AVR ByteForth

versie 2.07

20 juli 2004

Proeflezers: Albert Nijhof, Paul Wiegmans, Ernst Kouwe & Ron Minke.

Inhoudsopgave

Lijst van figuren

1 Inleiding

1.1 De AVR-chip

ByteForth is een compilerend Forth systeem, dat toepassingen kan genereren voor de microprocessoren uit de AVR-serie van het fabrikaat Atmel. Ondersteund worden o.a. de AT90S2313, AT90S2323, ATtiny26, ATmega8, ATmega16, etc.
Deze processoren hebben 1 kBytes tot 128 kBytes Flash ROM, 0 bytes tot 4 kBytes RAM en 64 bytes tot 4 kBytes EEPROM. De RISC AVR processorkern heeft 118 of meer instructies en is in CMOS uitgevoerd. De processor zit in een 8 tot 64 pens plastic behuizing. De voedingsspanning ligt tussen de 1,8 en 6 Volt en de maximum klokfrequentie ligt tussen 4 MHz en 16 MHz afhankelijk van het gekozen type. Lees voor meer info het datasheet op bladzijde 124.

1.2 De AVR ByteForth omgeving

ByteForth is zoals de naam al zegt, een Forth-systeem met een celbreedte van 8-bits, i.p.v. de vaker voorkomende 16-bits of 32-bits versies. Het is een optimaliserende macrocompiler die draait boven op CHForth versie 1.2.5a op de PC. Het Forth-systeem bevat een AVR-assembler, disassembler en simulator (hierdoor kun je AVR-code op de PC uitvoeren), een ISP Flash-programmer voor bijna alle AVR's, dan is er nog het AT51-breadboard versie-2 waarop projecten met de AT90S2313 te testen en uit te voeren zijn. Er is een uitgebreide set macro's aanwezig, voor o.a. arrays, 16 bits variabelen, vlaggen, toegang tot de interne registers van de AVR, strings, gestructureerde controlestructuren en lussen. Het systeem bezit ook een bibliotheek met *geteste* functies, waarin o.a:

- RS232 aansturing, midi I/O, I2C en LCD.
- RC5 decoder.
- Getal conversie en rekenkundige routines.
- Matrix toetsenbord uitlezing.
- Eenvoudige muziek routines.
- Random getal genereren.
- Fout opvang routines CATCH en THROW.
- High-level en low-level interrupt gebruik.
- Multitasking, etc.

Er zijn ook enkele kant en klare toepassingen toegevoegd waarvan veel te leren is. De omgeving is vanaf begin 2000 in gebruik, en er zijn al aardig wat toepassingen mee ontwikkeld, waaronder de Ushi robot, zie bladzijde 118.

1.3 Inhoud van het AVR ByteForth pakket

AVR ByteForth wordt uitsluitend als compleet pakket geleverd. Er zijn twee versies van het pakket, versie (a) en (b). Versie (a) bevat het volgende:

AT90S2313	Processor	2 stuks
AT51 versie-2	Breadboardprint	1 stuks
Programmeerdongle	ISP adapterprint	1 stuks
AVR ByteForth 2.07	Software op disk	1 stuks
AVR ByteForth handboek	Deze handleiding	1 stuks

Versie (b) bevat connectoren met dezelfde layout als op het ATS-bord. Zodoende past het LED-printje, de schakelaar-print en de LCD-print direct op dit bordje. Versie (b) bevat het volgende:

ATmega8515	Processor	1 stuks
AT8252	Breadboardprint	1 stuks
Programmeerdongle	ISP adapterprint	1 stuks
AVR ByteForth 2.07	Software op disk	1 stuks
AVR ByteForth handboek	Deze handleiding	1 stuks

Een demoversie van de software is op onze website te vinden.

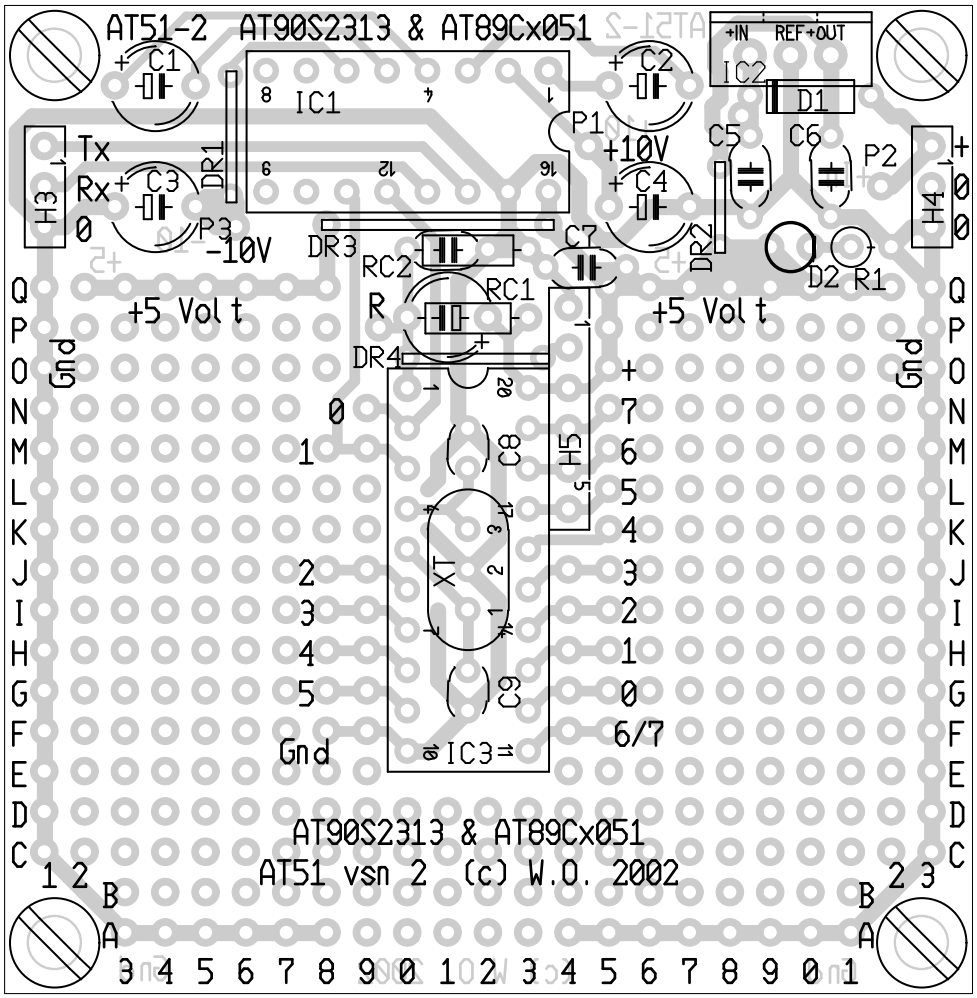
2 Installatie

2.1 Benodigdheden

Om een AVR ByteForth-systeem samen te stellen hebben we de volgende onderdelen nodig:

- PC of compatible computer.
- ISP Flash en EEPROM programmeer adapter (dongle) [in pakket].
- Een 9 Volt gelijkspanningsvoeding van 100 mA (een kleine ongestabiliseerde 9 tot 12 Volt adapter is meestal voldoende).
- AT51 versie-2 breadboard voor het testen/uitvoeren van een toepassing [in pakket]. De starterkits STK200(+) van Kanda en de STK500 van Atmel voldoen ook prima.

Begin met het aansluiten van de ISP (In System Programmer) adapter op de printerpoort, doe dit eerst op PRN1, later kan het veranderd worden. Deze ISP adapter is technisch gelijk aan die voor de STK200(+) van Kanda systems (wij gebruiken echter een andere printsteker). Meer info op de bladzijden 102, 103 en 119.



Figuur 2.1: Tekening van de AT51-2 print

2.2 Software

Een AVR ByteForth-systeem op de PC bevat de volgende componenten:

- Optimaliserende crosscompiler die code in een buffer genereert.
- AVR software simulator die de gegenereerde code in de buffer op de PC uit kan voeren.
- Configureerbare tracer met breekpunten.
- Een AVR assembler met gestructureerde controlestructuren.
- AVR disassembler die code in de buffer leesbaar op het scherm kan afbeelden.
- ISP flash programmer die code in de buffer overzet naar de AVR (met slechts zes draadjes).
- Enkele kant en klare toepassingen en een bibliotheek met geteste software.

2.3 Installeren van de software

1) Software van de AVR ByteForth omgeving kan geïnstalleerd worden van de meegeleverde floppy disk. Stop de disk in de PC en start vanuit DOS of een Windows DOS-box de software. Bijvoorbeeld `A:SETUP C: <enter>` en even later staat de software gebruiksklaar voor u op schijf C: in de directory AVRF. Door de batchfile AVRF.BAT wordt ByteForth correct gestart. Plaats deze batchfile b.v. in uw BATCH directory of op een andere plaats waar hij gemakkelijk gevonden kan worden.

2) Als de AVR ByteForth software klaar staat en de programmer aangesloten en juist geconfigureerd is, zullen zij zich melden:

```
AVR ByteForth crosscompiler vsn 2.07 (c) W.O. 2004
ISP Flashprogrammer versie 1.32 (c) W.O. 2000-2004
```
etc.

Daarna kun je enkele keren `<enter>` geven, ByteForth reageert dan met OK. Tik nu `COLD <enter>` in, ByteForth zal dan nogmaals reageren met zijn startup melding.

2.4 Configureren van de software

AVR ByteForth en de programmer zijn nu geïnstalleerd en werken. Alle basisinstellingen van ByteForth zijn te veranderen door de file AVRF.CFG te editen. Als editor staat de publiek domein editor SZ van Tom Zimmer opgegeven, maar je kunt natuurlijk je favoriete DOS-editor daarvoor in de plaats zetten. De file is opgesplitst in vijf delen:

1) De paden naar de bibliotheek- en helpfiles.
2) De strings voor de standaard file header, zie ook `PROJECT`.
3) De ISP-klokpulsvertraging en de gewenste werkdirectory.
4) Gewenste basisinstellingen voor ISP-poort en tracer.
5) Je favoriete DOS-editor, DOS-shell en andere DOS-hulpfiles.

```
\ Configuratie file voor AVR ByteForth 2.07

\ Defineer paden naar bibliotheek en hulp files
S" C:\AVRF\LIB" LIBPATH PLACE
S" C:\AVRF\HELP" HELPPATH PLACE

\ De drie strings proj$, cat$ en creat$ mogen hier worden aangepast
\ Maximum lengte: 54 karakters.
S" AVR ByteForth, een pub. domein Forth voor de AVR serie" PROJ$   PLACE
S" Applicatie, afmeting: .... bytes."                      CAT$    PLACE
S" Willem Ouwerkerk"                                       CREAT$  PLACE
\  1--------10--------20--------30--------40--------50---
```

```
100 SET-PAUSE           \ Zet ISP klokpuls vertraging

\ Zet pad naar uw AVR ByteForth werk directory
  SILENT  CD C:\AVRF\WORK  VIDEO

\ Zet basis instellingen van AVR ByteForth
  PRN1              ( Gebruik PRN1 of PRN2 of PRN3 of PRN4 )
\ ECHO-OFF          ( Aan is default )
\ PORTS-OFF         ( Aan is default )
\ STEP-ON           ( Uit is default )

\ Voeg je eigen favoriete programma's toe

DEBUG DEFINITIONS

  S" sz "         SET-EDITOR      \ Zet editor, vergeet de spatie niet!
  S" vc"          SET-SHELL       \ Zet dos shell

\ Programma naam .. AVR ByteForth naam .......

  S" hp "         DOS: HP         \ W.O's HP PCL print programma
  S" gloss "      DOS: GLOSS      \ L. Benschop's glossary generator
\ S" list "       DOS: L          \ View een file, (C) Vernon D. Buerg
\ S" grep "       DOS: GREP       \ Gebruik een tekst zoek programma
```

2.5 ByteForth functietoetsen

De actieve toetscombinaties en functietoetsen van ByteForth zijn:

- F1 Hulpfile bij ByteForth commandline editor.
- F2 Online ByteForth help functie.
- F3 Toon actuele directory inhoud.
- F4 Start tekstverwerker met de actuele tekstfile.
- F5 Compileer de actuele tekstfile.
- F6 Ga naar een operating system shell.
- F7 Selecteer en/of toon een directory.
- F8 Start tekstverwerker op de laatste fout.
- F9 Open en sluit een logfile.
- Alt-X Sluit alles af, ga terug naar DOS/Windows.

2.6 Adres van de HCC Forth gg

Stuur voor vragen en verdere informatie over AVR of 8051 ByteForth een email met een duidelijke beschrijving van het probleem. Vergeet niet de sourcecode toe te voegen.

Adres:	HCC Forth gg p/a Boulevard Heuvelink 126 6828 KW Arnhem
Tel:	026-4431305
Email:	ByteForth@hccnet.nl
Homepage:	http://www.forth.hccnet.nl
Willem's homepage:	http://home.hccnet.nl/willem.ouwerkerk/pr-bytef.htm

3 Spoedcursus

Als je niet bekend bent met AVR ByteForth, dan wordt je uitgenodigd om alles van de linker kolom op de volgende bladzijden in te tikken. Deze korte cursus neemt je mee door zo'n beetje alle onderdelen van AVR ByteForth, tot zelfs het maken van je eerste toepassingen.
Volg de tekst en type steeds de Forth code links op de bladzijde in ByteForth in. `<cr>` betekent druk de `<enter>`-toets in.
Het onderscheid tussen hoofd- en kleine letters is niet van belang. Spaties zijn heel erg belangrijk. Alles door een spatie gescheiden is een getal of een Forth 'woord' (zie 't als een subroutine). ByteForth reageert met OK na elke goed uitgevoerde regel. Als het een erge rommel wordt, of als je de draad kwijt bent, doe dat stuk dan opnieuw en let goed op wat je typt. Lees de opmerkingen rechts op de bladzijde goed door en LET BOVENAL GOED OP HET SCHERM!

Voor beginners in Forth is aan het begin van elk hoofdstuk een korte introductie opgenomen. Gevorderde Forth programmeurs hoeven zich slechts te concentreren op de ByteForth 'eigenaardigheden'.

Jij moet opletten wat er gebeurt.

3.1 Het begin

Forth is een stackgeorienteerde programmeertaal. Daardoor ziet alles er een beetje anders uit dan je misschien gewend bent in bijvoorbeeld BASIC. Het doet sterk denken aan de HP-rekenmachines van vroeger. Om 7 + 5 uit te rekenen moest je intoetsen 7 `<Enter>` 5 `<+>`. Het antwoord verscheen dan op het display. Je plaatst eerst het getal 7 op de stack (ned. stapel), dan de 5 en vervolgens geef je aan welke bewerking op die twee getallen uitgevoerd moet worden. Dit wordt ook wel de Reverse Polish Notation (RPN) genoemd.

Type in:	Uitleg en opdrachten:
`<cr>`	Er hoort nu OK te verschijnen.
`4 <cr>`	OK verschijnt en 4 is op de stack geplaatst. ByteForth gebruikt 8-bit integers.
`. <cr>`	`.` drukt de top van de stack af.
`. <cr>`	Hee, de stack is niet bodemloos.
`1 2 <cr>`	Plaats twee getallen op de stack.
`.s <cr>`	Druk de stack inhoud af zonder deze te veranderen.
`+ . <cr>`	In RPN komen eerst de getallen en dan de operatie.
`130 10 - . <cr>`	Nog wat meer RPN rekenen.
`5 dup <cr>`	Probeer `DUP` uit.
`* . <cr>`	Er zijn meer woorden om de stack te manipuleren. Forth werkt over het algemeen zo: eerst de data, dan de operator (actie).

3.2 Ietsje verder

In (Byte)Forth kun je makkelijk overgaan naar een andere getalbasis met b.v. `HEX` of `DECIMAL`. D.m.v. voorvoegsels aan een getal kan eenvoudig een getal in een andere getalbasis gebruikt worden. De karakters zijn: # = decimaal, $ = hexadecimaal, % = binair. Hieronder enkele voorbeelden.

`100 ms <cr>`	Wacht 100 milliseconden.
`12 <cr>`	Zet 12 op de stack.
`hex <cr>`	Maak de getalbasis hexadecimaal.
`. <cr>`	C hexadecimaal is 12 decimaal.
`10 decimal . <cr>`	Geeft dit het antwoord dat je verwacht had?
`$10  250 ms  . <cr>`	Zet hex 10 op de stack wacht even en druk het getal decimaal af.
`%1001 . <cr>`	Zet binair 1001 op de stack en druk decimaal af.
`10 12 *  50 -  2/ . <cr>`	Je gaat nu wat uitgebreider rekenen bereken eerst 10*12, trek er daarna 50 van af en deel het tenslotte door 2.
`page <cr>`	Maak het scherm (schoon wanneer je wilt). 8 bit integers met teken (signed): -127 ... +127 en zonder teken (unsigned): 0 ... 255. De gekozen woorden bepalen hoe de getallen opgevat worden.
`1000. 300. d+ d. <cr>`	Je kunt ook met 16 bit integers werken. Gebruik b.v. `HELP D+` om meer info te krijgen.
`&A . <cr>`	Zet ASCII-waarde van 'A' op de stack.
`^C . <cr>`	Zet CTRL-waarde van 'C' op de stack.

3.3 Nieuwe woorden maken

Je hebt nu Forth gebruikt als rekenmachine: hij voert iedere opdracht meteen voor je uit. Dit heeft dus nog niets met programmeren te maken. Je gaat nu nieuwe woorden maken die tijdens het intypen nog niets doen. Pas als zo'n woord wordt aangeroepen voert het wat uit.

`verhoog <cr>`	Oeps foutmelding. Forth is een verzameling woorden. En `verhoog` is geen Forth woord.
`: verhoog  3 + ; <cr>`	Maar het kan er een worden. De definitie van een nieuw woord begint met de dubbele punt `:` gevolgd door de naam van dat woord.
`10 verhoog . <cr>`	Al het andere tot de punt-comma is de code die uitgevoerd zal worden als het nieuwe woord wordt uitgevoerd.

