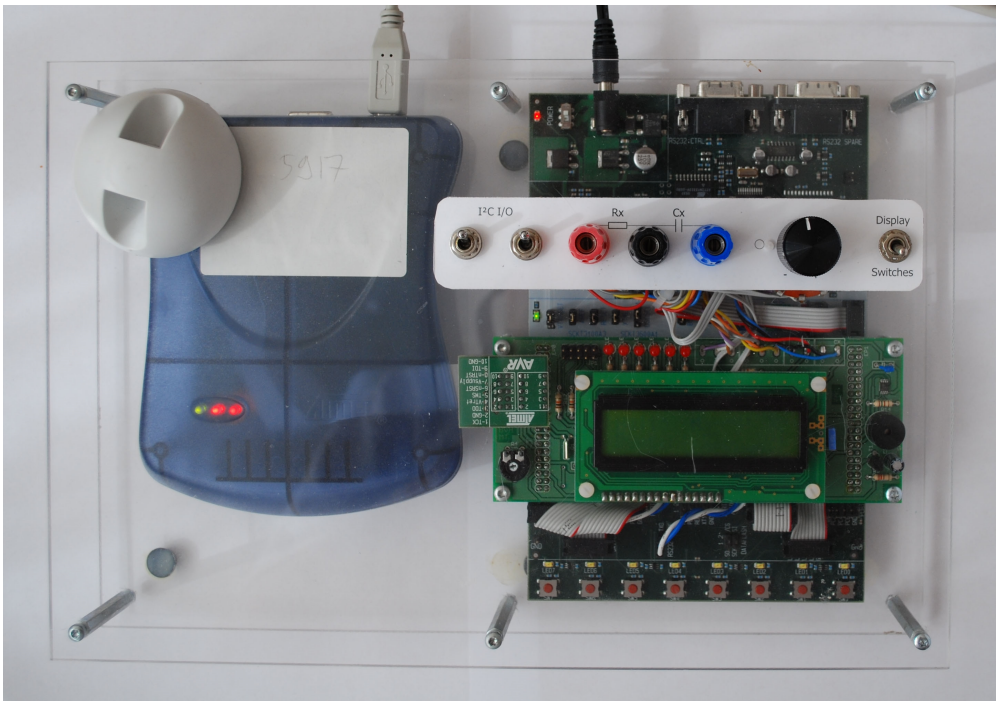
# HANDLEIDING

## AVR-ASSEMBLER PROGRAMMEREN

VERSIE 3.4$\alpha$




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

DE HAAGSE HOGESCHOOL

## Inhoudsopgave



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].

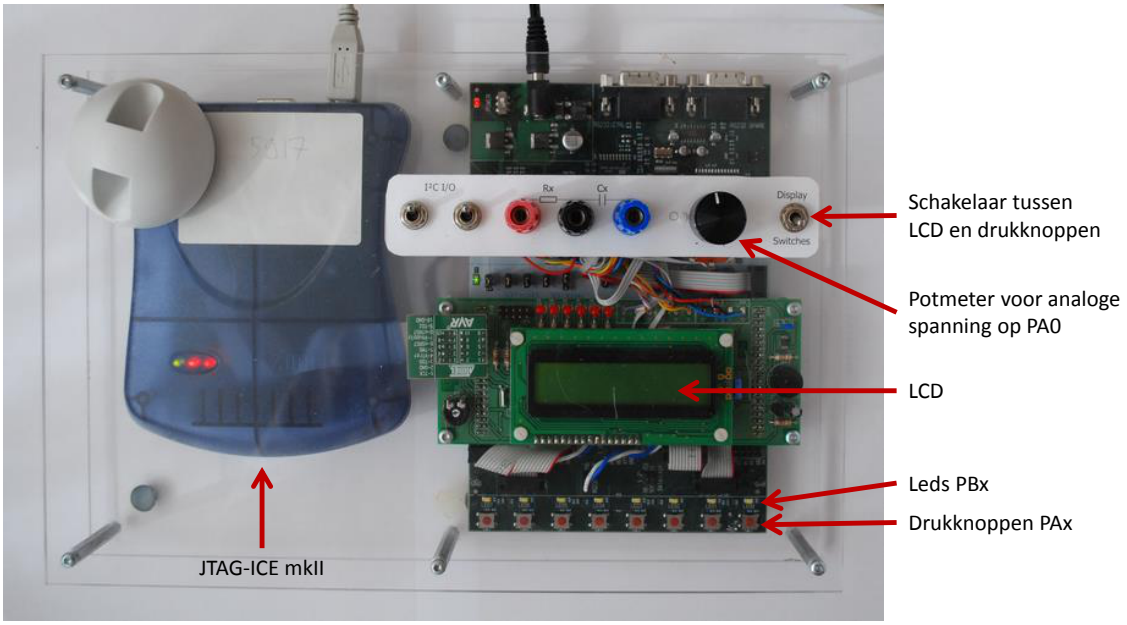
# 2 De ontwikkeling van software voor de AVR

