUBI en SBCI instructies [3, blz. 164–166]. De BRNE WAIT instructie zorgt ervoor dat naar het label WAIT wordt gesprongen zolang de zero-flag niet gelijk is aan nul, zie eventueel [3, blz. 108].

Om er voor te zorgen dat het een halve seconde duurt voordat deze sprong naar het label WAIT niet meer wordt genomen moeten we weten hoeveel klokcycles van de processor de instructies in deze loop duren. In Appendix *AVR Instructions Explained* van [3] kun je vinden hoeveel klokcycles elke instructie duurt. Dit kun je ook vinden in [1]. De SUBI en de SBCI instructies duren elk één clockcycle en de BRNE instructie duurt twee kockcycles (als de sprong wordt genomen). In totaal duurt de loop dus 1 + 1 + 1 + 2 = 5 klokcycles. De klokfrequentie van de AVR op het AVR development board staat ingesteld op 3,6864 MHz. Één klokcycle duurt dus 1/3686400 s. De loop duurt dus 5/3686400 s. Als we willen dat de loop 0,5 s duurt, dan moet de loop 0,5 / (5/3686400) keer worden uitgevoerd. (1/2) / (5/3686400) is gelijk aan⁶ (1/2) * (3686400/5) = 3686400 / 10 = 368640. Om de loop 368640 keer uit te voeren moet de drie bytes teller geladen worden met de waarde 368640 (= 0x05A000⁷). R22 (het MSB) moet dus worden geladen met 0x05, R21 moet worden geladen met 0xA0 en R20 (het LSB) moet worden geladen met 0x00.

⁶ Delen door een breuk is hetzelfde als vermenigvuldigen met het omgekeerde.

⁷ Het getal 368640 omzetten naar hexadecimaal kan eenvoudig door in Google in te typen: 368640 = hex.

3.11 Uitvoeren van one_hz.asm op het AVR development board

Voer het programma *one_hz.asm* tot slot uit op het AVR development board. Dit doe je door bovenaan *Project* > *OneHz*⁸ *Properties* te kiezen. Selecteer vervolgens de tabblad *Tool*. Je kunt nu de JTAG-ICE mkII programmer kiezen.

4 Atmel Studio sneltoetsen

Pictogram	Sneltoets	Werking/betekenis
H	F7	Assembleert de huidige file
	Ctrl+F7	Assembleert en laadt de file, en start de debug-omgeving
•	Ctrl+Shift+Alt+F5	Start de debug-omgeving (JTAG-ICE mkII of simulator)
≣↓	F5	Start het programma in de AVR
00	Ctrl+F5	Zet de debug-omgeving in pause (code wordt stilgelegd)
	Ctrl+Shift+F5	Sluit de debug-omgeving af
۲	F9	Toggle Breakpoint
S I	F11	Stapt steeds één instructie verder, volgt subroutines
(" =	F10	Stapt steeds één instuctie verder, subroutines worden als één instructie behandeld
C_	Shift+F11	Draait de huidige subroutine op volle snelheid en gaat over in pauze als die subroutine verlaten wordt
∃‡	Alt+F5	Stapt automatisch door de code heen, laat na elke stap up- dates in het I/O View zien
5	Shift+F5	Reset de debug-omgeving en laat de processor opnieuw be- ginnen.
69	Shift+F9	Opent het Add To Watch menu
A	Alt+1	Schakelt het QuickWatch menu aan en uit

Tabel 1	:	AVR	Studio	sneltoetsen.

Referenties

- [1] Atmel. 8-bit AVR instruction set. Nov 2016. URL: http://ww1.microchip.com/downloads/en/ devicedoc/atmel-0856-avr-instruction-set-manual.pdf (zie pag. 16).
- [2] Pieter Burghouwt. Introductie AVR-microcontroller. De Haagse Hogeschool. 2006 (zie pag. 2).
- [3] Muhammad Ali Mazidi, Sarmad Naimi en Sepehr Naimi. *AVR Microcontroller and Embedded Systems: Using Assembly and C.* International Edition. Pearson, 2013 (zie pag. 2, 4, 5, 15, 16).

⁸ Hier staat de naam van het project. Als je een andere naam aan het project hebt gegeven, dan staat deze naam er in plaats van OneHz.