`: tellus <cr>`
`  100  5 0 <cr>`
`  do verhoog loop ; <cr>`
	Het woord kan weer in andere definities gebruikt worden.

`tellus . <cr>`	Een gecompileerd woord wordt pas uitgevoerd als het later wordt aangeroepen. Je hebt `tellus` gemaakt en `verhoog` er in gecompileerd. Daarna heb je `tellus` uitgevoerd.

3.4 Variabelen, etc.

De meeste programmeertalen gaan uit van een computer met voldoende RAM-geheugen. Bij microcontrollers is dat niet het geval. Je moet vaak woekeren met het gebruik van RAM.
Een stack in plaats van veel verschillende variabelen bezuinigt ernorm op het gebruik van RAM. Toch ontkom je niet altijd aan het gebruik van variabelen. Maar minimaliseer het gebruik ervan!
Variabelen zijn goed bruikbaar voor communicatie tussen parallel draaiende programma's. Denk hierbij aan interrupts of meerdere programma's die tegelijkertijd afgewerkt worden (multitasking). Ook wanneer veel soortgelijke data afgehandeld wordt kunnen array's van variabelen uitkomst bieden.

`Empty <cr>`	Je ruimt eerst de voorgaande probeersels op.
`variable pils <cr>`	`variable` wordt gebruikt om globale variabelen te definiëren. Een variabele laat zijn adres op de stack achter als hij uitgevoerd wordt.
`5 pils ! <cr>`	Zet 5 (`!` = store) in de 8 bit `pils` variabele.
`pils @ . <cr>`	Lees de inhoud (`@` = fetch) van de 8 bit `pils` variabele.
`pils 2constant gluur <cr>`	`gluur` geeft het adres van `pils` op de stack.
`gluur @ . <cr>`	Ook met `gluur @` krijg je de inhoud van `pils` te zien.
`10 +to pils <cr>`	In ByteForth zijn de variabelen ook via zogenaamde prefixen toegankelijk. In dit geval is het aantal pilsjes met 10 toegenomen.
`pils @ . <cr>`	Zie maar.
`clear pils <cr>`	Deze techniek levert zeer doeltreffende code op en maakt het programma beter leesbaar.
`from pils . <cr>`	Ook op deze manier kan je een variable uitlezen probeer maar, de pilsjes zijn inderdaad op.

```
.. TO PILS      .. PILS !
FROM PILS       PILS @
.. +TO PILS     .. PILS +!
```

In ByteForth zijn alle getallen integers. Data kan zowel een 8- of 16-bits getal/adres zijn en moet voor gebruik gedefinieerd worden, zoals gewoon is in Forth. In ByteForth zijn o.a. de volgende datastructuren opgenomen:

VARIABLE	8-bits variabele	2VARIABLE	16-bits variabele
VARIABLES	8-bits array	2VARIABLES	16-bits array
VALUE	8-bits TO-variabele	REGISTER	8-bits register-variabele

Lokale variabelen zijn alleen binnen colon-definities toegestaan. Meer daarover in op bladzijde 13. Gebruik `HELP 'naam' <cr>` om meer uitleg over een Forth woord of begrip te krijgen.

3.5 Controlestructuren

Hier wordt het gebruik van enkele Forth 'control structures' gedemonstreerd. Ik doe dat met behulp van strings. Bij deze voorbeelden is ook het stackgedrag van de woorden gedocumenteerd, `( -- )` betekent dat het woord niets opneemt en achterlaat.

```
atom inline$
```
Om `."` te gebruiken moet je de assembler macro `inline$` importeren. Het woord `atom` verzorgd dat importeren. Meer info op bladzijde 22.

```
: .pils  ( -- ) <cr>
```
Je maakt een woord om te controleren hoeveel bier er nog is, de naam is `.pils` en de commentaarhaakjes geven aan dat `.pils` niets van de stack nodig heeft en ook niets achterlaat.

```
  from pils ?dup if <cr>
```
Er wordt getest: is er nog pils?

```
    ." nog " . <cr>
    ." stuks " <cr>
```
Zo ja, toon het aantal volle pilsjes.

```
  else <cr>
    ." de pils is op " <cr>
```
Zo nee, druk deze tekst af.

```
  then ; <cr>
```

Je gaat wat vrienden uitnodigen voor een feestje.

```
.pils <cr>
```
Voor feestje gebruik je een `begin until` lus met een daarin (geneste) `IF-THEN`.

```
: feestje  ( -- ) <cr>
```
Ook `feestje` is stack neutraal.

```
  24 to pils <cr>
```
Je koopt vantevoren een nieuwe krat pils.

```
  begin <cr>
```
Begin een lus.

```
    cr ." Pilsje J/N " <cr>
```
Druk een vragende tekst af.

```
    key &J = if <cr>
```
Als de hoofdletter `J` ingedrukt wordt...

```
      -1 +to pils <cr>
```
neem je een biertje uit de krat.

```
      .pils <cr>
```
Hoeveel zijn er nu nog over..

```
    then <cr>
  from pils 0= until <cr>
```
`Feestje` is afgelopen als de pilsjes op zijn.

```
  cr ." Tot ziens " ; <cr>
```
Tot een volgende keer dan maar.

ByteForth kent ook: `BEGIN WHILE REPEAT`, `CASE`, `FOR NEXT`, `SELECT`, `AHEAD` en `ENTRY`.

3.6 De AVR assembler

Voor tijdkritische stukken code is het handig om terug te kunnen vallen op de machinetaal van de processor. Net als veel andere Forth systemen heeft ByteForth daarom een assembler, zie bladzijde 95. Hier een assembler voorbeeld (dat je niet gelijk hoeft te begrijpen).

```
code 6+ ( x1 -- x2 ) <cr>
```
De AVR assembler is beschikbaar in `code` definities. De naam van het woord is `6+` en het telt 6 op bij de top van de stack. Een codedefinitie eindigt altijd met `ret,` omdat ByteForth subroutinebedraad is.

```
  r16 x+ ld, <cr>
  r16 6 addi, <cr>
  -x r16 st, <cr>
  ret, <cr>
end-code <cr>
```
`end-code` sluit de codedefinitie af.

`5 6+ . <cr>`	Probeer het code woord.

`: vul ( -- u ) <cr>`
`from pils <cr>`
`6+ to pils <cr>`
`from pils ; <cr>`
	Je vult met het woord `vul` de voorraad pilsen weer bij. Tenslotte lees je het aantal pilsjes met `from` weer uit.

`vul . <cr>`	Maar het feestje is afgelopen. Ga je al dat bier alleen opdrinken?
`vul . <cr>`	

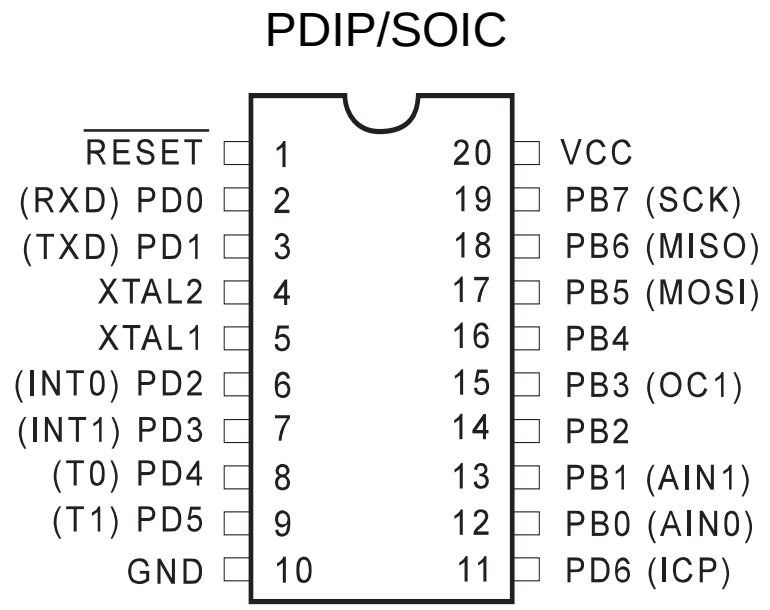
3.7 Special Function Registers (SFR's)

Het definiërend woord SFR regelt de toegang tot de 'I/O-space' van de AVR-microcontrollers. Alle speciale interne hardware van de AVR-chips kan hiermee benaderd worden. Op een AT90S2313 vindt je deze functies: twee timers, PulsBreedteModulatie, EEPROM, I/O-poorten, watchdog, uart (RS232), comparator. De andere chips uit de AVR-serie hebben soms meer timers, ADC, SPI-interface, I2C, etc. Zie bladzijde 92 voor een beschrijving.

`$18 sfr poortb <cr>`	Definiëer toegang tot PORTB van de processor, dit zijn de pennen PB0 t/m PB7 van de AT90S2313.
`-1 setdir poortb <cr>`	Maak van PORTB een uitgang (zie pen configuratie).
`poortb . <cr>`	Lees de toestand van poort-B.
`set poortb <cr>`	Maak alle uitgangen van poort-B hoog.
`poortb . <cr>`	Zie je wat er gebeurd is?
`1 to poortb <cr>`	Maak nu alleen bit 0 van poort-B hoog. Het gebeurt ook nog !
`poortb . <cr>`	Probeer maar.
`$18 1 bit-sfr uitgang <cr>`	Definiëer toegang tot bit 1 van poort-B.
`set uitgang <cr>` `poortb . <cr>`	Maak `uitgang` nu hoog.
`clear uitgang <cr>`	Maak `uitgang` weer laag.
`$18 7 bit-sfr ingang <cr>`	Definieer toegang tot bit 7 van poort-B.
`0 setdir ingang <cr>`	Maak alleen van bit 7 een ingang.
`set ingang <cr>`	Om een bit als ingang te kunnen gebruiken met pullup moet je eerst dit bit hoog maken.
`from ingang . <cr>`	Lees `ingang`, ingangen bij de AVR's zijn altijd laag actief! `from` heeft een speciaal gedrag bij een poort uitvoer register.
`help sfr <cr>`	Zie de beschrijving bij SFR.

3.8 Penconfiguratie AT90S2313

Een completer datasheet van de AT90S2313 vind je achter in dit boekwerk op bladzijde 124. Voor een volledig datasheet moet je naar de website van ATMEL gaan, de link daarvan vind je op bladzijde 119.



Figuur 3.1: Pen configuratie van AT90S2313

3.9 Het is een crosscompiler

De ByteForth compiler is gebouwd als een cross-compiler, dat wil zeggen dat de software op een ander platform, b.v. een PC (ook **host** genaamd), gemaakt wordt. De gegenereerde code draait niet op de PC maar op een andere processor (**target**), b.v. de AT90S2313. Meer info op bladzijde 18. Er is sprake van een **host** en een **target** (doel).

`words <cr>`	Laat alle woorden in het werkgebied zien. Als laatste zie je `uitgang` en `ingang`. Deze woorden hebben we in de vorige paragraaf gemaakt en ze staan in de `Forth` woordenlijst.
`>host <cr>`	Je schakelt nu naar het gewone Forth systeem, de **host** op de PC.
`uitgang . <cr>`	De woorden `uitgang` en `ingang` zijn hier niet te vinden.
`>cross <cr>`	Terug naar ByteForth.
`uitgang . <cr>`	En... de woorden zijn er weer.
`words <cr>`	Probeer maar.

`atom + <cr>`	Importeer de + macro in de woordenlijst.
`12 13 + . <cr>`	Zet getallen op de stack en voer + uit. Daarna wordt het resultaat door `.` getoond. De stack wordt leeg achter gelaten!
`: telop  12 13 + ; <cr>`	Elk nieuw woord kun je natuurlijk ook testen.
`telop . <cr>`	Zie je?

3.10 Gebruik van de decompiler

`see vul <cr>`	Bijna alles in ByteForth is machinecode, het
`see telop <cr>`	decompileren levert daarom voornamelijk een
`see 6+ <cr>`	lijst opcode's op met af een toe een RCALL of RJMP naar een ander woord. Druk op de spatiebalk voor de volgende opcode en een andere toets om te stoppen.

3.11 Hoe werkt de simulator

Om ByteForth op een 'normale' Forth te laten lijken is er een simulator toegevoegd. Code die eigenlijk voor een AVR-cpu is kan zo op de PC uitgeprobeerd worden. Je merkt nauwelijks verschil. De simulator kun je ook gebruiken als tracer om bugs te vinden in je code, meer over de simulator op bladzijde 25.

`tracer-on <cr>`	Je zet de tracer visueel aan.
`.tracer <cr>`	Toon de instellingen van de tracer.
`telop . <cr>`	Zie je de tracer lopen of gaat het te snel?
`step-on <cr>`	Stap voor stap mode aan.
`telop . <cr>`	Voer code uit in de stap voor stap mode, druk op de spatiebalk om de volgende opcode uit te laten voeren.
`step-off <cr>`	Stap voor stap mode weer uit.
`1 +to poortb MANY <cr>`	Zie je de bits op poortb veranderen? Druk op een toets om daarmee te stoppen.
`tracer-off <cr>`	De tracer weer uit en....
`empty <cr>`	Ruim de rommel tenslotte op.

3.12 Het maken van een toepassing

Microcontrollers worden vooral gebruikt om hardware mee te besturen. Vaak is er geen toetsenbord of beeldscherm aangesloten. Maar met bijvoorbeeld acht leds heb je al een primitieve "monitor". Hiermee kun je heel goed vaststellen of een programma werkt. De leds worden aangesloten op PORTB van de AT90S2313, zie ook de eerste ontwerpen in het 'Egelwerkboek. De pennen van deze poort kunnen heel eenvoudig softwarematig aan- of uitgezet worden. Deze poort moet in (Byte)Forth vooraf gedefinieerd worden met SFR, Special Function Register.

`empty <cr>`	Ruim alle rommel op.
`90S2313 <cr>`	Gebruik memory map voor een AT90S2313.
`needs target <cr>`	Voeg labels voor de AT90S2313 toe.
	Maak doelcode voor deze processor.
`portb sfr uitgang <cr>`	Gebruik Poort-B als uitgang.
`: teller ( -- ) <cr>`	De toepassing ...
`setup-byteforth <cr>`	Installeer de Forth machine (verplichte kost).
`-1 setdir uitgang <cr>`	Zet het richtingsregister van Poort-B als uitgang.
`clear uitgang <cr>`	Zet de uitgangen op nul.
`begin <cr>`	Start de hoofdlus, waarin de uitgang
`1 +to uitgang <cr>`	als binaire teller gebruikt wordt
`250 ms <cr>`	en elke 250 millisec. verhoogd wordt.
`again ; main <cr>`	En dat eindeloos lang.
	`MAIN` vist het adres van de toepassing op en installeert die in de reset vector. Controleer of de ISP-kabel aangesloten is op het STK200(+) bord, een AT51 versie-2 bord of AT8252 bord.
`e p v <cr>`	Wis hem eerst `e`, dan het `p` (programmeer) en het `v` (verifieer) commando. De processor is nu klaar en de toepassing loopt al !!

3.13 Programma's invoeren

Type: `EDIT DEMO <cr>`. Je komt terecht in de editor, die de file DEMO.FRT aanmaakt. Je bent nu in de editor. Druk op 'F1' voor uitleg over de editor functies. Type nu de code van de vorige paragraaf in, behalve de laatste regel. Met 'F10' save je de file en kom je terug in ByteForth.

3.14 Programma's compileren (laden)

Type: `IN DEMO <cr>` De file DEMO.FRT wordt nu door ByteForth regel voor regel vertaald (gecompileerd). Tenminste als er geen typefouten zijn gemaakt. Nu zijn alle in de file opgenomen woorden voor je beschikbaar. Speel er nog wat mee, en ga dan door naar het volgende deel. Heb je echter wel fouten gemaakt, dan stopt het compileren op de eerste fout. Als je de NE.COM of SZ.COM editor in gebruik hebt, kun je d.m.v. `WHAT` de editor starten. De cursor staat dan op de regel waar de fout is.

3.15 Locale variabelen

Om gedoe op de stack te vermijden kunnen locale-variabelen toegepast worden. Getallen worden van de stack gehaald en voorzien van een naam die alleen binnen één colon-definitie bruikbaar is. Ze worden op dezelfde manier gehanteerd als VALUE's.

```
: som1 ( a b c -- d ) <cr>
  locals| c b a | <cr>

  a b *  c -  2/ ; <cr>

10 12 50 som1 . <cr>

: som2 ( a b c -- d ) <cr>
  >r  *  r> -  2/ ; <cr>

10 12 50 som2 . <cr>
see som1 <cr>

see som2 <cr>
```

Er worden drie getallen van de stack gehaald.

Het bovenste getal wordt aan de eerste naam toegekend, etc.

Je hoeft niet meer met de stack te schuiven om de rekensom uit te werken.

Zoals je ziet is het resultaat hetzelfde als bij de som aan begin van de cursus. Je hoeft de som nu niet steeds uit te schrijven. Onthoud wel dat locale variabelen vaak wat meer ruimte gebruiken dan bij gebruik van de stack.

Dezelfde berekening maar nu via de stack.

Probeer maar uit.

Bekijk hoeveel code er voor zowel `som1` als `som2` gegenereerd is. Je hoeft geen assembler te kennen om met ByteForth te werken.

Waar de `som1`, de versie met locals 41 opcodes nodig heeft, gebruikt de versie met de stack er slechts 22. Dat is bijna de helft kleiner. Dat neemt niet weg dat locale variabelen voor ingewikkelde woorden handig kunnen zijn.