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

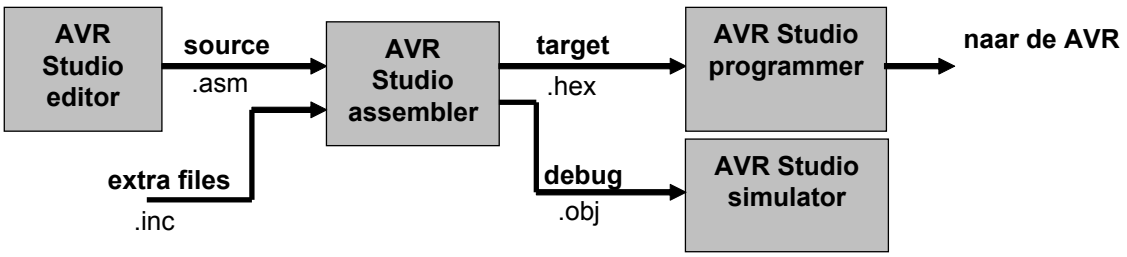
De software is gratis te downloaden en te gebruiken, ook voor commerciële doeleinden. We werken op de Haagse Hogeschool met Atmel Studio 6.2. We adviseren je om thuis ook versie 6.2 te installeren. De installatiebestanden kun je vinden op de Blackboard course van INLMIC onder *Documents > Links en downloads*.

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 EEPROM[1] 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
        LDI     R16,0xFF
        OUT     DDRB,R16        ; configure Port B as 8 outputs

LOOP:   IN      R0,PINA         ; read Port A (switches)
        OUT     PORTB,R0        ; 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.

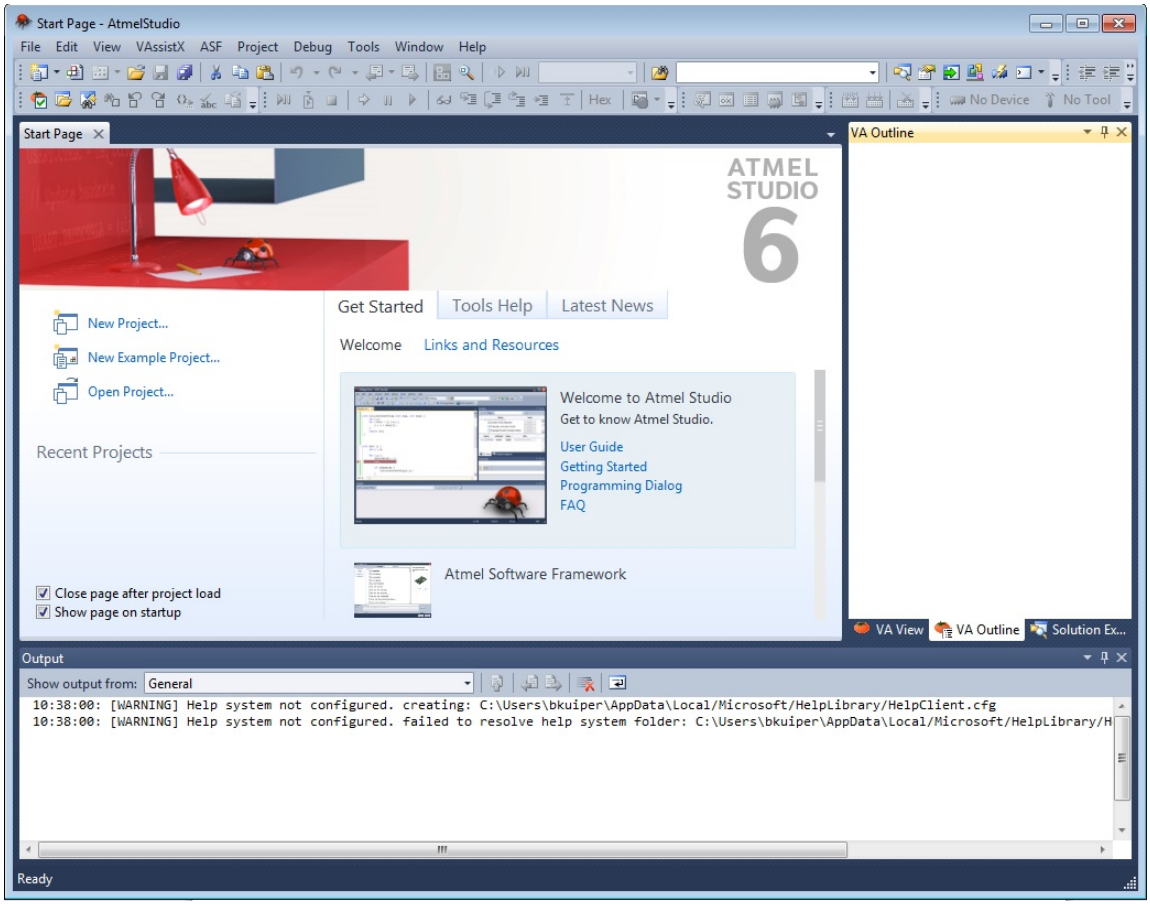
- In [3, blz. 56–57] worden de general purpose registers van de AVR besproken, `R16` is één van deze registers. In dezelfde paragraaf wordt ook de `LDI` instructie behandeld. *Lees deze twee pagina's uit het boek voordat je hier verder leest!* Na het lezen van deze paragraaf zal je begrijpen waarom de 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.

