

AT90CAN32, hoofdstuk 7

E. Gernaat (ISBN 978-90-79302-06-2)

1 De debugger van AVR Studio 4

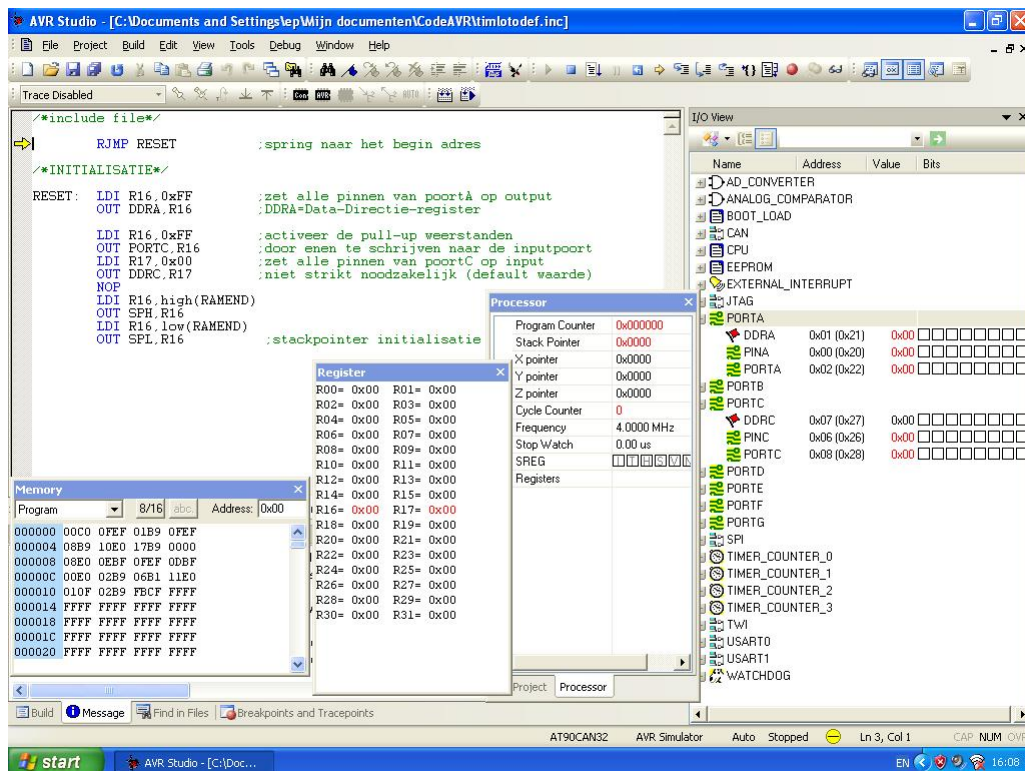
1.1 De werking

In zijn algemeenheid betekent 'debuggen' het opzoeken en herstellen van fouten in een software programma. Men onderscheidt wel syntax fouten en programmeer (denk)fouten. Wanneer we iets verkeerd hebben gespeld of we gebruiken een instructie verkeerd dan assembleert het programma niet en hebben we te maken met een zgn. syntax fout. Deze zijn meestal gemakkelijk te verhelpen. We kunnen, als we het niet onmiddellijk zien, gebruik maken van de aangemaakte .list file (aanvinken bij assembler options!) of we drukken op F1 als de cursor op een instructie staat. Moeilijker wordt het wanneer we een denkfout hebben gemaakt. Het programma werkt dan wel maar doet alleen niet wat we dachten dat het zou doen. Een debugger kan dan uitkomst brengen. AVR Studio 4 is een geïntegreerde programmeeromgeving (IDE) en bevat ook een debugger. Een debugger is een simulatieprogramma waarbij we het programmaverloop zichtbaar kunnen maken. We kunnen het programma stap voor stap doorlopen waarbij we na elke stap kunnen zien wat er in de diverse registers of geheugenplaatsen is gebeurd. Wanneer een programma in de debugger goed werkt is het nog geen 100% garantie dat het ook in de controller werkt. Educatief is een debugger ook waardevol omdat we het programmaverloop tot in details kunnen volgen. De debugger werkt met een aangemaakte .hex file. We moeten dus een gemaakt programma eerst assembleren. Als het assembleren niet lukt vanwege syntax fouten dan moeten die eerst worden verbeterd. Alleen programmeerfouten kunnen zichtbaar worden gemaakt. We nemen programma 3 als voorbeeld.

Ga naar 'Project', 'Open Project' en selecteer PROGRAM3. Assembleer PROGRAM3 door op 'Build' te klikken of druk op F7. Als alles goed gaat krijgen we onderin het scherm 'Assembly complete' te zien met '0 errors en '0 warnings'. De .hex file is dan aangemaakt.

Ga nu naar 'Debug' en klik op 'Start Debugging'. De hex file wordt dan geladen en we zien een gele pijl staan bij de eerste instructie van het programma. Ook zien we nog vier andere vensters op het scherm: I/O view, Processor, Register

en Memory. Mochten de vensters of één van de vensters ontbreken, ga dan naar 'View', 'Toolbars' en klik op I/O en/of processor. Vanuit 'View' kunnen we ook op Memory en/of Register klikken. In I/O view kunnen we weer op de '+' tekens klikken om bijv. de registers te bekijken die allemaal betrekking hebben op de CAN-communicatie. In dit programma hebben we te maken met Poort A en C dus we klikken op de '+' tekens van poort A en C. Het scherm zou er dan als fig. 1 kunnen uitzien. De gele pijl staat nu bij de 1e instructie Rjmp RESET.



Figuur 1: Een printscreen van de debug-mogelijkheden van Studio 4

Drukken we op F10 dan springen we naar het label RESET. Drukken we weer op F10 dan wordt R16 met 0xFF gevuld. Dit is te volgen in het register window. Stap nu maar door tot de eerste instructie van het hoofdprogramma 'LDI R16, 0x00'. Zet nu PINC in IO View op 7 door de laatste 3 hokjes zwart te maken en stap door tot ADD R16,R17. We zien nu dat in R16 0x07 en in R17 0x01 zit. Stap nu door tot 'Rjmp BEGIN' en zie dat via poort A 0x08 naar buiten wordt gestuurd. Dus wanneer de eerste drie schakelaars een 1 hebben gaat het 4e ledje (led3) aan. Nog een stap en we beginnen weer bij 'BEGIN'. Let ook op de verandering van de Program Counter. Het zal nu duidelijk op welke wijze debuggers werken en hoe we programma's op hun werking kunnen controleren. Het vraagt wel enige oefening om goed en vlot met debuggers te kunnen werken.

2 Vragen en opgaven

1. Waarvoor worden 'debuggers' gebruikt?
2. Waarom is deze debugger in feite een simulator?
3. Wat verstaat men onder een syntax fout?
4. Zoek eens uit wat het verschil is tussen 'step' en 'run'.
5. Loop nu zelf PROGRAM5 door met de debugger.