3.16 Nieuwe datastructuren

Gevorderde Forth gebruikers maken hun toepassingsgerichte datastructuren op maat. Daarvoor gebruiken ze CREATE en DOES>. Hieronder twee voorbeelden van datastructuren in ROM en RAM. In AVR ByteForth gebruik je een speciale colon-definitie, eentje beginnend met een dubbele dubbelepunt, om een nieuwe datastructuur te maken. Zie ook bladzijde 21.

```
ram <cr>
:: vars ( aantal -- ) <cr>
  create <cr>

    allot align <cr>

  does> d+ ; <cr>

10. vars lijstje <cr>
12 9. lijstje ! <cr>
100 0. lijstje ! <cr>
0. lijstje @ . <cr>
9. lijstje @ . <cr>
```

Een datastructuur die in RAM werkt.

De naam ervan is VARS.

Het woord CREATE zorgt dat de nieuwe structuur een naam krijgt.

En ALLOT reserveert een rij bytes in RAM, ALIGN regelt het afronden van het geheugenblok zodat de cpu niet kan struikelen.

DOES> zet het adres van de rij RAM bytes op de stack en D+ telt de opgegeven index erbij op. Adressen zijn hier 16-bits getallen, D+ is een 16-bits optelling, de index is daarom ook 16-bits!!

Je maakt ARRAY met 10 bytes opslagruimte.

Zet 12 op positie 9 in het lijstje,

Zet 100 op positie 0.

Lees positie 0 terug, klopt het?

Lees ook positie 9 terug, klopt die ook? De telling begint bij nul. Het gaat om een offset!

```
rom <cr>                          Nu maak je een datastructuur in ROM.
:: exec <cr>                      Een executietabel genaamd EXEC.
  create  ( -- ) <cr>             Op de stack verwacht die niets.
  does>  ( n -- i*x ) <cr>        Bij uitvoering wordt op de stack het nummer van
                                  het gewenste token 'n' verwacht.
    rot m+  2rom@ <cr>            Na manipulatie en een berekening wordt het juiste
                                  token uit de tabel opgevist door
    execute ; <cr>                2ROM@ en vervolgens uitgevoerd door EXECUTE.
                                  Let op er is geen enkele beveiliging aangebracht!
: aap   10 ; <cr>                 Vier programma's voor in de executietabel.
: noot  20 ; <cr>
: mies  30 ; <cr>
: wim   40 ; <cr>

exec ina <cr>                     Maak een executietabel met de naam ina.
' aap d,   ' noot d, <cr>         Zet de gewenste tokens in de executietabel.
' mies d,  ' wim d, <cr>

0 ina . <cr>                      Het eerste token wordt uitgevoerd.
3 ina . <cr>                      Het vierde token wordt uitgevoerd.
5 ina . <cr>                      Omdat 5 geen geldig token opleverd, wordt een on-
                                  geldig token uitgevoerd. Gesnapt? Maak je geen
                                  zorgen als dat nog niet zo is, CREATE DOES> is
                                  Forth voor gevorderden.
```

3.17 Een toepassing met code en interrupts

```
empty <cr>                        Ruim eerst alle voorgaande rommel op. Je start
                                  de file editor nu op een speciale manier.
project teller <cr>               Er wordt een file gemaakt met een standaard tekst-
                                  blok er in. De strings daarvan kun je aanpassen
                                  in de file AVRF.CFG Type het nu volgende pro-
                                  gramma in.

register teller                   Maak een 8 bits teller register-variabele.
```

```
code tel ( -- )
  r16 push,
  r17 push,
  r17 sreg in,
  r16 -156 ldi,
  tcnt0 r16 out,
  adr teller inc,
  sreg r17 out,
  r17 pop,
  r16 pop,
  reti,
end-code  t0-overflow
```

Definiëer nu de interrupt routine.
Bewaar de gebruikte registers eerst.
Bewaar het statusregister in R17. Elke veertig millisec. wordt de variabele `teller` verhoogd.
Herstel het statusregister.
Herstel de gebruikte registers weer.
Omdat dit geen gewone subroutine is, maar een interrupt, eindigt hij niet met een `ret,` maar met een `reti,` instructie. Het commando `t0-overflow` zet `tel` in de gewenste interrupt vector van de AVR.

```
code setup-tel ( -- )
  adr teller clr,
  r16 -156 ldi,
  tcnt0 r16 out,
  r16 5 ldi,
  tccr0 r16 out,
  r16 2 ldi,
  timsk r16 out,
  sei,
  ret,
end-code
```

Maak een definitie die timer-0 als klok klaarzet die elke 40 millisec. afloopt.
Zet timer-0 klaar
Timer-0 aan met een prescaler van 1024.
Timer-0 interrupt aan.
Interrupt mechanisme aan.

```
portb sfr leds
```

Uitvoer naar leds op Poort-B

```
:main ( -- )
  -1 setdir leds
  setup-tel
  begin
    teller
    invert to leds
  again ;
```

Start hoofdprogramma
Zet het richtingsregister van Poort-B als uitgang.
Initialiseer tel interrupt
Begin van eindeloze lus
Lees `teller` uit
Keer om en toon op de leds
Terug naar begin

Hier kun je de editor verlaten.

`IN <cr>`	Het woord `IN` laad (include) altijd de laatst gebruikte file. Dat maakt ontwikkelen ietsje makkelijker.
`e p v <cr>`	Zet interrupt voorbeeld in een AT90S2313 chip op een AT51 versie-2 of STK200(+) bord en lopen maar. Zie de tellerstand veranderen.
`empty <cr>`	Ruim ook dit lesmateriaal weer op.

Ook 'high-level' interrupts zijn toegestaan in ByteForth. Voor toepassing daarvan zijn enkele speciale commando's opgenomen. Zie daarvoor de files HILEVEL1.FRT en HILEVEL2.FRT als voorbeeld in de EXAMPLES directory en bladzijde 24 van dit boek.

4 Meer over de compiler

4.1 De Compiler instellingen

Als ByteForth is opgestart of herstart, dan staat hij gereed voor de AT90S2313 met de tracer uit. Een andere chip moet altijd expliciet genoemd worden.

90S2313	Doel processor is AT90S2313 of een van 28 andere AVR's.
90S2313?	Laat true vlag op de stack achter als de AT90S2313 het doel is of false indien een andere AVR geselecteerd is.
MEMORY	Wijzig de geheugen indeling van de controller. Een voorbeeld: `11 6 8 MEMORY` Start code op 16-bits adres 11, reserveer ruimte voor zes bitvlaggen (één register) en 8 variabelen. Het resterende RAM max. tot adres 256 wordt gebruikt voor de returnstack. Al het geheugen daarboven kan o.a. voor arrays gebruikt worden.
MAP	Wijzig de geheugen indeling van de controller geheel. Voorbeeld: `2. 11 25 16 20 MAP` Start code op 16-bits adres 2, reserveer ruimte voor 11 bitvlaggen (twee registers), 25 variabelen, een datastack van 16 bytes en een returnstack van 20 16-bits cellen. De resterende ruimte is beschikbaar voor arrays en `CREATE` en `DOES>`.
.MEMORY	Toon de geheugen indeling van de controller, enz.
.FREE	Toon vrije programma geheugen van de controller.
OPTIMIZER-ON	Zet de optimalisator aan (default).
OPTIMIZER-OFF	Zet de optimalisator uit.

4.1.1 Ondersteunde AVR's

AVR-type	Selector	Vlag	AVR-type	Selector	Vlag
AT90S1200	90S1200	90S1200?	AT90S2323	90S2323	90S2323?
AT90S2343	90S2343	90S2343?	AT90S2333	90S2333	90S2333?
AT90S4433	90S4433	90S4433?	AT90S4414	90S4414	90S4414?
AT90S4434	90S4434	90S4434?	AT90S8515	90S8515	90S8515?
AT90S8535	90S8535	90S8535?	ATtiny12	tiny12	tiny12?
ATtiny15	tiny15	tiny15?	ATtiny22	tiny22	tiny22?
AT90C8534	90C8534	90C8534?	AT90S8555	90S8555	90S8555?
ATmega161	mega161	mega161?	ATmega163	mega163	mega163?
ATmega103	mega103	mega103?	ATmega323	mega323	mega323?
ATmega8	mega8	mega8?	ATmega16	mega16	mega16?
ATmega32	mega32	mega32?	ATmega64	mega64	mega64?
ATmega8515	mega8515	mega8515?	ATmega8535	mega8535	mega8535?
ATmega162	mega162	mega162?	ATmega169	mega169	mega169?
ATtiny26	tiny26	tiny26?	ATtiny13	tiny13	tiny13?
ATtiny2313	tiny2313	tiny2313?	ATmega48	mega48	megta48?
ATmega88	mega88	mega88?	ATmega169	mega169	mega169?
ATmega165	mega165	mega165?			

4.1.2 `MEMORY` voorbeeld

Hoe gebruik ik `MEMORY`? Kies eerst de 'target' chip b.v. `90S2313` (de AT90S2313). Als je weet welke interrupt vectoren je niet gebruikt en hoeveel RAM en bit-vlaggen je exact nodig hebt, dan kun je het systeem 'finetunen'. Als voorbeeld een programma dat geen interrupts

gebruikt, geen bitvlaggen en slechts vier variabelen.
Voor het fijn afstellen kun je het volgende opgeven: `1 0 4 MEMORY`. Bij het trimmen van het codegeheugen kun je als hulp de tabel op bladzijde 22 met opstart en interrupt vectoren gebruiken. De code start op 16-bits adres 1 van het Flash-ROM (direct na de opstartvector), er worden geen bitvlaggen gereserveerd en slechts 4 variabelen.
Het lijkt misschien een zinloze bezigheid en dat is het hier ook! Dat verandert echter zodra je enkele bytes RAM of Flash tekort komt. De woorden `MEMORY` of `MAP` helpen je dan uit de brand.
De hardware stack begint direct achter de datastack en loopt maximaal tot RAM adres 255. Na de hardwarestack zitten max. 64 variabelen. ByteForth geeft een foutmelding als de returnstack kleiner wordt dan 8 cellen. Het systeem wordt met onvoldoende returnstackruimte gemakkelijk onstabiel!
Als er geheugen boven adres 255 beschikbaar is (niet bij de AT90S2313), dan wordt dat o.a. gebruikt voor arrays (`VARIABLES`). Maar ook door woorden die met `CREATE` en `DOES>` gemaakt zijn, een voorbeeld daarvan vindt je op bladzijde 21.

4.1.3 `MAP` voorbeeld

Ook `MAP` wordt gebruikt voor het wijzigen van de geheugen layout: `11. 16 32 16 16 MAP`. De code start op 16-bits adres 11 (dit is een dubbel getal). Reserveer 16 bitvlaggen (dat kost 2 registers) en 32 variabelen. Een datastack van 16 posities en een returnstack van 16 cellen. De opgave voor de returnstack is een voorkeurswaarde, het systeem besteed 16 cellen of minder aan deze stack. Het commando `MAP` is verder gelijk aan `MEMORY`, het geeft gebruikers van AVR ByteForth de mogelijkheid het systeem nog verder te 'finetunen'.

4.2 Controle structuren

ByteForth heeft net zoals elke Forth de beschikking over een aantal controle structuren. Om te beginnen een lijst van de beschikbare standaard structuren. Het woord `<test>` staat voor een van de Forth test instructies als `0= > U<` etc, `<code>` staat voor een willekeurig stukje programma.

```
<test> IF <code> THEN
<test> IF <code1> ELSE <code2> THEN

BEGIN <code>  <test> UNTIL
BEGIN <code> AGAIN
BEGIN <test> WHILE <code> REPEAT

<limiet> <start> DO <code> LOOP

CASE
   <getal> OF <code> ENDOF
ENDCASE
```

Verder zijn er nog enkele niet standaard structuren. Deze structuren zijn aangepast aan de instructieset van de AVR. De `FOR NEXT` lus heeft nog bijna in de ANSI standaard gezeten. Het `SELECT`-statement is handig als een programma erg veel keuzes moet maken. Het grootste nadeel ervan is, dat het niet direct in een lus gebruikt kan worden door de verplichte `EXIT` aan het einde van elke `SELECT` ingang. Door deze `EXIT` valt het namelijk uit elke lus. Het woord `<getaln>` hieronder staat voor een konstante als selectie waarde voor het `SELECT`-statement. Zie voor een uitgebreidere uitleg de woordenlijst aan het eind van dit hoofdstuk of type `HELP` en daarna het woord in.

```
<getal> FOR <code> NEXT

BEGIN-SELECT
  SELECT <getal1> <code> EXIT
  SELECT <getal2> <code> EXIT
         <etc>
END-SELECT
```

Voor DO LOOP zijn er nog de woorden I J UNLOOP LEAVE beschikbaar en voor FOR NEXT de woorden I' J' UNNEXT. Zoek de werking op met behulp van de help functie of in het woorden overzicht (glossary).

4.3 Definiërende woorden

ByteForth bevat een groot aantal zogenaamde definiërende woorden (datatypen). Een voorbeeld hiervan is VARIABLE dat een naam toekent aan een uniek RAM adres. Bijna alle definiërende woorden zijn via prefix operators toegankelijk, met uitzondering van CONSTANT en 2CONSTANT. De meest voorkomende operators zijn: ADR, FROM, TO, CLEAR en SET. Daarnaast vindt je nog: +TO, TOGGLE, etc. Een complete lijst geldige prefixen is bij ieder datatype afgedrukt. Hier een voorbeeld van het gebruik: $18 0 BIT-SFR UITGANG maakt het poortbit PB.0 onder de naam UITGANG bruikbaar in een programma. Met CLEAR UITGANG wordt PB.0 laag gemaakt, SET UITGANG maakt PB.0 hoog. Een kort overzicht van alle definiërende woorden:

VALUE	8 bits TO-variabele.
2VALUE	16 bits TO-variabele.
VALUES	Array van 8 bits TO-variabelen.
2VALUES	Array van 16 bits TO-variabelen.
VARIABLE	8 bits variabele.
2VARIABLE	16 bits variabele.
VARIABLES	Array van 8 bits variabelen.
2VARIABLES	Array van 16 bits variabelen.
CONSTANT	8 bits constante.
2CONSTANT	16 bits constante.
CONSTANTS	Tabel van 8 bits constanten.
REGISTER	8 bits register-variabele.
(REGISTER)	8 bits register-variabele (adres van stack).
SFR	8 bits Special Function Register toegang.
BIT-SFR	High-level bit-SFR toegang.
FLAG	Toegang tot een bit-vlag.
(FLAG)	Toegang tot een bit-vlag (adres van stack).
CREATE	Begin een nieuwe datastructuur.
DOES>	Definieer een nieuwe executie interpreter.
::	Begin een nieuw definiërend woord.

4.4 Nieuwe definiërende woorden maken

Het is nu ook in ByteForth mogelijk zelf nieuwe datastructuren erbij te maken. Dat doe je met de woorden CREATE en DOES>. Een voorbeeld waar VARIABLE opnieuw gemaakt wordt is op zijn plaats. Meer voorbeelden vindt je op bladzijde 14.

```
RAM                  \ Maak datastructuur in RAM
:: VAR               \ Start nieuw definierend woord genaamd VAR
   CREATE  1. ALLOT  \ Maak header en reserveer een byte in RAM
   DOES>             \ Maak uitvoerende code die het adres van
   ;                 \ de gereserveerde RAM locatie op stack zet
ROM                  \ Maak volgende structuren weer in ROM (default)
VAR AAP              \ Maak een VAR met de naam AAP
1 AAP !              \ Zet 1 in AAP
AAP @ .              \ Lees inhoud van AAP en druk af
12 AAP +!            \ Verhoog inhoud van AAP
AAP @ .              \ Toon nieuwe inhoud van AAP
```

Het stuk code tot DOES> bestaat alleen in de ByteForth crosscompiler op de PC, de code die DOES> genereert tot en met ; komt in de microcontroller te staan.

4.5 Definiërende woorden voor gevorderden

Er is nog een tweede manier om definïerende woorden aan AVR ByteForth toe te voegen. Deze methode staat het gebruik van prefixen toe. Met CONSTRUCT zet je een speciale compiler aan. Datastructuren maak je zo op dezelfde manier als de bestaande ByteForth definiërende woorden. Met het commando TARGET keer je terug in ByteForth. Daar ga je de nieuwe datastructuren gebruiken.

```
CONSTRUCT               \ Activeer speciale compiler
RAM                     \ Nieuwe datastructuur in RAM
: VAL                   \ Nieuwe type heet VAL
  CREATE  HERE REAL> ,  \ Maak header, onthoudt RAM index
  1. ALLOT  IMMEDIATE   \ Reserveer 1 byte, VAL is immediate
  DOES> @ FLYER >R      \ VAL is ook interactief bruikbaar
    R16 R> [Y] LD,      \ Laad VAL-data in R16
    PUSHA, ;            \ Zet inhoud VAL op stack
                        \ Optimizer code hiervan is 1
ROM                     \ Data structuren weer in ROM (default)
METHODS VAL             \ Maak prefix acties voor VAL
: TO    @  POPA,        \ Bewaar RAM-index, pop getal van stack
        [Y] R16 ST,     \ Zet getal in VAL
        NO-OUTPUT ;     \ Geen uitvoer voor optimizer en klaar
: +TO   @ >R  POPA,     \ Bewaar RAM index, pop getal van stack
        R17 R@ [Y] LD,  \ Lees inhoud VAL naar R17
        R16 R17 ADD,    \ Tel inhoud R16 en R17 op
        R> [Y] R16 ST,  \ Zet resultaat terug in VAL
        NO-OUTPUT ;     \ Geen uitvoer voor optimizer en klaar
END-METHODS             \ Stop het definieren van prefix acties
TARGET                  \ Terug naar ByteForth

VAL NOOT                \ Maak een VAL met de naam NOOT
1 TO NOOT               \ Zet 1 in NOOT
NOOT .                  \ Lees inhoud van NOOT en druk af
12 +TO NOOT             \ Tel 12 op bij de inhoud van NOOT
NOOT .                  \ Toon nieuw inhoud van NOOT
```

4.6 Speciale commando's

De ByteForth compiler kent enkele speciale commando's. Dit zijn `."`, `SLITERAL` en `S"`. Als je ze toe wil passen, moet je er voor zorgen dat geschikte primitieven aanwezig zijn. Dit betekent voor alle bovenstaande woorden, dat de speciale macro `INLINE$` aanwezig moet zijn. Je doet dit door: `ATOM INLINE$` voor het gebruik van deze commando's in het programma op te nemen. Voor `."` is daarnaast ook het woord `TYPE` nodig die o.a. voorkomt in de bibliotheek files: RS232.FRT en LCD.FRT zie bladzijde 67, maar ook als debugger woord! Als een van deze files niet geladen is, wordt de versie uit de debugger gebruikt. Wil je weten voor welke uitvoer een `TYPE` beschikbaar is, bekijk dan op bladzijde 68 het woorden overzicht (glossary) van de bibliotheek. Andere speciale woorden zijn:

`SETUP-BYTEFORTH`	Installeer de Forth virtuele machine.
`ATOM`	Importeer een macro als woord (subroutine).
`>HOST`	Ga naar CHForth, verlaat ByteForth.
`>TARGET`	Keer terug naar ByteForth.
`>CROSS`	Pseudoniem van `>TARGET`.
`SET-CRYSTAL`	Geef kloksnelheid in MHz aan compiler door.
`CRYSTAL?`	Geef true als op de stack de geselecteerde kloksnelheid staat. Zie o.a. MS.FRT

4.7 Interrupt- en resetvectoren

AVR ByteForth heeft ook commando's om de reset- en interrupt-vectoren te zetten. Een vector moet het adres mee krijgen van het woord dat je uit wilt laten voeren. Interrupt vectoren van de AT90S2313 zijn: `MAIN`, `EXTERN0`, `EXTERN1`, `T1-CAPTURE`, `T1-COMPARE`, `T1-OVERFLOW`, `T0-OVERFLOW`, `UART-RX`, `UART-EMPTY`, `UART-TX` en `COMPARATOR`. Bedenk wel dat het aantal interruptvectoren sterk verschilt per AVR-chip.