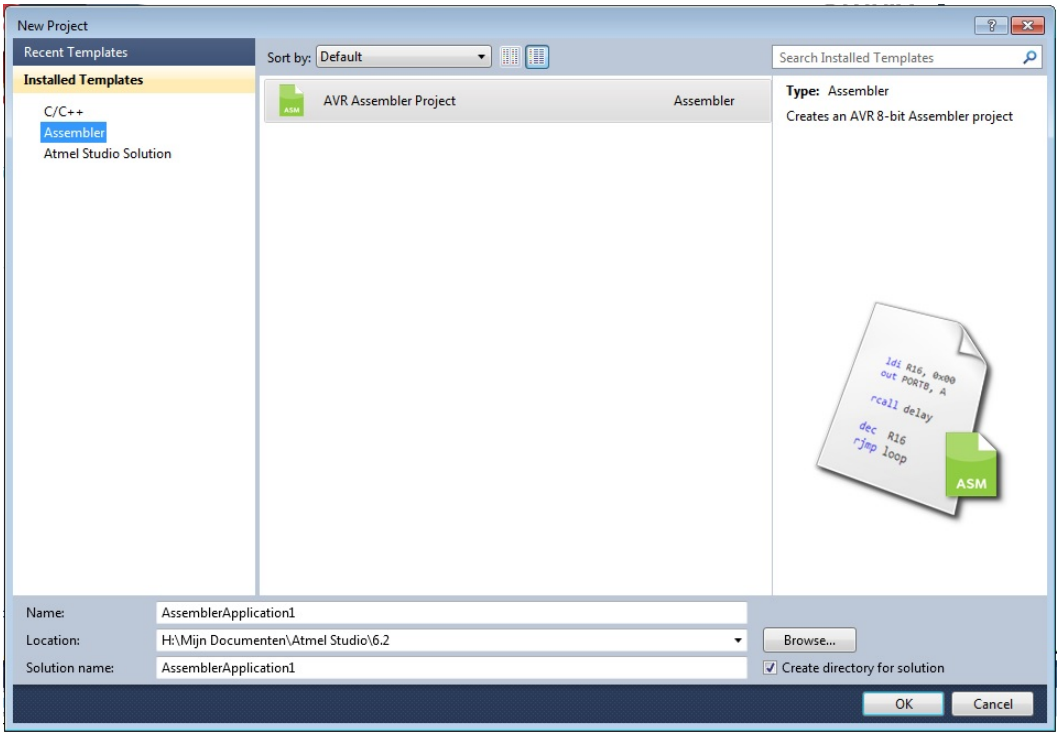

Figuur 3: Het pictogram van Atmel Studio.

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):

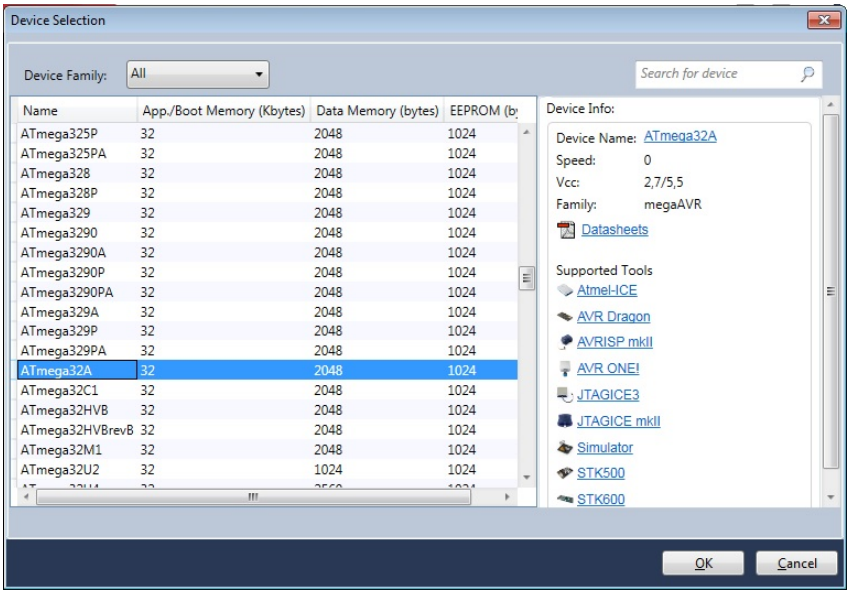



Figuur 5: Het scherm waarin nieuwe projecten kunnen worden aangemaakt.