Enkele voorbeelden:

```
CODE Externe-interrupt  ( -- )
     ( Doe iets zinnigs ) RETI,
END-CODE  EXTERN0

: Hoofdprogramma        ( -- )
     SETUP-BYTEFORTH  BEGIN ( nuttige code ) AGAIN ;  MAIN
```

Vector No.	Program Address	Source	Interrupt Definition
1	$000	RESET	Hardware Pin, Power-on Reset and Watchdog Reset
2	$001	INT0	External Interrupt Request 0
3	$002	INT1	External Interrupt Request 1
4	$003	TIMER1 CAPT1	Timer/Counter1 Capture Event
5	$004	TIMER1 COMP1	Timer/Counter1 Compare Match
6	$005	TIMER1 OVF1	Timer/Counter1 Overflow
7	$006	TIMER0 OVF0	Timer/Counter0 Overflow
8	$007	UART, RX	UART, RX Complete
9	$008	UART, UDRE	UART Data Register Empty
10	$009	UART, TX	UART, TX Complete
11	$00A	ANA_COMP	Analog Comparator

Figuur 4.1: Reset- en Interruptvectoren van de AT90S2313

Meer gegevens over interrupt's zijn te vinden in de voorbeeld files: PBM-INT.FRT en HILEVEL1.FRT. In het ATMEL microcontroller databoek uit 1999 en op het internet (zie bladzijde 119) is alle informatie terug te vinden:

Atmel Coorporation
AVR RISC Microcontroller databoek, Augustus 1999
E-mail: literature@atmel.com
Web Site: http://www.atmel.com

In Nederland vertegenwoordigd door ALCOM electronics bv
Tel: 010 - 288 25 00 Fax: 010 - 288 25 25

Timer/Counter Interrupt Mask Register - TIMSK

Bit	7	6	5	4	3	2	1	0	
$39 ($59)	TOIE1	OCIE1A	-	-	TICIE1	-	TOIE0	-	TIMSK
Read/Write	R/W	R/W	R	R	R/W	R	R/W	R	
Initial value	0	0	0	0	0	0	0	0	

- **Bit 7 - TOIE1: Timer/Counter1 Overflow Interrupt Enable**

When the TOIE1 bit is set (one) and the I-bit in the Status Register is set (one), the Timer/Counter1 Overflow interrupt is enabled. The corresponding interrupt (at vector $005) is executed if an overflow in Timer/Counter1 occurs, i.e., when the TOV1 bit is set in the Timer/Counter Interrupt Flag Register - TIFR.

- **Bit 6 - OCIE1A: Timer/Counter1 Output Compare Match Interrupt Enable**

When the OCIE1A bit is set (one) and the I-bit in the Status Register is set (one), the Timer/Counter1 Compare Match interrupt is enabled. The corresponding interrupt (at vector $004) is executed if a Compare match in Timer/Counter1 occurs, i.e., when the OCF1A bit is set in the Timer/Counter Interrupt Flag Register - TIFR.

- **Bit 5,4 - Res: Reserved bits**

These bits are reserved bits in the AT90S2313 and always read as zero.

- **Bit 3 - TICIE1: Timer/Counter1 Input Capture Interrupt Enable**

When the TICIE1 bit is set (one) and the I-bit in the Status Register is set (one), the Timer/Counter1 Input Capture Event Interrupt is enabled. The corresponding interrupt (at vector $003) is executed if a capture-triggering event occurs on PD6(ICP), i.e., when the ICF1 bit is set in the Timer/Counter Interrupt Flag Register - TIFR.

- **Bit 2 - Res: Reserved bit**

This bit is a reserved bit in the AT90S2313 and always reads as zero.

- **Bit 1 - TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is set (one) and the I-bit in the Status Register is set (one), the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt (at vector $006) is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter Interrupt Flag Register - TIFR.

- **Bit 0 - Res: Reserved bit**

This bit is a reserved bit in the AT90S2313 and always read as zero.

Figuur 4.2: TIMSK register beschrijving

General Interrupt Mask Register - GIMSK

Bit	7	6	5	4	3	2	1	0	
$3B ($5B)	INT1	INT0	-	-	-	-	-	-	GIMSK
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial value	0	0	0	0	0	0	0	0	

- **Bit 7 - INT1: External Interrupt Request 1 Enable**

When the INT1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control1 bits 1/0 (ISC11 and ISC10) in the MCU general Control Register (MCUCR) defines whether the external interrupt is activated on rising or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of External Interrupt Request 1 is executed from program memory address $002. See also "External Interrupts" on page 26.

- **Bit 6 - INT0: External Interrupt Request 0 Enable**

When the INT0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), the external pin interrupt is enabled. The Interrupt Sense Control0 bits 1/0 (ISC01 and ISC00) in the MCU general Control Register (MCUCR) defines whether the external interrupt is activated on rising or falling edge of the INT0 pin or level sensed. Activity on the pin will cause an interrupt request even if INT0 is configured as an output. The corresponding interrupt of External Interrupt Request 0 is executed from program memory address $001. See also "External Interrupts."

- **Bits 5..0 - Res: Reserved bits**

These bits are reserved bits in the AT90S2313 and always read as zero.

Figuur 4.3: GIMSK register beschrijving

4.8 High-level interrupts

Voor het gebruik van High-level interrupts is ondersteuning toegevoegd, een voorbeeld:

```
TCNT0 SFR TELLER0       \ Timer-0 register
VARIABLE TELLER         \ Teller voor gebruiker
: TEL ( -- )            \ Begin high level interrupt
    PUSHALL             \ Bewaar Forth omgeving
    0 TO TELLER0        \ Herstart timer-0
    INCR TELLER         \ Verhoog teller
    POPALL              \ Herstel Forth omgeving
    ;INT  T0-OVERFLOW   \ Sluit af als interrupt en zet vector
```

4.9 Commando's aan de compiler toevoegen

Je kan de crosscompiler ook op andere vlakken uitbreiden. Doordat het woord `::` is toegevoegd kun je commando's maken die bestaan uit woorden van het host-Forth systeem. Een voorbeeld:

```
:: BIN  ( -- )  2 BASE ! ;  \ Zet getal basis binair
```

4.10 Assembler

De assembler past zich automatisch aan bij het geselecteerde AVR-type. Bij AVR's met een klein FLASH geheugen zijn in de generieke `GJMP,` en `GCALL,` de `(E)JMP,` en `(E)CALL,` uitgeschakeld. Zie verder bladzijde 130 (Datasheet AT90S2313) voor een overzicht van zijn AVR-instructies. Sommige AVR's hebben meer of minder instructies, zie daarvoor de assembler beschrijving, beginnend op bladzijde 95.

4.11 Het testen van code

De code is met zekere beperkingen, te testen. Hiervoor hoef je slechts de naam van een woord in te toetsen en het wordt uitgevoerd. Ook hier een voorbeeld, voer eerst EMPTY uit om er voor te zorgen dat de compiler geheel blanco begint.

```
EMPTY <cr>
: Verlaag-met-tien  10 - ; <cr>
100 Verlaag-met-tien . <cr>
```

Je maakt het woord `Verlaag-met-tien`. Dit verlaagt het getal bovenop de stack met 10. Om te controleren of het werkt type je bovenstaande regel in. Het getal 100 wordt op de stack geplaatst en daarna wordt `Verlaag-met-tien` uitgevoerd. Het woord `.` toont daarna het het resultaat.

4.12 Foutzoeken

Foutzoeken gaat in (Byte)Forth altijd het best door alle losse woorden eerst te testen. Zet pas daarna het hele programma in een AVR-chip en probeer het uit. Een volledige disassembler is in deze release opgenomen, SEE `ccc`. Werk nauwkeurig en gestructureerd, dan worden veel fouten voorkomen. Probeer ook met zoveel mogelijk kleine en makkelijk testbare woorden te werken.

4.13 De simulator

AVR ByteForth heeft een ingebouwde software simulator. Veel van de zelfgemaakte code kan zo uitgeprobeerd worden alsof je op de processor zelf werkt. In de simulator is ook een tracer ingebouwd, inclusief de mogelijkheid van breekpunten. Standaard staat deze tracer uit, maar met `TRACER-ON` worden de uitgevoerde instructies (en het resultaat) op het scherm afgedrukt. De instelling van de tracer geschiedt geheel naar wens van de programmeur. b.v. `ECHO-ON PORTS-ON SHORT` laat na het voorbijkomen van het breekpunt de code zien met ervoor een korte weergave van de poorten en interne registers. Als er breekpunten gebruikt zijn wordt de tracer actief gemaakt door `BREAKPOINT` en inactief door `RUNPOINT` te gebruiken in de code. Speel er mee dan ontdek je vanzelf de mogelijkheden. De tracer kan maximaal tien 'break'-punten (tracer aan) en tien 'run'-punten (tracer uit) aanbrengen. Een overzicht van de commando's:

Commando	Tracer actie
`ECHO-ON`	Druk opcodes af tijdens het uitvoeren.
`ECHO-OFF`	Druk geen opcodes af.
`HEXDUMP-ON`	Voeg hexdump toe voor opcodes.
`HEXDUMP-OFF`	Laat hexdump weg voor opcodes.
`STEP-ON`	Voer opcodes stap voor stap uit.
`STEP-OFF`	Stap voor stap uitvoeren staat uit.
`PORTS-ON`	Display interne (I/O)-registers tijdens uitvoeren.
`PORTS-OFF`	Zet register display af.
`SHORT`	Toon poort info voor de opcode op zelfde regel.
`LONG`	Toon poort info als een apart blok.
`TRACER-ON`	Activeer bovenstaande tracer opties.
`TRACER-OFF`	Deactiveer bovenstaande tracer opties.
`BREAKPOINT`	Zet tracer aan op dit punt in de code.
`RUNPOINT`	Zet tracer uit op dit punt in de code, ga door met executeren van de code.
`BREAK-OFF`	Zet break en run punten uit (op nul).
`.TRACER`	Toon instelling van de tracer.
`.BREAKPOINTS`	Toon break en run punten in de code.
`CONTINUE`	Ga door waar je gebleven was bij de vorige simulatie.

De actieve toetsen in de tracer zijn alle gewone toetsen, de spatiebalk en de escapetoets. De escapetoets werkt altijd als noodstop. In de stap-voor-stap toestand voer je met elke druk op de spatiebalk een opcode uit, elke andere toets laat het tracen stoppen.
In de toestand waar de code vrij loopt werkt de spatiebalk als een Stop/Start toets, alle andere toetsen behalve escape doen niets.

4.14 Breakpoint voorbeeld

Het aanbrengen van een breekpunt gebeurt als hieronder getoond. Zodra je `tel` uitvoert wordt de tracer geactiveerd op code tussen `breakpoint` en `runpoint`, hier is dat `+`.

```
: tel       ( -- u )
    0               ( Zet teller op stack )
    200 for         ( Doe lus 200 maal )
        1           ( Zet 1 op de stack )
        breakpoint  ( Start tracer hier )
        +           ( Tel 1 op bij teller )
        runpoint    ( Stop tracer hier weer )
    next ;          ( Naar begin lus )

step-off            ( Code mag vrij lopen )
tel                 ( Voer voorbeeld uit )
```

4.15 Tracer schermafdruk

```
$008B  R16 X+ LD,
$008C  R17 X+ LD,
$008D  R16 R17 ADD,
$008E  -X R16 ST,                       PrtB  1 1 1 1 1 1 1 1
$008F  RET,                             PinB  1 1 1 1 1 1 1 1
110  Ok                                 DdrB  1 1 1 1 1 1 1 1
Forth> 100 10 + .
                                        PrtD  1 1 1 1 1 1 1 1
                                        PinD  1 1 1 1 1 1 1 1
                                        DdrD  1 1 1 1 1 1 1 1


                                              I T H S V N Z C
                                        Sreg  0 0 0 0 0 0 0 0

                                         R16  $6E    110
                                         R17  $64    100
                                        Xreg  $1D    29
                                          SP  $44    68
                                         TOS  $6E    110
                                         SOS  Empty


Vsn 2.00demo   0   10  $000B   C:\AVRF\WORK
```

Hierboven een vereenvoudigde scherm afdruk van de tracer in actie. De tracer heeft net de code `100 10 + .` uitgevoerd. De code van het woord plus is linksboven op het scherm getoond. Rechtsboven de toestand van I/O-poort `PORTB` en `PORTD`, daaronder `Sreg` (statusregister). Het blok rechtsonder bevat achtereenvolgens de inhoud van de accu's `R16` en `R17`, de inhoud van het `X-reg` (de stackpointer), `SP` de returnstackpointer, tenslotte `TOS` en `SOS` (Top Of Stack en Second Of Stack). De onderste regel is de statusregel met daarop: links het versienummer van de compiler, dan de stackdiepte (0), het talstelsel (10=decimaal), de inhoud van `HERE` `$000B` tenslotte het actieve directory.

4.16 Extra debugger woorden.

Om fouten op te sporen heeft de debugger ook de mogelijkheid om CHForth afdruk operaties toe te voegen aan de code. Hieronder een lijst:

.S	Toon de stack inhoud.
.	Druk byte met teken af.
U.	Druk byte zonder teken af.
D.	Druk 16-bits word met teken af .
DU.	Druk 16-bits word zonder teken af.
.HEX	Druk byte hexadecimaal af.
D.HEX	Druk 16-bits word hexadecimaal af.
Q.	Druk 32-bits word met teken af.
QU.	Druk 32-bits word zonder teken af.
EMIT	Druk ASCII karakter af.
TYPE	Druk ASCII string af.
KEY?	Test of toets ingedrukt is.
KEY	Geef toetsaanslag op stack.
CR	Ga naar begin nieuwe regel.
PAGE	Maak het scherm schoon.

De debugger woorden kunnen ook gebruikt worden om code te tracen, je traced dan op de 'high-level' Forth manier. Door tussentijds enkele getallen of de toestand van de datastack te printen controleer je de code.

```
: tel       ( -- u )
    0               ( Zet teller op stack )
    200 for         ( Doe lus 200 maal )
        1           ( Zet 1 op de stack )
        +           ( Tel 1 op bij teller )
        .s          ( Toon toestand van datastack )
    next ;          ( Naar begin lus )

tel                 ( Voer voorbeeld uit )
```

4.17 Betekenis stack commentaar

Symbool	Datatype	Stack formaat
x	ongedefinieerde cel inhoud	1 cel
n	getal van -127 tot 127	1 cel
u	getal van 0 tot 255	1 cel
+n	getal van 0 tot 127	1 cel
flag	een vlag	1 cel
true	een vlag met alle bits hoog	1 cel
false	een vlag met alle bits laag	1 cel
char	ASCII karakter	1 cel
dx	ongedefinieerde dubbele cel inhoud	2 cellen
d	getal van -32767 tot 32767	2 cellen
du	getal van 0 tot 65535	2 cellen
+d	getal van 0 tot 32767	2 cellen
q	getal van -2147483647 tot 2147483647	4 cellen
qu	getal van 0 tot 4294967295	4 cellen
addr	ram- of romadres	2 cellen
$addr	ram- of romadres van een string	2 cellen
c-addr	code (ROM) adres	2 cellen
xt	execution token (c-addr)	2 cellen
"name"	naam van Forth woord	0 cellen
"ccc"	willekeurige string	0 cellen
sys	systeem data	?? cellen

4.18 CROSS woordenlijst

In een glossary beschrijving betekent een getal tussen haakjes (10) bijvoorbeeld, dat de beschreven Forth instructie 10-bytes assembleert als er niet geoptimaliseerd wordt.

```
'                   "tick"                                  Cross
   ( "name" -- xt )
   Lees het woord 'name' uit de invoerstroom en laat 'xt'
   het token op de stack achter, indien 'name' bestaat.

(                   "paren"                                 Cross
   ( "ccc" -- )
   Skip commentaar tot het afsluitende rechter haakje (0).

(FLAG)              "paren-flag-paren"                      Cross
   ( flag-nr "name" -- flag )
   Definieer een bitvlag 'flag-nr' met de naam 'name'. Tijdens
   het uitvoeren zet het zijn conditie als forth vlag op de stack.
   Het moet gebruikt worden met de >FLAG conversie operator. Een
   voorbeeld: 20 4 >FLAG (FLAG) AAP  dit definieert een vlag in
   register 20 op bit positie 4 met de naam AAP (10).
   Geldige prefixen zijn: ADR TO SET CLEAR TOGGLE
   Noot: De ADR prefix mag alleen in de assembler worden gebruikt.
```

(REGISTER) "paren-register-paren" Cross
(reg "name" -- x)

Definieer register toegang tot ’reg’ met de naam ’name’. Dit is een variable achtig woord, maar sneller en korter in gebruik. Te gebruiken op plaatsen waar snelheid en/of RAM grootte erg belangrijk zijn (4). Geldige prefixen zijn: ADR TO +TO CLEAR SET INCR DECR TOGGLE

+TO "plus-to" Cross
(n|u "name" --)

Tel n|u op bij ’name’. Zie ook: VARIABLE SFR 2VARIABLE etc.

, "comma" Cross
(x --)

Compileer het datawoord ’x’ in ROM of geef een foutmelding als een poging wordt gewaagd om in RAM te compileren.

. "dot" Cross
(n --)

Druk een ByteForth enkel getal ’n’ met teken af.

." "dot-quote" Cross
("ccc" --)

Compileer een inline string in ByteForth. Dit woord heeft de woorden INLINE$ en TYPE nodig (gebruik ATOM) (5+stringlengte).

.("dot-paren" Cross
("ccc<paren>" --)

Lees de invoerstroom tot het eerstvolgende haakje sluiten en druk die tekst zonder het haakje sluiten af, ook tijdens het compileren.

.BREAKPOINTS "dot-breakpoints" Cross
(--)

Toon alle actieve breakpoints & runpoints.

.FREE "dot-free" Cross
(--)

Toon het vrije en gebruikte code geheugen in 16-bits cellen.

.HELP "dot-help" ISP
(--)

Toon de commando’s van de programmeer software.

.HEX "dot-hex" Cross
(x --)

Druk het ByteForth enkel getal ’x’ hexadecimaal af.

.MEMORY "dot-memory" Cross
(--)

Toon de toestand van de verschillende ByteForth geheugen gebieden en systeem data. Zie ook: MEMORY MAP .FREE

.PAUSE "dot-pause" ISP
(--)

Toon klokpuls verlengtijd, default is 100.

.S "dot-s" Cross
(--)

Copieer en toon alle waarden op de datastack.

.TICKS "dot-ticks" Cross
(--)

Toon de gebruikte processortijd in cpu-tikken. Alleen geldig nadat een stuk code door de simulator is uitgevoerd.

.TRACER "dot-tracer" Cross
(--)

Toon de tracer configuratie.

2CONSTANT "two-constant" Cross
(dx "name" -- dx)

Definieer een dubbele konstante met de naam ’name’ en de waarde ’dx’ (8).

2LITERAL "two-literal" Cross
(d --)

Compileer ’x’ als een 16 bits dubbel getal (8).

2VALUE "two-value" Cross
("name" -- dx)

Definieer een dubbele varlue met de naam ’name’.
Zet de inhoud van de dubbele cel dx op de stack.
Geldige prefixen zijn:
ADR FROM TO +TO SET CLEAR INCR DECR POP PUSH TOGGLE
Noot: De ADR prefix mag alleen in de assembler worden gebruikt (8).
Zie ook: LOW

2VALUES "two-values" Cross
(u "name" -- dx)

Definieer een array van ’u’ dubbele value’s met de naam ’name’.
Zet de dubbel ’dx’ behorend bij ’u’ op de stack.
Pas op, er is geen bereik controle op ’u’ ingebouwd. Geldige prefixen zijn: ADR FROM TO +TO CLEAR SET INCR DECR
Noot: De ADR prefix mag alleen in de assembler worden gebruikt (18).
Zie ook: LOW

2VARIABLE "two-variable" Cross
("name" -- addr)

Definieer een dubbele variabele met de naam ’name’.
Zet het adres ’addr’ van de dubbele cel op de stack.
Geldige prefixen zijn:
ADR FROM TO +TO SET CLEAR INCR DECR POP PUSH TOGGLE
Noot: De ADR prefix mag alleen in de assembler worden gebruikt (8).
Zie ook: LOW 2@ 2!

2VARIABLES "two-variables" Cross
(u "name" -- addr)

Definieer een array van ’u’ dubbele variabelen met de naam ’name’.
Zet het dubbelecel adres ’addr’ behorend bij ’u’ op de stack.
Pas op, er is geen bereik controle op ’u’ ingebouwd. Geldige
prefixen zijn: ADR FROM TO +TO CLEAR SET INCR DECR
Noot: De ADR prefix mag alleen in de assembler worden gebruikt (16).
Zie ook: LOW 2@ 2!

90S1200? "90S1200-query" Cross
(-- flag)

Zet true op de stack als de AT90S1200 geselecteerd is, anders
false. AVR ByteForth ondersteund 28 verschillende AVR Cpu’s.

90S2313 Cross
(--)

Selecteer de juiste geheugenmap, instructieset, ISP-programmer
en geef de compiler op dat de AT90S2313 het doel (target) is.
AVR ByteForth ondersteund 30 of meer verschillende AVR Cpu’s.

: "colon" Cross
("name" --)

Definieer een colon definitie met de naam ’name’ (0).

:: "colon-colon" Cross
("name" --)

Start een nieuwe colon definitie op de host met ’name’.
De zoekvolgorde is INTERPRETER INTERPRETER HOST-FORTH.
Het nieuwe woord wordt aan de CROSS woordenlijst toegevoegd.
Wordt gebruikt om de ByteForth crosscompiler uit te breiden.

:MAIN "colon-main" Cross
(--)

Definieer het hoofdprogramma, dit woord zet de startup
vector van de gewenste processor en compileert de minimaal
benodigde startup code (12 tot 36).

; "semicolon" Cross
(--)

Sluit een colon definitie af (0 of 2).

```
;ASS                "semicolon-ass"                          Cross
   ( -- )
   Schakel om naar Forth zodat het assembleren afgesloten wordt.

;INT                "semicolon-interrupt"                    Cross
   ( -- )
   Sluit een colon definitie die als 'high-level' interrupt
   code gebruikt zal worden (2).

>AVR                "to-a-v-r"                               ISP
   ( b0 b1 b2 b3 -- )
   Stuur de vier bytes 'b0', 'b1', 'b2' en 'b3' naar een AVR chip
   waarmee handmatig een schrijfcommando toegevoegd wordt.
   Letop: De bytes staan in omgekeerde volgorde!

>CROSS              "to-cross"                               Cross
   ( -- )
   Schakel over naar de ByteForth dictionary en interpreter.
   Zie ook: >TARGET  >HOST

>FLAG               "to-flag"                                Cross
   ( reg bit -- flag-nr )
   Converteer de 'bit' in register 'reg' naar de vlag 'flag-nr'.
   Wordt gebruikt om vlag nummers voor het definierend woord (FLAG)
   te berekenen.

>HOST               "to-host"                                Cross
   ( -- )
   Schakel over naar de HOST dictionary en interpreter.
   Zie ook: >CROSS >TARGET

>REAL               "to-real"                                Assembler
   ( +n -- addr )
   Converteer een (2)VARIABLE offset '+n' naar een echt RAM adres 'addr'.

>TARGET             "to-target"                              Cross
   ( -- )
   Schakel over naar de ByteForth dictionary en interpreter.
   Dit is een pseudoniem voor >CROSS. Zie ook: >HOST

A:                  "drive-a"                                Cross
   ( -- )
   Selecteer drive-a als de actieve diskdrive. Er is ook
   ondersteuning voor drive: C: D: E: en F:
```

ADR "address" Cross

("name" -- addr)

Plaats ’addr’ behorend bij ’name’ op de stack.
Let op: Mag alleen tijdens het assembleren gebruikt worden !!
Zie ook: VARIABLE SFR 2VARIABLE etc.

AGAIN Cross

(--)

Ga door met uitvoeren na BEGIN. Als er geen andere controle structuren dan AGAIN zijn gebruikt, zal de code na AGAIN niet uitgevoerd worden (2).
Zie ook: BEGIN IF

AHEAD Cross

(--)

Ga verder met uitvoeren na THEN of ENTRY (2). Zie ook: THEN ELSE ENTRY

ALIAS Cross

(xt "ccc" --)

Definieer een alias voor de bestaande routine ’xt’ met de naam ’ccc’.
Zoals intern gebruikt bij: ’ R1 ALIAS LOOP-CNT (R1 = Lus teller).

ALIGN Cross

(--)

Align de coderuimte op de volgende 16-bits cel.

ALIGNED Cross

(addr -- a-addr)

’a-addr’ is het eerste gealignde code-adres groter dan of gelijk aan ’addr’ (14).

ALLOT Cross

(d --)

Reserveer ’d’ bytes van de dataruimte in ROM of RAM, afhankelijk van het laatste gebruik van ROM of RAM.

ASS: "ass-colon" Cross

(--)

Schakel om naar de assembler in een colon definitie.
Hierdoor kan er handgecodeerde assembly in opgenomen worden (0).

ASSEMBLER Cross

(--)

Vervang de bovenste lijst in de zoekvolgorde door de assembler woordenlijst.

ATOM Cross

("name" --)

Copieer de macro ’name’ in de ByteForth woordenlijst als subroutine. De lengte wordt macro+2 bytes.

AVR> "a-v-r-from" ISP

(b0 b1 b2 -- b3)

Stuur de drie bytes ’b0’, ’b1’ en ’b2’ naar een AVR chip om waarmee handmatig een leescommando toegevoegd wordt, ’b3’ is het resultaat. Letop: De bytes staan in omgekeerde volgorde!

B>M "bit-to-mask" Assembler

(n -- mask)

Converteer het bitnummer ’n’ naar het bit-masker ’mask’.

BEGIN Cross

(--)

Markeer het begin van een lus, ga verder met uitvoeren na BEGIN (0). Zie ook: AGAIN UNTIL REPEAT WHILE

BEGIN-SELECT Cross

(x --)

Haal ’x’ van de stack, voor gebruik met een SELECT statement (2). Zie ook: SELECT en END-SELECT Een voorbeeld:

```
BEGIN-SELECT
     SELECT 1  20 SPACES ." Aapjes " EXIT
     SELECT 2  WORDS                 EXIT
     SELECT 3  $10                   EXIT
END-SELECT
```

BIT-SFR "bit-s-f-r" Cross

(sfr-addr bit "name" -- flag)

Definieer een I/O-bit ’bit-addr’ met de naam ’name’. Zet de toestand van het I/O-bit ’flag’ op de stack (8). Geldige prefixen zijn: ADR TO SET CLEAR TOGGLE PULSE SETDIR Noot: De ADR prefix mag alleen in de assembler worden gebruikt. Als ’bit-addr’ een uitgangsbit is, leest FROM het invoer bit (PINx) behorend bij (PORTx).

BREAK-OFF "breakpoints-off" Cross

(--)

Verwijder alle breakpoints & runpoints.

BREAKPOINT Cross

(--)

Zet een breakpoint voor de simulator. Activeer de tracer op dit punt.

BYE Cross
(--)
verlaat ByteForth en geef de controle terug aan DOS.

C, "c-comma" Cross
(char --)
Compileer het karakter 'char' in ROM of geef een foutmelding
als een poging wordt gewaagd om in RAM te compileren.

CAPITALISE-OFF Cross
(--)
Converteer geparste source tekst niet naar hoofdletters.

CAPITALISE-ON Cross
(--)
Converteer geparste source tekst naar hoofdletters.

CASE Cross
(--)
Markeer de start van een CASE .. OF .. ENDOF .. ENDCASE structuur.
Zie ook: ENDCASE ENDOF OF (0).

CD "choose-dir" Cross
("ccc" --)
Voer CD uit met als parameters 'ccc'.

CDATA "c-data" Cross
(--)
Schakel de ROM adresruimte in. Zie ook: ROM, RAM en UDATA.
Dit woord is een pseudoniem voor ROM.

CHAR Cross
("abc" -- char)
Lees het woord 'abc' uit de invoerstroom en zet de ASCII waarde
van het eerste karakter 'char' op de stack.

CLEAR Cross
("name" --)
Sla nul op in 'name'. Zie ook: VARIABLE SFR 2VARIABLE etc.

CODE Assembler
("name" -- sys)
Definieer een codedefinitie met de naam 'name', voeg
de assembler woordenlijst aan de 'search order' toe (0).

COLD Cross
(--)
Herstart ByteForth.

COMPILER Cross
(--)
Vervang de eerste woordenlijst in de search-order door
de COMPILER woordenlijst.

CONSTANT Cross
("name" -- x)
Definieer een konstante met de naam ’name’ en de waarde ’x’ (4).

CONSTANTS Cross
(x1 .. xu u "name" -- || u -- xu)
Definieer een array van konstanten met als naam ’name’ met ’u’
elementen. ’x1’ t/m ’xu’ zijn de waarden van de tabel in omgekeerde
volgorde. De maximum lengte van een array is 256 konstanten.
De array wordt op een 16-bit cel gealligned zoals voor AVR’s nodig is.
Wees voorzichtig, er wordt geen bereik controle uitgevoerd (16).
Zet bij uitvoering van ’u’ ’name’ de konstante ’xu’ op de stack.
Geldige prefixen zijn: ADR STRUCTURE EXEC
Noot: De ADR prefix mag alleen in de assembler worden gebruikt.

CONSTRUCT Cross
(--)
Start het maken van low-level ’defining words’ voor ByteForth.

CONTINUE Cross
(--)
Ga verder met executeren vanaf het punt waar dit was gestopt.

CPU-VECTOR Cross
(addr "name" --)
Maak een nieuw woord ’name’, die wanneer hij uitgevoerd wordt de
interruptvector op ’addr’ vult met het beginadres van het laatst
gedefinieerde woord. Zie ook: MAIN

CR "c-r" Cross
(--)
Ga naar een nieuwe regel.

CREATE Cross
("name" -- addr)
Definieer een data- of coderuimte locatie met ’name’, geef bij het
uitvoeren ’addr’ die het adres is van ’name’s dataveld. ’addr’
is altijd een dubbel getal of het nu in RAM of ROM ligt (8).

CROSS Cross
(--)

Vervang de eerste woordenlijst in de search-order door de CROSS woordenlijst.

CRYSTAL? "crystal-query" Cross
(+n -- flag)

Laat een true flag op de stack achter als +n gelijk is aan de gewenste kristal frequentie. Zie ook: SET-CRYSTAL

CSSWAP "c-s-swap" Assembler
(--)

Verwissel de systeemdata van controle structuren (voor het ongestructureerd programmeren). Een voorbeeld:
BEGIN, .. AHEAD, CSSWAP .. <tst> UNTIL, .. THEN,

D, "d-comma" Cross
(dx --)

Compileer een dubbelwoord in ROM of geef een foutmelding als een poging wordt gewaagd om in RAM te compileren.

D. "d-dot" Cross
(d --)

Druk een ByteForth dubbel ’d’ met teken af.

D.HEX "d-dot-hex" Cross
(dx --)

Druk een ByteForth dubbel getal ’dx’ hexadecimaal af.

DB "d-b" ISP
(--)

Toon hexdump van het gebruikte deel van de ByteForth buffer.

DEBUG Cross
(--)

Vervang de eerste woordenlijst in de search-order door de DEBUG woordenlijst.

DECIMAL Cross
(--)

Maak de getal basis decimaal.

DECR Cross
("name" --)

Verlaag de inhoud van ’name’ met 1. Zie ook: VARIABLE SFR etc.

DIR "directory" Cross
("ccc" --)
Voer het DOS of Windows commando DIR uit met als parameters ’ccc’.

DIS "disassemble" Cross
(c-addr --)
Disassembleer code vanaf het geheugen-adres ’c-addr’.

DO Cross
(limit index -- sys)
Zet de gegevens voor een DO LOOP lus klaar de lus zal ’limiet-index’ maal worden doorlopen (12). Zie ook: I J LEAVE UNLOOP LOOP

DOC Cross
("ccc" --)
Skip alle tekst uit de invoerstroom totdat het woord ENDDOC gevonden woord. Geef een foutmelding als ENDDOC niet wordt gevonden. Is bedoeld om documentatie in sources op te nemen.

DOES> "does" Cross
(--)
Begin een nieuwe executie interpreter.

DSP0 "d-s-p-zero" Assembler
(-- addr)
Geef het adres ’addr’ van de data-stack bodem.

DU. "du-dot" Cross
(du --)
Druk een ByteForth dubbel ’du’ zonder teken af.

DUMP Cross
(c-addr ud --)
Dump ’ud’ bytes van het ByteForth codegeheugen vanaf ’c-addr’.
Noot: ’c-addr’ en ’ud’ zijn 16-bit cellen en niet bytes!

E "erase" ISP
(--)
Wis het gehele FLASH en EEPROM van de microcontroller.

EB "e-b" ISP
(--)
Wis de code buffer door hem met $FF te vullen.

EDIT Cross
("name" --)
Start de file-editor met de file ’name’. Als ’name’ niet opgegeven is, wordt de laatst gebruikt filenaam gebruikt. Zie ook: IN

EEALLOT "e-e-allot" Cross
(d --)
Reserveer ’d’ bytes van de EEPROM dataruimte.

EEHERE "e-e-here" Cross
(-- addr)
Geef het eerste vrije EEPROM adres ’addr’ (dubbel).

EEPROM "e-e-prom" ISP
(--)
De programmer werkt op het data (EEPROM) geheugen van de cpu.

EESIZE "e-e-size" Cross
(-- du)
Geef de EEPROM grootte ’du’ van de huidige ’target’ cpu.

ELSE Cross
(--)
Ga verder met uitvoeren na ELSE (2). Zie ook: IF THEN

EMIT Cross
(char --)
Zet het karakter ’c’ op het beeldscherm.

EMPTY Cross
(--)
Verwijder alle code uit ByteForth en herstart.

END-CODE "end-code" Assembler
(sys --)
Sluit de huidige woord definitie af. Verwijder de ASSEMBLER woordenlijst uit de ’search order’. Dit woord wordt gebruikt samen met het woord CODE .