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.

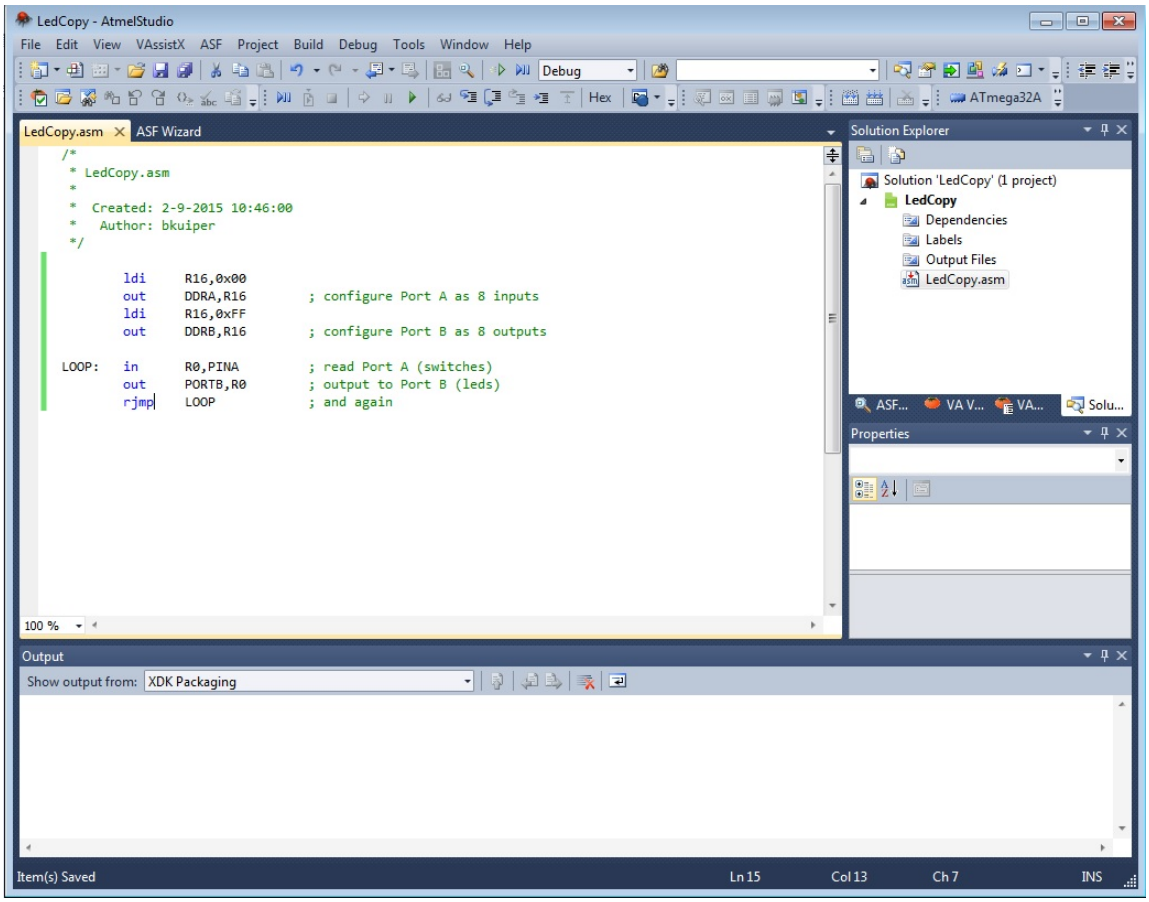



Figuur 6: Het scherm waarin het AVR-type geselecteerd moet worden.

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 ziet[3].

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).