END-SELECT Cross
(--)
Sluit een SELECT statement af (0). Zie ook: BEGIN-SELECT SELECT

ENDCASE "end-case" Cross
(x --)

Markeer de basis optie van een CASE .. OF .. ENDOF .. ENDCASE contructie. Verwijder de case-selectie waarde 'x' en ga verder met uitvoeren. Zie ook: CASE ENDOF OF ENDCASE (2).

ENDOF "end-of" Cross
(--)

Markeer het einde van een .. OF .. ENDOF .. stuk van een CASE-structuur. Ga verder met uitvoeren achter ENDCASE (2).
Zie ook: CASE ENDCASE OF

ENTRY Cross
(--)

Ga verder met uitvoeren na THEN (0). Zie ook: IF ELSE THEN etc.
V.b. AHEAD BEGIN <code> ENTRY <test> UNTIL <meer code>
Noot: De code in de lus wordt de eerste keer overgeslagen.

EXEC Cross
(+n "name" --)

Executeer het token in het cell paar '+n' in de CONSTANTS tabel 'name'.
Let op: Alleen geldig op tabellen gedefinieerd met CONSTANTS.

EXECUTE Cross
(xt --)

Executeer het executie token 'xt'.

EXIT Cross
(--)

Geef de controle terug aan de aanroepende definitie (0 or 2).

F>B "flag-to-bit" Assembler
(fa -- bitnr)

Converteer het vlag adres 'fa' naar het bitnummer 'bitnr'.

FILL Cross
(c-addr ud byte --)

Vul 'ud' bytes van het ByteForth codegeheugen vanaf 'c-addr' met 'byte'.

FLAG Cross
("name" -- flag)

Definieer een bit vlag met de naam 'name', tijdens het uitvoeren zet het zijn toestand als een forth vlag op de stack (10).
Geldige prefixen zijn: ADR TO SET CLEAR TOGGLE
Noot: De ADR prefix mag alleen in de assembler worden gebruikt.

FLASH ISP

(--)

De programmer werkt op het code (FLASH) geheugen van de cpu.

FOR Cross

(u --)

Zet de gegevens voor een FOR NEXT klaar zodat de lus ’u’ maal doorlopen wordt. (6). Zie ook: I’ J’ NEXT UNNEXT

FORGET Cross

("ccc" --)

Verwijder alle woorden uit het ByteForth woordenboek.
Letop, dit is niet standaard gedrag!

FORTH Cross

(--)

De woordenlijst met de woorden gedefineerd door je programma.

FROM Cross

("name" -- x)

Zet ’x’ de inhoud van ’name’ op de stack. Zie ook: VARIABLE SFR 2VARIABLE etc. Let op: Toegepast op een uitvoer-poort, wordt de data van de invoer-poort gelezen!

HELP Cross

("name" --)

Toon de glossary tekst, indien aanwezig, behorend bij het woord ’name’.

HERE Cross

(-- addr)

Geef het vrije adres ’addr’ (dubbel) in het geselecteerde ROM of RAM.

HEX Cross

(--)

Maak de getal basis hexadecimaal.

I Cross

(-- index)

Copieer de DO .. LOOP index I op de stack.
De LOOP index moet wel beschikbaar zijn (6).

I’ "i-tick" Cross

(-- index)

Copieer de FOR .. NEXT index I’ op de stack.
De NEXT index moet dan wel beschikbaar zijn (4).

IF Cross

(flag --)

Ga door met uitvoeren na IF als de ’flag’ true is. Is de ’flag’ false, ga dan door met executeren achter ELSE of THEN (2 tot 6).

IN Cross

("name" --)

Laad de file ’name’ vanuit de ’current directory’. Als ’name’ niet opgegeven is wordt de laatst gebruikte filenaam gebruikt. Zie ook: EDIT

INCLUDE Cross

("name.ext" --)

Compileer de file ’name.ext’.

INCR Cross

("name" --)

Verhoog de inhoud van ’name’ met 1. Zie ook: VARIABLE SFR etc.

INFO ISP

(--)

Toon de versie info van de programmeer software.

J Cross

(-- index)

Copieer de buitenste DO .. LOOP index J op de stack.
Deze loop index moet wel beschikbaar zijn (8).
Noot: Een geneste DO .. LOOP met FOR .. NEXT gaat mis!!

J’ "j-tick" Cross

(-- index)

Copieer de buitenste FOR .. NEXT index J’ op de stack.
De NEXT index moet dan wel beschikbaar zijn (6).
Noot: Een geneste FOR ... NEXT met DO .. LOOP gaat fout!!

KEY Cross

(-- char)

Wacht op de toetsaanslag ’char’ van het toetsenbord.

LEAVE Cross

(--)

Verlaat een DO LOOP constructie (2) (Niet een FOR .. NEXT).

LITERAL Cross

(x --)

Compileer ’x’ als een 8 bits getal (4).

LOCAL Cross

(x "name" --)

Definieer een ’local variable’ met de naam ’name’ en de waarde ’x’. Tijdens uitvoering zet ’name’ ’x’ op de stack. De waarde van de local ’name’ kan veranderd worden door de volgende prefixen: TO +TO INCR DECR CLEAR SET (8 to 14)

LOCALS| "locals-bar" Cross

(xn .. x1 name1 .. namex --)

Definieer de ’local variables’ met de namen ’name1’ t/m ’namex’ en de waarde ’x1’ t/m ’xn’ ’. Tijdens uitvoering zet ’namex’ ’xn’ op de stack. De waarde van de locals kan veranderd worden door de volgende prefixen: TO +TO INCR DECR CLEAR SET (8 to 14)

LOCK1 "lock-one" ISP

(--)

Zet lock-bit 1, die het verder programmeren van de microcontroller verbiedt.

LOCK2 "lock-two" ISP

(--)

Zet lock-bit 2, die het uitlezen van de microcontroller verbiedt.

LOOP Cross

(--)

Verlaag de lus index met een. Ga verder met uitvoeren na LOOP als de lus index de limiet bereikt. Ga verder na DO als de lus index kleiner is dan de limiet (14). Zie ook: DO I J LEAVE UNLOOP

LOW Assembler

(addr -- addr+1)

Deze operator is alleen geldig op 2VARIABLE(S), tel 1 bij een RAM offset of echt (dubbel) RAM adres ’addr’.

MACRO Cross

(modes "name" --)

Defineer een ByteForth macro genaamd ’name’. De stack i/o wordt beschreven door ’modes’. Elke macro heeft een ingebouwde optimizer. Let op: Een macro wordt altijd aan de macros wordenlijst toegevoegd.

MACRO: "macro-colon" Cross

(name --)

Definieer een tekst macro ’name’ er wordt geen code gegenereerd. De code wordt pas gegenereerd als ’name’ gebruikt wordt.

MACROS Cross

(--)

Vervang de eerste woordenlijst in de search-order door de MACROS woordenlijst.

MAIN Cross

(--)

Zet het adres van het laatst gedefinieerde woord in de resetvector van de AVR. De AVR start dan op met dit woord.

MANY Cross

(--)

Interpreteer de invoerbuffer opnieuw, tot een toets ingedrukt is. Letop, moet direct achter de uit te voeren Forth woorden staan!

MAP Cross

(dcode bits vars dstk rstk --)

Wijzig de AVR ByteForth geheugen layout volledig. Start code op het adres 'dcode'. Noot: Het code geheugen wordt opgegeven in 16-bits cellen! Er kunnen max. 64 bitvlaggen 'bits' gebruikt worden, en ook max. 64 variabelen. Voor de data-stack worden 'dstk' bytes gereserveerd en er wordt geprobeerd 'rstk' 16-bits cellen voor de return-stack te reserveren. Als er niet voldoende ruimte is wordt de return-stack kleiner. Er wordt een foutmelding gegenereerd als de return-stack kleiner is dan 8 cellen.
Het resterende RAM kan gebruikt worden voor arrays m.b.v. de woorden VARIABLES en 2VARIABLES en woorden die gemaakt zijn met CREATE of CREATE en DOES>, lees de uitleg bij deze woorden voor meer informatie. Ondersteund 30 of meer verschillende AVR microcontrollers.
Zie ook: MEMORY .MEMORY .FREE

MARK-OUT Cross

(--)

Mark end of AVR system constants for the ByteForth decompiler.

MDUMP "m-dump" Cross

(addr ud --)

Dump 'ud' bytes van het ByteForth data-geheugen (ram) vanaf 'addr'.

MEMORY Cross

(code bits vars --)

Verander de ByteForth geheugenindeling. Begin het programma vanaf het adres ’code’. Standaard is dit voor de AT90S2313 vanaf celadres 11. Let op: Het codegeheugen wordt geadresseerd in 16-bits cellen!! Er zijn max. 64 bit vlaggen ’bits’, standaard 16 stuks. Er zijn max. 64 variabelen ’vars’, standaard 32 stuks. Na het opstarten/resetten is er plaats voor 16 bit vlaggen, 32 variabelen en een returnstack van 32 adrescellen diep. Een voorbeeld: 35 8 47 MEMORY . De code start op (16bit-cel) adres 35. Er is ruimte voor 8 bit vlaggen, 47 variabelen en een returnstack van 24 adrescellen diep. Voor de ATmega8515 zijn de standaard waarden: 17 16 48 MEMORY . Inhoudend: Begin het programma vanaf adres 17, laat ruimte voor 16 bit vlaggen, 48 variabelen en een returnstack van 64 adrescellen diep. De resterende 304 bytes RAM kunnen gebruikt worden met de woorden VARIABLES, 2VARIABLES en door woorden die CREATE en CREATE .. DOES> gebruiken, kijk bij deze woorden voor een verdere uitleg. De gebruiker wordt gewaarschuwd als de returnstack minder dan 8 cellen diep is! Er worden 30 of meer verschillende AVR cpu’s ondersteund. Zie ook: .MEMORY .FREE en MAP

MS "m-s" Cross

(u --)

Wacht minimaal ’u’ milliseconden.

NEEDS Cross

("ccc" --)

Breidt ByteForth uit met de bibliotheek file ’ccc’.

NEXT Cross

(sys --)

Verlaag de lus index met een. Ga verder met uitvoeren na NEXT als de lus index nul wordt. Ga verder na FOR als de lus index niet gelijk aan nul is (6). Zie ook: FOR I’ J’ UNNEXT

OF Cross

(x1 x2 -- x1 |)

Als ’x1’ en x2’ niet gelijk zijn, gooi dan ’x2’ weg en ga verder met uitvoeren na ENDOF. Zijn ze gelijk verwijder dan beide waarden en ga door na OF (8 tot 22). Zie ook: CASE ENDCASE ENDOF

OPTIMIZER-OFF Cross

(--)

Compileer code met de optimizer uit.

OPTIMIZER-ON Cross

(--)

Compileer code met de optimizer aan.

ORDER Cross
(--)
Toon de ByteForth zoekvolgorde en het compilatie vocabulary.

OS "o-s" Cross
("name" --)
Start een DOS shell en voer het commando ’name’ uit.

P "program" ISP
(--)
Schrijf het gebruikte deel van de ByteForth buffer
naar de microcontroller. Zie ook: FLASH en EEPROM.

PAGE Cross
(--)
Maak het beeldscherm schoon.

POP Cross
(R: x "name" --)
Haal ’x’ de top van de return stack af en zet die in ’name’.
Zie ook: VARIABLE SFR 2VARIABLE etc.

PR "p-r" ISP
(--)
(Wis), programmeer en verifieer een microcontroller.

PRINTER Cross
(--)
Stuur uitvoer naar de printer en het standaard beeldscherm.

PRN1 "p-r-n-one" ISP
(--)
De ISP adapter is verbonden met printerpoort 1.
Er zijn ook: PRN2, PRN3 en PRN4

PROJECT Cross
("ccc" --)
Maak de file ’ccc’ met een standaard header. Start daarna
het editen van deze file ’ccc’. Informatie voor de header
kan aangepast worden en is te vinden in de file AVRF.CFG

PULSE Cross
(u "name" --)
Verander de toestand van het bit ’name’ gedurende u maal 10
microsec, herstel daarna de oorspronkelijke toestand.
Zie ook: BIT-SFR . Een voorbeeld: $90 BIT-SFR UIT de code
25 PULSE UIT keert het uitgangsbit 250 microseconden om en
zet het dan terug in de oorspronkelijke toestand.

PUSH Cross
(R: "name" -- x)
Zet ’x’ de inhoud van ’name’ op de return stack.
Zie ook: VARIABLE SFR 2VARIABLE etc.

Q. "q-dot" Cross
(q --)
Druk een ByteForth viervoudig getal ’q’ met teken af.

QU. "qu-dot" Cross
(qu --)
Druk een ByteForth viervoudig getal ’qu’ zonder teken af.

R "read" ISP
(--)
Lees het hele FLASH of EEPROM van de microcontroller naar de
ByteForth buffer. Zie ook: FLASH en EEPROM.

RAM Cross
(--)
Schakel de RAM adresruimte in. Zie ook: ROM, CDATA en UDATA.

RAMDUMP Cross
(addr ud --)
Dump ’ud’ bytes van het ByteForth data-geheugen (ram) vanaf ’addr’.

RAMTOP Assembler
(-- addr)
Geef het hoogste RAM adres ’addr’.

READ Cross
("name.ext" --)
Lees de ByteForth binary met de naam ’name.ext’ van disk.

READ-HEX Intel-Hex
("name.ext" --)
Lees de Intel-Hex file ’name.ext’ en maak er een ByteForth binary van.

REAL> "real-from" Assembler
(addr -- +n)
Converteer een (2)VARIABLE (echt) RAM adres ’addr’ naar de offset ’+n’.

REGISTER Cross
(-- x)
Definieer een register variabele met de naam ’name’. Dit is een VARIABLE achtig woord, maar sneller en korter in gebruik. Te gebruiken op plaatsen waar snelheid en/of RAM grootte erg belangrijk zijn (4).
Geldige prefixen zijn: ADR TO +TO CLEAR SET INCR DECR TOGGLE

REPEAT Cross
(--)
Ga verder met uitvoeren na BEGIN (2). Zie ook: BEGIN WHILE

RESET Cross
(--)
Voer code uit beginnend vanuit de reset-vector.

RESTART ISP
(--)
Herstart de via ISP aangesloten cpu door een reset puls.

ROM Cross
(--)
Schakel de ROM adresruimte in. Zie ook: CDATA, RAM en UDATA.

ROMDUMP Cross
(c-addr ud --)
Dump ’ud’ bytes van het ByteForth codegeheugen vanaf ’c-addr’.
Noot: ’c-addr’ en ’ud’ zijn 16-bit cellen en niet bytes!

RSP0 "r-s-p-zero" Assembler
(-- addr)
Geef het adres ’addr’ van de return-stack bodem.

RUN ISP
(--)
Laat de via ISP aangesloten chip vrij lopen.

RUNPOINT Cross
(--)
Zet een runpoint voor de simulator. Stop de tracer op dit punt.

S" "s-quote" Cross
("ccc" --)
Compileer een inline string in ByteForth. Dit woord heeft het woord INLINE$ in de ByteForth woordenlijst nodig (3+$).

SEE Cross

("ccc" --)

Toon het woord ’ccc’ in een door mensen leesbare vorm.

SELECT Cross

("ccc1" "ccc2" --)

Compileer het SELECT statement. ’ccc1’ is de selectie waarde en ’ccc2’ de string van woorden die uitgevoerd zal worden. Vergeet niet BEGIN-SELECT voor de eerste SELECT operator te gebruiken en END-SELECT na de laatste SELECT operator.
SELECT is een alternatief voor het CASE-statement (4+woorden).
Zie ook: BEGIN-SELECT END-SELECT
Noot: Er is maar een SELECT statement per regel toegestaan !!

SET Cross

("name" --)

Zet alle bits van ’name’ hoog.
Zie ook: VARIABLE SFR 2VARIABLE FLAG etc.

SET-CRYSTAL Cross

(n --)

Zet gewenste kristal frequentie in MHz. Zie ook: CRYSTAL?

SET-PAUSE ISP

(u --)

Zet de klokpuls verlengtijd, default is 100.

SETDIR "set-dir" Cross

(x "name" --)

Zet ’x’ in het richtingsregister die hoort bij de I/O-port ’name’.
Let op: Alleen geldig op uitvoer poorten (PORTD, etc).

SETUP Cross

(--)

Een pseudoniem voor SETUP-BYTEFORTH, alleen aanwezig om oude sources te ondersteunen.

SETUP-BYTEFORTH Cross

(--)

Compileer de minimale initialisatie voor een ByteForth systeem, installeer de data-stack en return-stack (8 tot 36).

SFR "s-f-r" Cross
(address "name" -- x)

Definieer een SFR ’address’ (I/O-adres) met de naam ’name’. Tijdens het uitvoeren wordt de toestand van de SFR ’x’ op de stack gezet (4).
Geldige prefixen zijn: ADR FROM TO +TO SET CLEAR
INCR DECR PUSH POP TOGGLE SETDIR
Noot1: De ADR prefix mag alleen in de assembler worden gebruikt.
Als ’address’ een uitgangsadres is, leest FROM het invoer adres (PINx) behorend bij (PORTx). SETDIR zet het (DDRx) richtingsregister behorend bij de uitvoer poort ’address’.
Noot2: Extended SFR-adressen worden nu ook ondersteund!
SFR adr. van $40 en hoger worden benaderd als EXT-SFR-adr.

SHELL Cross
(--)

Start een ’operating system shell’ b.v. de Norton Commander.

SLITERAL "s-literal" Cross
(addr u --)

Compileer ’addr u’ als een string constante. Dit woord heeft INLINE$ in de ByteForth woordenlijst nodig (3+$).

STOP ISP
(--)

Stop de via ISP aangesloten chip.

STRUCTURE Cross
("name" -- c-addr u)

Plaats ’c-addr u’ behorend bij ’name’ op de stack, dit zijn het begin-adres en lengte van de (data)structuur.
Let op: Mag niet ’compile time’ gebruikt worden !!
Zie ook: VARIABLE SFR 2VARIABLE etc.

T "cpu-type" ISP
(--)

Toon het typenummer van de aangesloten microcontroller.

TARGET Cross
(--)

Vervang de eerste woordenlijst in de search-order met de Forth woordenlijst. Ter ondersteuning van het ANSI standaard voorstel.

TEST Cross
(i*x "name" -- j*x)

Voer het woord ’name’ uit.

THEN Cross
(--)

Ga verder met uitvoeren na THEN (0). Zie ook: IF ELSE

TIMES Cross
(u --)
Executeer de invoerbuffer ’u’ keer of tot een toets ingedrukt is.
Letop, moet direct achter de uit te voeren Forth woorden staan!

TO Cross
(x "name" --)
Sla ’x’ op in ’name’. Zie ook: VARIABLE SFR 2VARIABLE etc.

TOGGLE Cross
("name" --)
Inverteer alle bits van ’name’. Zie ook: VARIABLE SFR BIT-SFR etc.

TYPE Cross
(addr u --)
Type de string vanaf ROM ’addr’ met de lengte ’u’ op het beeldscherm.

U. "u-dot" Cross
(u --)
Druk een ByteForth enkel getal ’u’ zonder teken af.

UDATA "u-data" Cross
(--)
Schakel de niet geinitialiseerde RAM adresruimte in.
Dit woord is een pseudoniem voor RAM.
Zie ook: ROM, RAM en CDATA.

UNLOOP "un-loop" Cross
(--)
Verwijder de LOOP gegevens (4) (Niet FOR .. NEXT). Zie ook: DO I LOOP

UNNEXT "un-next" Cross
(--)
Verwijder de gegevens van een FOR .. NEXT lus (2).

UNTIL Cross
(flag --)
Als ’flag’ true is ga dan verder met uitvoeren na UNTIL.
Als ’flag’ false is ga dan verder na BEGIN (6).

V "verify" ISP
(--)
Verifeer het gebruikte deel van de ByteForth buffer met
dat van de microcontroller. Zie ook: FLASH en EEPROM.

VALUE Cross

("name" -- x)

Definieer een value met de naam ’name’ zonder waarde. Tijdens het uitvoeren van naam wordt de inhoud van ’name’ op de stack geplaatst. Geldige prefixen zijn: ADR FROM TO +TO SET CLEAR INCR DECR POP PUSH TOGGLE
Noot: De ADR prefix mag alleen in de assembler worden gebruikt (4).

VALUES Cross

(u "name" -- x)

Definieer een array van ’u’ value’s met de naam ’name’.
Zet de inhoud ’x’ van het ’u’-de element op de stack.
Wees voorzichtig er is geen bereik controle op ’u’ ingebouwd.
Geldige prefixen zijn: ADR FROM TO +TO CLEAR SET INCR DECR
Noot: De ADR prefix mag alleen in de assembler worden gebruikt (12).

VARIABLE Cross

("name" -- addr)

Definieer een variabele met de naam ’name’. Zet het adres ’addr’ van de datacel op de stack. Geldige prefixen zijn:
ADR FROM TO +TO SET CLEAR INCR DECR POP PUSH TOGGLE
Noot: De ADR prefix mag alleen in de assembler worden gebruikt (4).
Zie ook: @ ! +!

VARIABLES Cross

(u "name" -- addr)

Definieer een array van ’u’ variabelen met de naam ’name’.
Zet het celadres ’addr’ behorend bij ’u’ op de stack.
Wees voorzichtig er is geen bereik controle op ’u’ ingebouwd.
Geldige prefixen zijn: ADR FROM TO +TO CLEAR SET INCR DECR
Noot: De ADR prefix mag alleen in de assembler worden gebruikt (14).
Zie ook: @ ! +!

VIDEO Cross

(--)

Stuur uitvoer naar het standaard beeldscherm.

WHAT Cross

(--)

Start de editor en positioneer de cursor op de regel waar de laatste fout optrad tijdens het compileren van die file.

WHILE Cross

(flag --)

Als ’flag’ true is ga dan verder na WHILE. Als ’flag’ false is ga dan verder met uitvoeren na REPEAT (2 to 6).

WORDS Cross

(--)

Toon alle woorden in de eerste vocabulary van de zoekvolgorde.

WORK Cross
(--)

Vervang de bovenste woordenlijst in de ’search order’ door de Forth woordenlijst. Alleen nog aanwezig om oude sources te ondersteunen!

WRITE Cross
("name.ext" --)

Schrijf de ByteForth binary met de naam ’name.ext’ naar disk.

WRITE-HEX Intel-Hex
("name.ext" --)

Schrijf de ByteForth binary in het Intel-Hex formaat naar de file ’name.ext’.

["left-bracket" Cross
(--)

Schakel de ByteForth interpreter in (0).

[’] "bracket-tick" Cross
("name" -- || -- xt)

Lees het volgende woord ’name’ en zet het token als dubbele literal in de code, alleen tijdens het compileren gebruiken. v.b. : AAP [’] NOOT ; Het token van NOOT wordt gecompileerd in het woord AAP (8).

[CHAR] "bracket-char" Cross
("abc" --)

Lees het volgende woord ’abc’ uit de invoerstroom en compileer de ASCII waarde van het eerste karakter ’char’ als een getal (4).

[ELSE] "bracket-else" Cross
(--)

Skip de invoerstroom inclusief het geneste voorkomen van [IF] ... [ELSE] ... [THEN] etc. totdat het woord [THEN] gevonden wordt. Er is een foutsituatie als er geen [THEN] gevonden wordt.

[IF] "bracket-if" Cross
(flag --)

Als de ’flag’ waar is doe dan niets. Skip anders de invoerstroom inclusief het geneste voorkomen van [IF] ... [ELSE] ... [THEN] etc. totdat het woord [ELSE] of [THEN] gevonden wordt.
Er is een foutsituatie als er geen [ELSE] of [THEN] gevonden wordt.

[THEN] "bracket-then" Cross
(--)

Doe niets. [THEN] is een immediate woord. Zie ook: [IF] en [THEN] .

\ "backslash" Cross
("ccc" --)
Skip commentaar tot het einde van de regel (0).

\G "slash-g" Cross
(--)
Sla de rest van de regel over. \G is een immediate woord en een pseudoniem voor \. Het wordt gebruikt om glossaries te genereren.

] "right-bracket" Cross
(--)
Start ByteForth compiler state (0).

5 Macro's

De AVR ByteForth compiler is een z.g.n. macrocompiler. Dit wil zeggen dat elk Forth woord, opgenomen in het basissysteem, een of meer AVR-instructies assembleert bij gebruik ervan. De optimalisator die in AVR ByteForth is opgenomen laat de macro's efficiënt in elkaar overvloeien. Overbodige instructies worden verwijderd, waardoor de gegenereerde code compacter en sneller wordt.

5.1 Werken met macro's

Het werken met een Forth macrocompiler is over het algemeen niet anders dan het werken met elke andere Forth. Elk woord assembleert zichzelf en zo ontstaat uiteindelijk een toepassing. Om er achter te komen hoeveel ruimte een macro inneemt, zul je de macro woordenlijst moeten bekijken. Als voorbeeld neem ik de macro - :

```
-                   "minus"                                    Macros
   ( x1 x2 -- x3 )
   Trek 'x2' van 'x1' af, het resultaat is 'x3' (10).
```

De tussen haakjes geplaatste 10 geeft de lengte van de macro aan (10 bytes). Dit betekent dat elk gebruik van "minus"10 bytes kost van de code ruimte! Je kan een veel gebruikte of grote macro ook importeren, als woord (subroutine) in het woordenboek opnemen. Hiervoor dient het commando ATOM. Door de CALL en RETURN die nodig zijn voor de aanroep, wordt de routine echter wel iets langer en langzamer. Als je ATOM - uitvoert, wordt het woord - in het woordenboek geimporteerd en neemt daar 10+2 bytes in beslag. Elk gebruik van - kost nu 2 bytes, hierdoor is het makkelijk uit te rekenen wanneer importeren met ATOM interessant wordt. Voor - is dit al bij twee keer: 10+10=20 bytes tegen 12+2+2=16 bytes.
Nu er een optimizer in ByteForth opgenomen is, wordt het gebruik van ATOM alleen nog lonend bij grote macro's die meer dan één maal aangeroepen worden. Alle woorden gemerkt met Macros behalve de hieronder genoemde, kunnen met behulp van ATOM in de ByteForth woordenlijst gezet worden.
Sommige woorden bestaan ook in de ASSEMBLER woordenlijst, ze zijn aanwezig voor het rekenwerk tijdens het assembleren van code definities.

5.2 Speciale macro's

Er zijn enkele speciale macro's die niet met het woord ATOM te importeren zijn. Dit zijn >R, R> en R@. Ze zijn gemerkt met een bit in de header, zodat ze door de ByteForth compiler alleen als macro te gebruiken zijn. Het woord INLINE$ is op een andere manier bijzonder, hij is niet als macro te gebruiken, maar alleen als subroutine met behulp van ATOM.
SETUP-BYTEFORTH is een macro die vanwege zijn functie opgenomen is in de compiler. De macro CATCH is terug te vinden in de bibliotheek file: CATCH.FRT

5.3 Zelf macro's definiëren

Het is vanaf release 2.00 mogelijk zelf nieuwe macro's toe te voegen aan ByteForth. Een macro is een code definitie waarvan zijn gedrag op de stack intern gedocumenteerd is. Met behulp van een speciale notatie wordt die opgegeven. Wat die getallen betekenen wordt in een volgende alinea uitgelegd. Een voorbeeld:

```
$11 MACRO 2+   ( x - x+2 )
    R16 X+ LD,           \ Pop x van de stack
    R16 2 ADDI,          \ Verhoog x met 2
    -X R16 ST,           \ Push x weer naar de stack
    RET,
END-CODE
```

Wat gebeurt hier? `$11 MACRO 2+` maakt een nieuwe macro-definitie, deze bestaat alleen binnen de crosscompiler op de PC. Het eerste getal `$1x` vertelt de compiler bij gebruik van de macro dat er een item van de stack gepopt wordt. Het tweede getal `$x1` laat de compiler weten dat er achteraf ook een getal op de stack teruggezet wordt. De optimizer weet hierdoor hoe het de nieuwe macro 2+ kan optimaliseren. Het is natuurlijk erg belangrijk dat de opgave correct wordt ingegeven, anders gaat de optimizer de mist in.

5.4 Tekstmacro's in ByteForth

Voor de leesbaarheid van een programma, kan het handig zijn om korte stukken code een naam te geven, zonder er een aparte subroutine van te maken. Op zo'n moment is een tekstmacro op zijn plaats. Een tekstmacro bestaat alleen binnen de compiler, pas bij gebruik wordt er code gegenereerd. Tekstmacro's kunnen zowel in colon-definities als in andere tekst-macro's toegepast worden. Enkele voorbeelden van een tekstmacro: Voeg >= aan ByteForth toe.

```
MACRO: >=    < 0= ;
```

Een korte uitvoer routine levert zo efficiënte code op.

```
P1: SFR LEDS
MACRO: TO-LEDS  INVERT TO LEDS ;
```

Een typische Nijhof factorisatie.

```
MACRO: }  EXIT  THEN ;
```

5.5 Over de optimizer

De mogelijkheden van de optimizer zijn in een tabel naast elkaar geplaatst. Hierdoor zijn de opties in een keer te overzien.

Nummer	Invoer actie	Uitvoer actie
0	Doe niets	Doe niets
1	Pop een getal van de stack	Push getal op de stack
2	Pop twee getallen van de stack	Push twee getallen op de stack
3	Pop drie getallen van de stack	Push drie getallen op de stack
4	Pop vier getallen van de stack	Push vier getallen op de stack
5	Verwijder getal van de stack	Push Literal op de stack
6	Verwijder twee getallen van stack	Push 2Literal op de stack
7	Pop getal naar variable	Push Variable op de stack
8	Pop twee getallen naar variable	Push 2Variable op de stack
9	Pop getal naar I/O-bit	Push I/O-bit op de stack
A	Pop getal naar vlag	Push vlag op stack
B	Pop getal naar carry-vlag	Push carry-vlag op de stack
C	Move getal naar accu (R16)	Move accu (R16) naar TOS
D	Pop getal naar register	Push register op stack
E	Assembleer RET(urn)	Call geassembleerd
F	Niet gebruikt	DROP

5.6 MACROS woordenlijst

```
!                   "store"                                 Macros
   ( x addr -- )
   Sla 'x' op in adres 'addr' (8).
   Noot: Mag alleen gebruikt worden op 'echte' RAM adressen.

*                   "star"                                  Macros
   ( x1 x2 -- x3 )
   Vermenigvuldig 'x1' en 'x2', het resultaat is het produkt 'x3' (26).

+                   "plus"                                  Macros
   ( x1 x2 -- x3 )
   Tel 'x1' en 'x2' bij elkaar op, het resultaat is 'x3' (8).

+!                  "plus-store"                            Macros
   ( x addr -- )
   Verhoog de inhoud van het adres 'addr' met 'x' (12).
   Noot: Mag alleen gebruikt worden op 'echte' RAM adressen.

-                   "minus"                                 Macros
   ( x1 x2 -- x3 )
   Trek 'x2' van 'x1' af, het resultaat is 'x3' (10).

-ROT                "minus-rote"                            Macros
   ( x1 x2 x3 -- x3 x1 x2 )
   Roteer de top van de stack 'x3' naar de derde positie (12).

/MS                 "slash-m-s"                             Macros
   ( u -- )
   Wacht tenminste 'u' maal 0.1 milliseconden (bij een 4 MHz Xtal) (16).
   Zie ook de file: ..\LIB\MS.FRT

/STRING             "slash-string"                          Macros
   ( $addr1 u1 n -- $addr2 u2 )
   Knip aan de voorkant van de string '$addr1' 'u1' er 'n' karakters
   vanaf. Het resultaat is de string '$addr2' 'u2' (22).

0<                  "zero-less"                             Macros
   ( n -- flag )
   Zet een true vlag op de stack als 'n' kleiner dan nul is (8).

0=                  "zero-equals"                           Macros
   ( x1 -- flag )
   Zet een true vlag op de stack als 'x' nul is, anders false (12).
```

0> "zero-greater" Macros
(n -- flag)
Zet een true vlag op de stack als ’n’ groter is dan nul (10).

1+ "one-plus" Macros
(x1 -- x2)
Tel 1 op bij ’x1’, ’x2’ is het resultaat (6)

1- "one-minus" Macros
(x1 -- x2)
Trek 1 af van ’x1’, ’x2’ is het resultaat (6)

2! "two-store" Macros
(dx addr --)
Sla de dubbel ’dx’ op in adres ’addr’ (12).
Noot: Mag alleen gebruikt worden op ’echte’ RAM adressen.

2* "two-star" Macros
(x1 -- x2)
Schuif ’x1’ een bit naar links, het resultaat is ’x2’ (6).

2/ "two-slash" Macros
(x1 -- x2)
Schuif ’n1’ een bit naar rechts, het resultaat is ’n2’,
’n2’ zal hetzelfde teken hebben als ’n1’ (6).

2>R "two-to-r" Macros
(x1 x2 --)
Zet ’x1’ en ’x2’ op de returnstack.
Gelijk aan de code: SWAP >R >R (8).

2@ "two-fetch" Macros
(addr --- dx)
Zet ’dx’ de inhoud van de dubbelecel ’addr’ op de stack (12).
Noot: Mag alleen gebruikt worden op ’echte’ RAM adressen.

2DROP "two-drop" Macros
(dx --)
Verwijder de dubbel ’dx’ van de top (2).

2DUP "two-dupe" Macros
(dx -- dx dx)
Copieer de dubbel ’dx’ bovenop de stack (10).

2NIP "two-nip" Macros
(dx1 dx2 -- dx2)
Verwijder het tweede dubbele item ’dx1’ van stack (10).

2OVER "two-over" Macros
(dx1 dx2 -- dx1 dx2 dx1)
Copieer de dubbel ’dx1’ naar de top van de stack (12).

2R> "two-r-from" Macros
(-- x1 x2)
Haal ’x1’ en ’x2’ van de returnstack af.
Gelijk aan de code: R> R> SWAP (8).

2R@ "two-r-fetch" Macros
(-- x1 x2)
Copieer ’x1’ en ’x2’ van de returnstack zonder ze te verwijderen..
Gelijk aan de code: R> R> 2DUP >R >R SWAP (12).

2ROM@ "two-rom-fetch" Macros
(c-addr -- dx)
Zet de dubbele inhoud ’dx’ uit het byte-address ’c-addr’ op de stack (18).

2SWAP "two-swap" Macros
(dx1 dx2 -- dx2 dx1)
verwissel de dubbel ’dx1’ en ’dx2’ (16).

< "less" Macros
(n1 n2 -- flag)
Zet een true vlag op de stack als ’n1’ kleiner is dan ’n2’ (16).

<> "not-equal" Macros
(x1 x2 -- flag)
Zet een true vlag op de stack als ’x1’ ongelijk is aan ’x2’ (12).

= "equal" Macros
(x1 x2 -- flag)
Zet een true vlag op de stack als ’x1’ gelijk is aan ’x2’ (14).

> "greater" Macros
(n1 n2 -- flag)
Zet een true vlag op de stack als ’n1’ groter is dan ’n2’ (16).

>R "to-r" Macros
(x --)
Zet ’x’ op de returnstack (4).

?DUP "question-dupe" Macros
(x -- 0 | x x)
Copieer de top van de stack ’x’ alleen als ’x’ niet nul is (10).

?NEGATE "question-negate" Macros
(n1 n2 -- n3)

Keer het teken van ’n1’ om als ’n2’ kleiner dan nul is,
het resultaat is ’n3’ (14).

@ "fetch" Macros
(addr --- x)

Zet ’x’ de inhoud van ’addr’ op de stack (8).
Noot: Mag alleen gebruikt worden op ’echte’ RAM adressen.