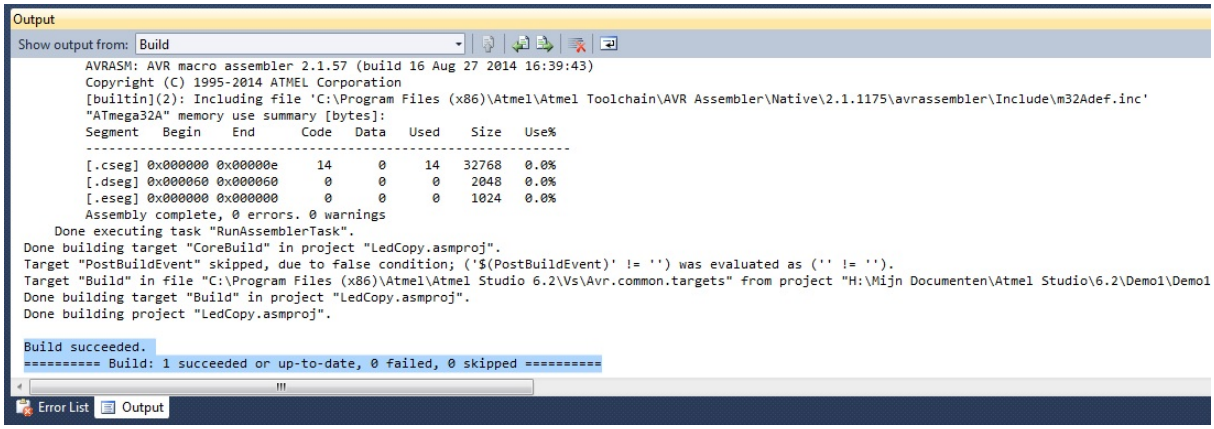



Figuur 7: Atmel Studio na het invoeren van *ledcopy.asm*.

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.



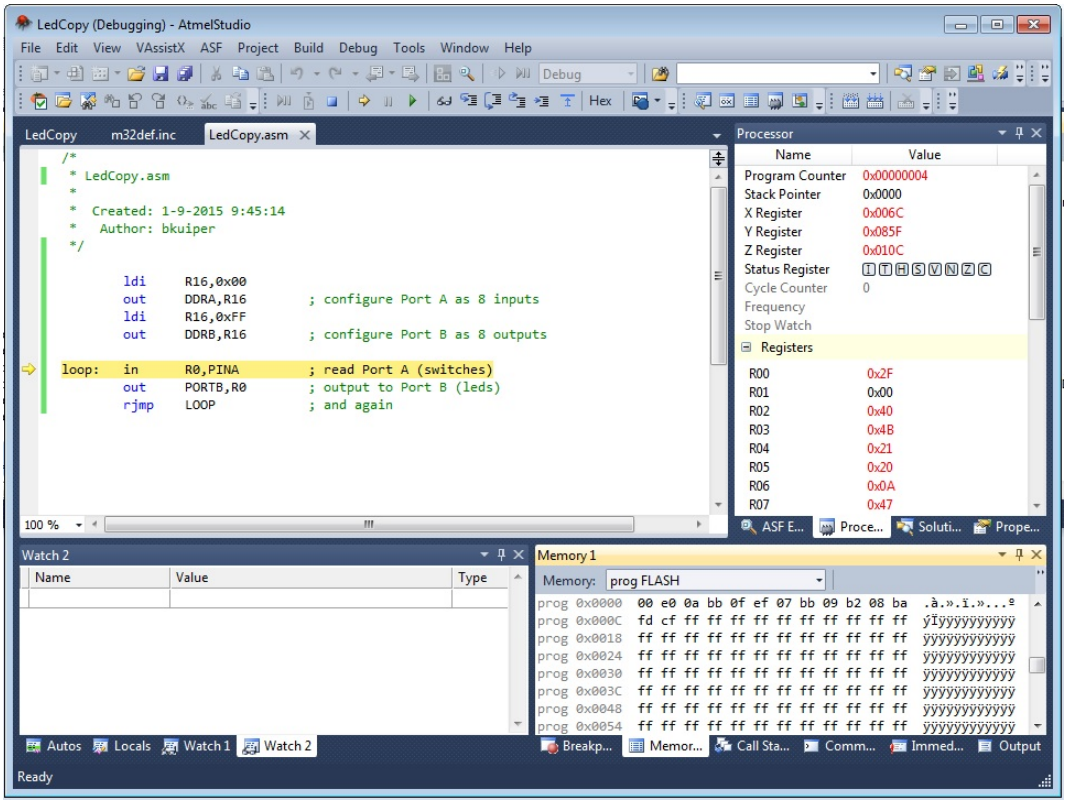

Figuur 8: Atmel Studio assembler output na het succesvol assembleren van *ledcopy.asm*.

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 mkII

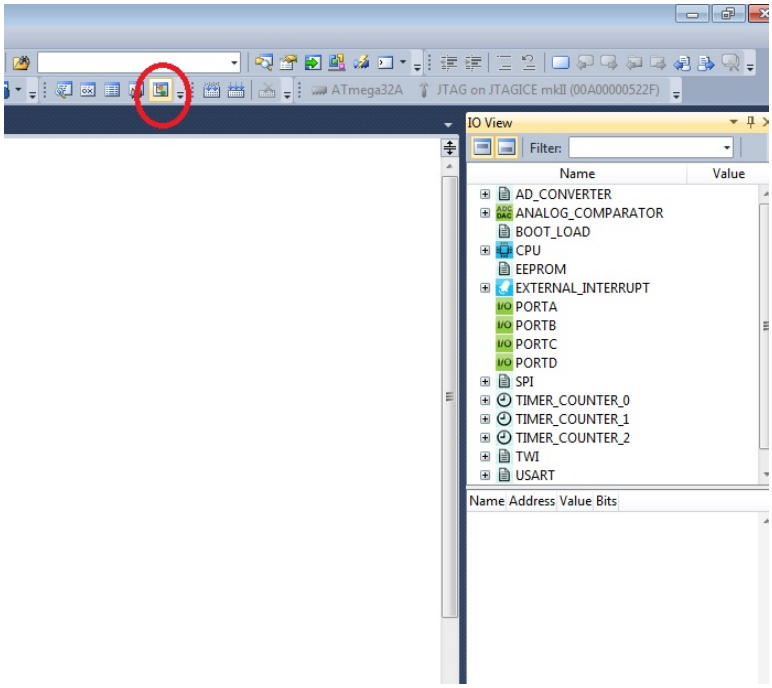
Het foutloos assembleren van het bestand wil nog niet zeggen dat de software correct werkt. Om na te gaan of het programma doet wat het zou moeten doen, kan de werking op het AVR development board (zie paragraaf 2.1) worden gevolgd. Dit wordt het tracen van een programma genoemd. Het opsporen van fouten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




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.

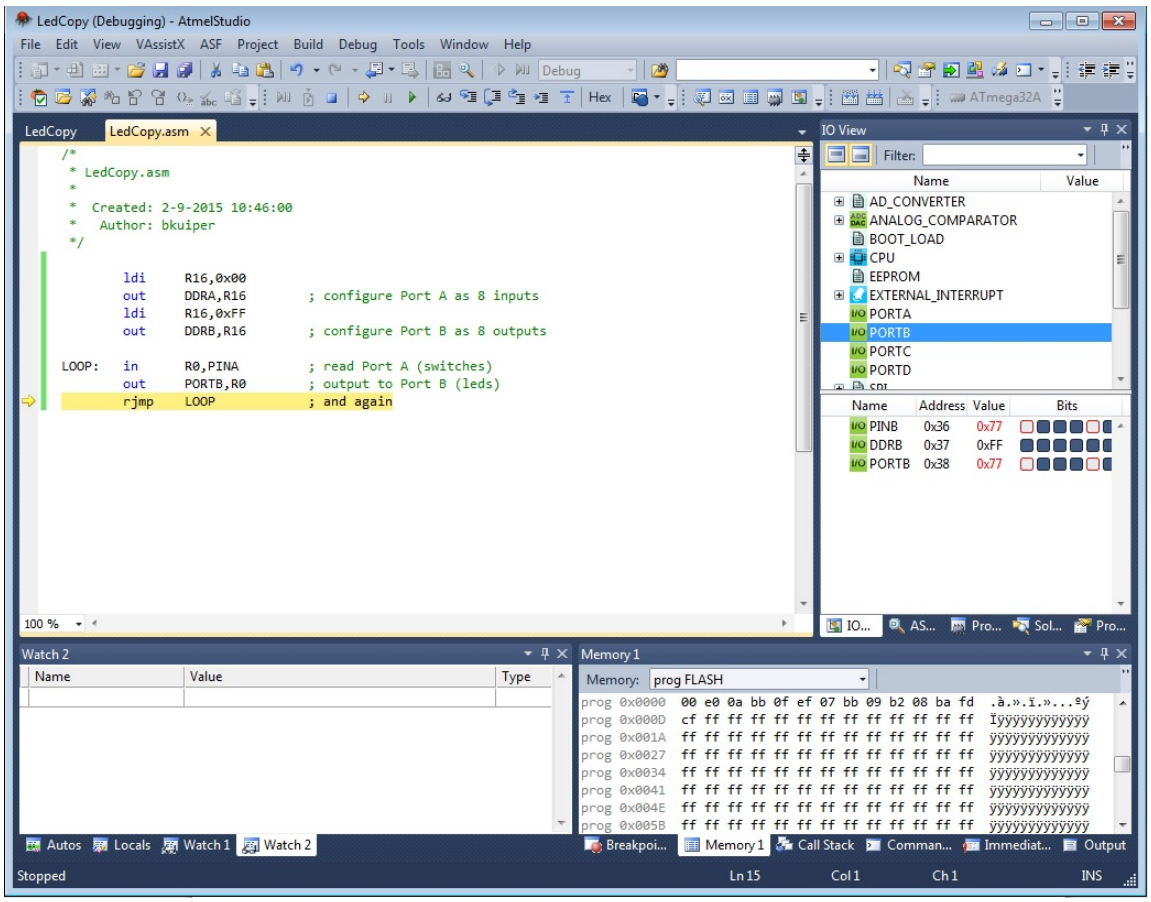
Verander nu een ingang op poort A door een drukknop op het AVR development bord (zie figuur 1) in te drukken. Houdt deze ingedrukt en stap steeds door de software heen. Je ziet nu in het *I/O View*-scherm bepaalde bits van *PINA* en *PORTB* veranderen.

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 AVR-controller. 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
;   Target:      STK 500 (AVR development board)
;   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
        SBCI    r21,0
        SBCI    r22,0       ; wait 0.5 secs by decrementing the
        BRNE    WAIT        ; 3 byte counter until zero

        RJMP    START       ; and again
```

Listing 2: one_hz.asm

### 3.7 Simulatie van *one_hz.asm*

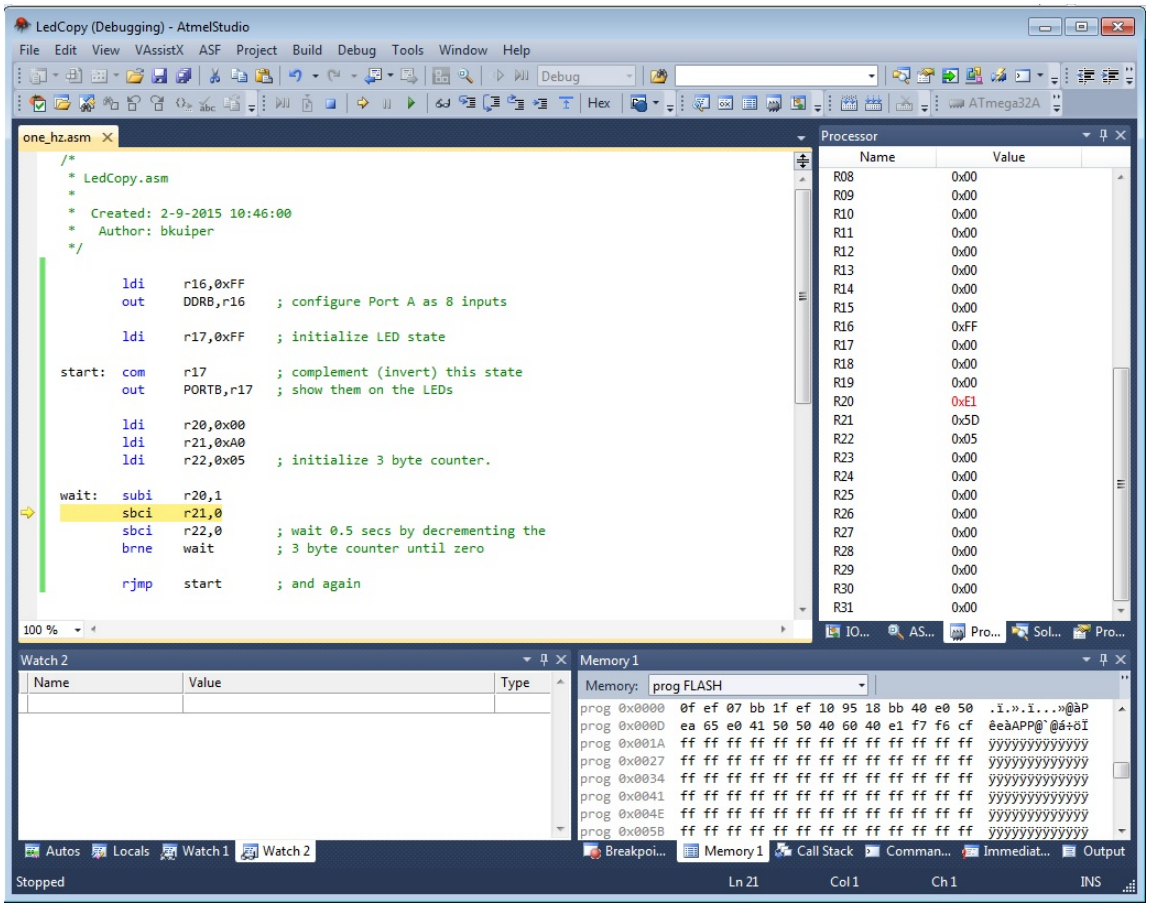
Start de debugger door op F5 of de Play knop te drukken. Je krijgt weer een foutmelding en daarna de mogelijkheid om een debugger/programmer te selecteren. Dit keer selecteren we niet JTAGICE, maar de simulator. Dit heeft als effect dat het geassembleerde programma niet op de ATMega32A via de JTAGICE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

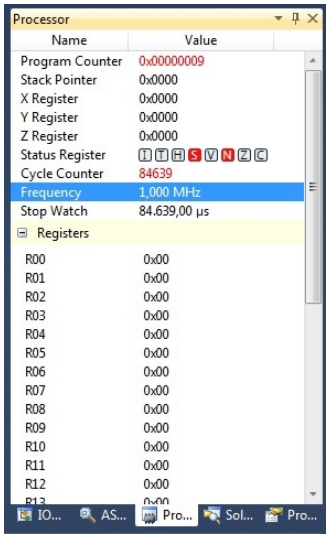
Zoals je aan de broncode kan zien, worden registers `R16`, `R17` en `R20` tot en met `R22` gebruikt. In het *Processor*-scherm kun je de inhoud van deze registers bekijken. Dit *Processor*-scherm kun je bekijken door in het rechter-scherm op het tabblad *Processor* te klikken. Zorg er met behulp van 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.




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):




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

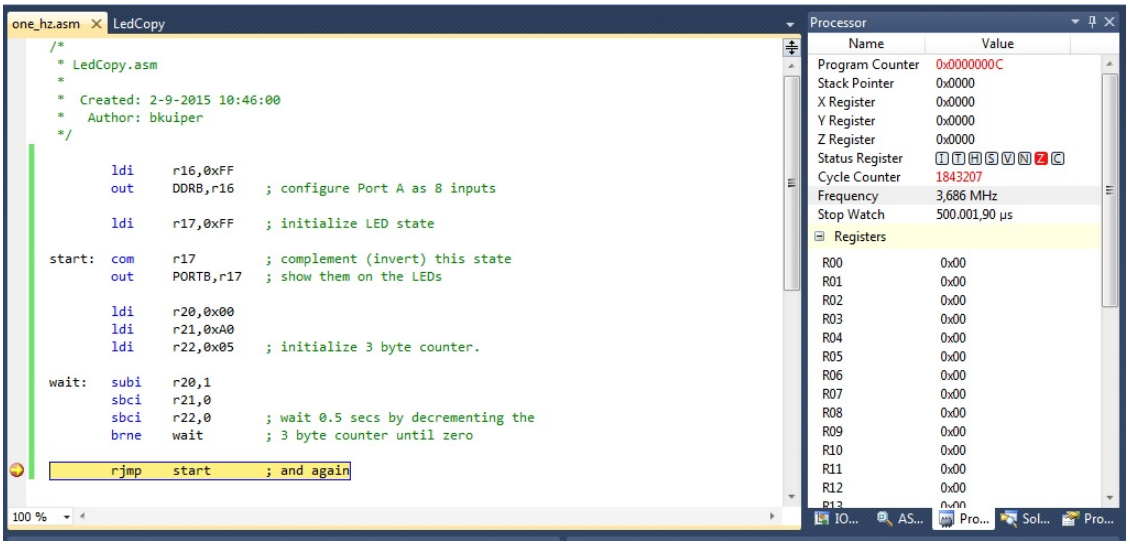
daarna de klokfrequentie. Je kunt vervolgens stap voor stap door de software lopen met F11. Na een tijdje zal het je opvallen dat je in een lus terecht bent gekomen en dat steeds dezelfde instructies worden doorlopen.

### 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!




Figuur 14: De simulator is gestopt bij het breakpoint.

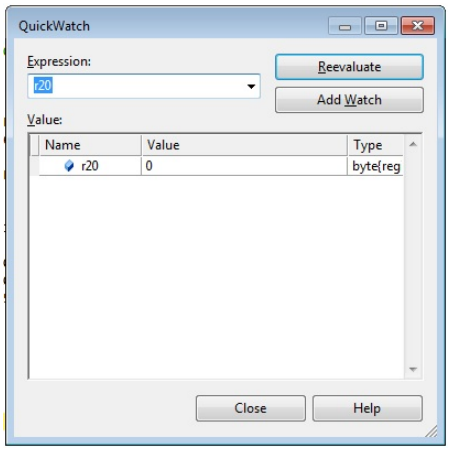
In het *Processor*-scherm kan je nu informatie vinden over de toestand van de processor. Bekijk vooral de *Cycle Counter* en *Stop Watch*.

### 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):




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.

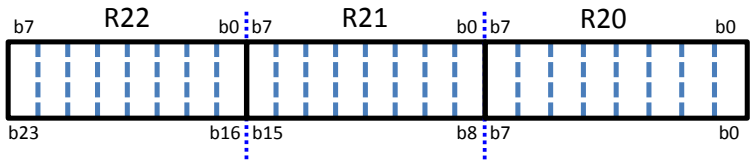
### 3.10 Werking *one_hz.asm*

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 aan[6] (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 (= 0x05A000[7]). `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

Tabel 1: AVR Studio sneltoetsen.

| Pictogram | Sneltoets | Werking/betekenis |
|---|---|---|
|  | 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 |
|  | Ctrl+F5 | Zet de debug-omgeving in pause (code wordt stilgelegd) |
|  | Ctrl+Shift+F5 | Sluit de debug-omgeving af |
|  | F9 | Toggle Breakpoint |
|  | F11 | Stapt steeds één instructie verder, volgt subroutines |
|  | F10 | Stapt steeds één instuctie verder, subroutines worden als één instructie behandeld |
|  | 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 updates in het I/O View zien |
|  | Shift+F5 | Reset de debug-omgeving en laat de processor opnieuw beginnen. |
|  | Shift+F9 | Opent het Add To Watch menu |
|  | Alt+1 | Schakelt het QuickWatch menu aan en uit |

## 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.