@+ "fetch-plus" Macros
(a1 --- a2 x)

Lees de inhoud van ’a1’ en laat ’a2’ en ’x’ de inhoud
van a1 op de stack achter. ’a2’ is ’a1+1’ (12).
Noot: Mag alleen gebruikt worden op ’echte’ RAM adressen.

ABS "abs" Macros
(n -- u)

’u’ is de absolute waarde van ’n’ (10).

AND Macros
(x1 x2 -- x3)

’x3’ is een bitgewijze logische and van ’x1’ en ’x2’ (8).

CELL+ "cell-plus" Macros
(x1 -- x2)

Tel 1 op bij ’x1’ het resultaat is ’x2’ (6).

CELLS Macros
(x1 -- x2)

Zet ’x1’ om in ’x2’, ’x2’ is het aantal adres units
nodig voor ’x1’ cellen (een cel is hier 1) (0).

CMOVE "c-move" Macros
(addr1 addr2 u --)

Verplaats ’u’ bytes vanaf RAM ’addr1’ naar RAM ’addr2’ (30).
Noot: Mag alleen op ’echte’ RAM adressen worden gebruikt.

COUNT Macros
($addr1 -- addr2 u)

Converteer het byte-adres ’$addr1’ tot ’addr2’ en ’u’, ’addr2’
is het begin van een string van ’u’ karakters (16).

CSWAP "c-swap" Macros
(x1 -- x2)

Verwissel de hoge en lage nibble van ’x1’, het resultaat is ’x2’ (6).

D+ "d-plus" Macros
(d1 d2 -- d3)
Tel de dubbele getallen ’d1’ en ’d2’ op, ’d3’ is de uitkomst (16).

D- "d-minus" Macros
(d1 d2 -- d3)
Trek ’d2’ van ’d1’ af, de uitkomst is ’d3’ (20).

D2* "d-two-star" Macros
(dx1 -- dx2)
Schuif ’dx1’ een bit naar links, het resultaat is ’dx2’ (12).

D2/ "d-two-slash" Macros
(d1 -- d2)
Schuif ’d1’ een bit naar rechts, het resultaat is ’d2’,
’d2’ zal hetzelfde teken hebben als ’d1’ (12).

DABS "d-abs" Macros
(d -- du)
’du’ is de absolute waarde van ’d’ (20).

DEPTH Macros
(-- +n)
Zet de diepte van de stack ’+n’ bovenop (6).

DNEGATE "d-negate" Macros
(d1 -- d2)
Keer het teken van de dubbel ’d1’ om, het resultaat is ’d2’ (16).

DROP Macros
(x --)
Verwijder de top van de stack ’x’ (2).

DU2/ "d-u-two-slash" Macros
(du1 -- du2)
Schuif ’du1’ een bit naar rechts, het resultaat is ’du2’.
De meest significante bits van ’du2’ worden met nul gevuld (12).

DUM* "d-u-m-star" Macros
(du1 du2 -- qu)
Vermenigvuldig ’du1’ met ’du2’, het resultaat is het product ’qu’
dat vier cellen in beslag neemt. Alle waarden zijn unsigned (50).

DUM/MOD "d-u-m-slash-mod" Macros

(uq ud1 -- ud2 ud3)

Deel ’uq’ door ’ud1’, het resultaat is het quotient ’ud3’ en de rest ’ud2’. Alle waarden zijn unsigned. Er onstaat een speciale conditie als ’ud1’ nul is, of het quotient buiten het bereik van een unsigned dubbel integer is. ’uq’ is een getal van vier cellen (82).

DUP "dupe" Macros

(x -- x x)

Copieer de top van de stack ’x’ (4).

EXECUTE Macros

(i*x xt -- j*x)

Voer het execution token ’xt’ uit, ’i*x’ is de stack voor uitvoering van ’xt’, ’j*x’ is de stack na uitvoering van ’xt’ (8).

FILL Macros

(addr u char --)

Vul ’u’ bytes in het RAM geheugen vanaf ’addr’ met ’char’ (18).
Noot: Mag alleen op ’echte’ RAM adressen worden gebruikt.

IDLE Macros

(--)

Zet een AVR cpu in de idle toestand (10).

INLINE$ "inline-string" Macros

(inline$ -- $addr u)

Geef het begin van een inline string ’$addr’ en zijn lengte ’u’ (34).
Noot: Mag alleen worden gebruikt via de import functie ATOM !

INTERRUPT-OFF Macros

(--)

Deactiveer het AVR interrupt mechanisme (2).

INTERRUPT-ON Macros

(--)

Activeer het AVR interrupt mechanisme (2).

INVERT Macros

(x1 -- x2)

Inverteer alle bits van ’x1’ het resultaat is ’x2’ (6).

KICKSTART "kick-start" Macros

(--)

Herstart (Spring naar reset) de ByteForth applicatie (6).

LSHIFT "l-shift" Macros
(x1 n -- x2)
Schuif ’x1’ ’n’ posties naar links, het resultaat is ’x2’.
De vrij gekomen plaatsen worden gevuld met nul (18).

M+ "m-plus" Macros
(dx1 n -- dx2)
Tel de dubbel ’dx1’ en ’n’ bij elkaar op, het resultaat is ’dx2’ (24).

MAX Macros
(n1 n2 -- n1 | n2)
Laat de grootste van de signed getallen ’n1’ en ’n2’ achter (12).

MIN Macros
(n1 n2 -- n1 | n2)
Laat de kleinste van de signed getallen ’n1’ en ’n2’ achter (12).

MS "m-s" Macros
(u --)
Wacht tenminste ’u’ milliseconden (bij een 4 MHz Xtal) (16).
Zie ook de file: ..\LIB\MS.FRT

NEGATE Macros
(n1 -- n2)
Keer het teken van ’n1’ om, het resultaat is ’n2’ (6).

NIP Macros
(x1 x2 -- x2)
Verwijder het tweede item van de stack ’x1’ (4).

NOOP "no-op" Macros
(--)
Wacht 1 klok-tik. doe verder niets (2).

NOT Macros
(x1 -- x2)
Inverteer alle bits van ’x1’ het resultaat is ’x2’
(dit is een pseudoniem van INVERT) (6).

OR Macros
(x1 x2 -- x3)
’x3’ is een bitgewijze logische or van ’x1’ en ’x2’ (8).

OVER Macros
(x1 x2 -- x1 x2 x1)
Copieer het tweede item van de stack ’x1’ naar de top (8).

PICK Macros
(xu ... x1 x0 u -- xu ... x1 x0 xu)
Verwijder ’u’ en copieer ’xu’ naar de top van de stack (12).

POPALL "pop-all" Macros
(--)
Herstel 17 registers en originele stack. Dit is nodig om een high-level AVR ByteForth interrupt te kunnen uitvoeren (36).

POWERSAVE "power-save" Macros
(--)
Zet sommige AVR cpu’s in de power-save (sleep) mode. Dit doet hetzelfde als SLEEP op enkele andere AVR cpu’s (8).

PUSHALL "push-all" Macros
(--)
Bewaar 17 registers en maak de stack 10 bytes diep. Dit is nodig om een high-level AVR ByteForth interrupt te kunnen uitvoeren (38).

R> "from-r" Macros
(-- x)
Haal ’x’ van de returnstack af (4).

R@ "r-fetch" Macros
(-- x)
Lees ’x’ van de returnstack zonder hem te verwijderen (6).

RESET-WATCHDOG "reset-watch-dog" Macros
(--)
Reset de AVR watchdog timer aan boord van de AVR (2).

ROLL Macros
(xu ... x1 x0 u -- x2 ... x1 x0 xu)
Verwijder ’u’ en roteer het ’u+1’ item naar de top van de stack (26).

ROM@ "rom-fetch" Macros
(c-addr -- x)
Zet de inhoud ’x’ uit het byte-address ’c-addr’ op de stack (10).

ROT "rote" Macros
(x1 x2 x3 -- x2 x3 x1)
Roteer het derde item op de stack ’x1’ naar de top (12).

RSHIFT "r-shift" Macros
(x1 +n -- x2)
Schuif ’x1’ ’n’ posties naar rechts, het resultaat is ’x2’. De vrij gekomen plaatsen worden gevuld met nul (18).

S>D "s-to-d" Macros
(x -- dx)
Converteer de single ’x’ tot de dubbel ’dx’ (12).

SLEEP Macros
(--)
Zet een AVR cpu in de power-down (sleep) toestand (8).

SPLIT Macros
(char -- lownibble highnibble)
Splits het karakter ’char’ in een ’lownibble’ en een ’highnibble’ (14).

SWAP Macros
(x1 x2 -- x2 x1)
Verwissel ’x1’ en ’x2’ op de stack (8).

TUCK Macros
(x1 x2 -- x2 x1 x2)
Copieer de top van de stack ’x2’ onder het tweede item ’x1’ (10).

U2/ "u-two-slash" Macros
(u1 -- u2)
Schuif ’u1’ een bit naar rechts, het resultaat is ’u2’.
De meest significante bits van ’u2’ worden met nul gevuld (6).

U< "u-less" Macros
(u1 u2 -- flag)
Zet een true vlag op de stack als ’u1’ kleiner is dan ’u2’ (10).

U> "u-greater" Macros
(u1 u2 -- flag)
Zet een true vlag op de stack als ’u1’ groter is dan ’u2’ (10).

UM* "u-m-star" Macros
(u1 u2 -- ud)
Vermenigvuldig ’u1’ en ’u2’, resultaat is het dubbele produkt ’du’ (28).

UM/MOD "u-m-slash-mod" Macros
(ud u -- ur uq)
Deel de unsigned dubbel ’du’ door de single ’u’, het resultaat is
de unsigned rest ’ur’ en het quotient ’uq’ (44).

UMAX "u-max" Macros
(u1 u2 -- u1 | u2)
Laat de grootste van de unsigned getallen ’u1’ en ’u2’ achter (12).

UMIN "u-min" Macros
(u1 u2 -- u1 | u2)

Laat de kleinste van de unsigned getallen ’u1’ en ’u2’ achter (12).

WATCHDOG-OFF "watch-dog-off" Macros
(--)

Deactiveer de AVR watchdog timer (12).

WATCHDOG-ON "watch-dog-on" Macros
(+n --)

Activeer de watchdog timer & zet de watchdog reset interval
timer op een van acht voorgedefinieerde waarden, zie hieronder (10).
De getoonde tijden zijn typical waarden bij een Vcc=5.0V):
0 = 15 ms 2 = 60 ms 4 = 240 ms 6 = 970 ms
1 = 30 ms 3 = 120 ms 5 = 490 ms 7 = 1900 ms

WITHIN Macros
(x1 x2 x3 -- flag)

Zet een true vlag op de stack, als ’x1’ in het bereik
ligt van ’x2’ tot ’x3-1’ (16).

XOR "x-or" Macros
(x1 x2 -- x3)

’x3’ is een bitgewijze logische xor van ’x1’ en ’x2’ (8).

6 Bibliotheek

ByteForth bezit een bibliotheek met geteste code. Dit is code uit toepassingen die voor hergebruik in aanmerking komt. Al werkend met ByteForth, kun je deze bibliotheek steeds verder uitbreiden met nieuwe stukken code.

6.1 Eenvoudig gebruik

Voor eenvoudig gebruik las je een complete bibliotheekfile in een nieuw programma in. Wel even controleren of de door de bibliotheek gebruikte poorten en adressen niet dubbel gebruikt worden in de programma code. In b.v. de code van de file LCD.FRT worden 6 bits van poort-B gebruikt, de rest van de programmacode kan deze bits niet zomaar gebruiken.

6.2 Aanpassen code

Bij een geavanceerder gebruik van de bibliotheek ga je de code aanpassen voor toepassing in een nieuw programma. Je kan de routines herschrijven voor een andere hardware configuratie. Doe dit altijd met beleid en test het zo veel mogelijk. De bibliotheek file NUMBERS.FRT bevat code voor het afdrukken van getallen in verschillende talstelsels. Is dit niet nodig dan kan de file enigzins vereenvoudigd worden. Doe dit alleen als de code voor een toepassing te groot is geworden voor de Flash EPROM van de gekozen AVR-chip.

6.3 Een lijst van bibliotheek files

I2C-PRIM.FRT	De basisbouwstenen voor I2C-routines op blz. 68.
I2C-8574.FRT	Deze en andere I2C-chips zijn opgenomen op blz. 69.
I2C24C65.FRT	I2C EEPROM-routines van 8 kByte t/m 64 kByte op blz. 71.
ADC549IP.FRT	Seriële ADC omzetter TLC549IP op blz. 72.
TRACER.FRT	Tracer met breekpunten (via RS232) op blz. 73.
RS232.FRT	Diverse RS232 i/o routines op blz. 73 en verder.
LCD.FRT	Vier bits LCD uitvoer op blz. 76.
NUMBERS.FRT	Universele getal omzet- en afdrukroutines op blz. 77.
ARITH.FRT	Extra rekenkundige mogelijkheden op blz. 79.
DOUBLE.FRT	Rekenen en vergelijken met 16 bits getallen op blz. 80.
MS.FRT	MS routine voor verschillende klok frequenties op blz. 81.
RANDOM.FRT	Twee verschillende random generators op blz. 81.
CATCH.FRT	De CATCH foutopvang methode op blz. 82.
KEYB1.FRT	3x4 en 3x4+1 toetsenbord aansturing op blz. 82 en. 82.
RC5.FRT	RC5 decodeer software op interrupt op blz. 83.
MIDI.FRT	Midi i/o routines op blz. 85.
MUSIC.FRT	Speel muziek, 22 noten, met slechts een i/o bit op blz. 83.
BAMBOE.FRT	8-bit bamboe i/o op blz. 83.
7SEGM.FRT	Zevensegment display uitvoer via drie bits op blz. 84.
GLCD.FRT	Grafisch LCD 122x32 pixels op blz. 75.
LETTERS.FRT	Letterset voor grafisch LCD 122x32 op blz. 75.
TASKER.FRT	Multitasking voor alle AVR-typen met RAM op blz. 72.
GP2D02.FRT	IR afstandmeter van 10cm tot 80cm op blz. 85.
TLC834CN.FRT	ADC met 4-ingangen en seriële interface op blz. 72.
CP-ADC.FRT	ADC via de ingebouwde comparator op blz. 84.
I2C-LM75.FRT	Temperatuursensor via I2C op blz. 71.
BCD.FRT	BCD-conversie operatoren op blz. 84.
PIR.FRT	Code voor een passieve infraroodsensor op blz. 84.

6.4 Bibliotheek woordenlijst

6.4.1 I2C-PRIM.FRT

```
(BYTE-UIT)                                                    I2C-Prim
   ( x -- )
   Verzend byte 'x' via de I2C bus.

AANWEZIG?                                                     I2C-Prim
   ( dev-id -- vlag )
   De vlag is true als het device geadresseerd met 'dev-id'
   aanwezig is op de I2C bus en anders is hij false.

ACK-BIT                                                       I2C-Prim
   ( -- )
   Genereer een I2C acknowledge.

ACK?                                                          I2C-Prim
   ( -- vlag )
   Test het inkomende bit. Geef true als het een acknowledge is
   en anders false.

BYTE-IN                                                       I2C-Prim
   ( -- x )
   Ontvang byte 'x' via de I2C bus.

BYTE-UIT                                                      I2C-Prim
   ( x -- )
   Verzend byte 'x' via de I2C bus en genereer niets als de
   geadresseerde device niet reageert met een acknowledge.

NACK-BIT                                                      I2C-Prim
   ( -- )
   Genereer een I2C negatieve acknowledge.

SETUP-I2C                                                     I2C-Prim
   ( -- )
   Initialiseer de I2C bus.

START-BIT                                                     I2C-Prim
   ( -- )
   Genereer een I2C startconditie.

STOP-BIT                                                      I2C-Prim
   ( -- )
   Genereer een I2C stopconditie.
```

WACHT I2C-Prim
(--)
Wacht ongeveer 4.7 microsec. bij een 4 MHz kristal.
Er zijn versies voor 4, 8, 10 en 11.059 MHz.

6.4.2 I2C-8574.FRT

!BYTE "store-byte" I2C-8574
(x chip --)
Zet data ’x’ in de PCF8574 met als device-id ’chip’.

@BYTE "fetch-byte" I2C-8574
(chip -- x)
Lees data ’x’ van de PCF8574 met als device-id ’chip’.

6.4.3 I2C-8583.FRT

!KLOK I2C-8583
(x addr --)
Zet data ’x’ op adres ’addr’ van de PCF8583.

@KLOK I2C-8583
(addr -- x)
Lees data ’x’ van adres ’addr’ uit de PCF8583.

KLOK I2C-8583
(mode --)
Zet klok ’mode’ van de PCF8583. Geldige modes zijn: NORMALE en WEK

LEES-DATUM I2C-8583
(-- dag mnd djaar)
Lees de datum ’dag’, ’mnd’ en ’djaar’ (een dubbel getal).

LEES-KLOK I2C-8583
(-- sec min uur)
Lees de tijd ’sec’, ’min’ en ’uur’.

LEES-WEKKER I2C-8583
(-- sec min uur)
Lees de wektijd als ’sec’, ’min’ en ’uur’.

SETUP-KLOK I2C-8583
(--)
Initialiseer een normale 24 uurs klok zonder wekker.

WEKKER I2C-8583

(mode --)

Zet de wekkermode 'mode', geldige modes zijn:
GEEN wekker = Geen wekker functie.
DAGELIJKSE wekker = Wek elke dag van de week.
WEEKDAG wekker = Wek alleen op deze dag.
DATUM wekker = Wek alleen op deze datum.

WEKKER? I2C-8583

(-- vlag)

De 'vlag' is true als de wekker afgelopen is.

ZET-DATUM I2C-8583

(dag mnd djaar --)

Zet de datum week'dag', 'mnd' en 'djaar' (een dubbel getal).

ZET-KLOK I2C-8583

(sec min uur --)

Zet de tijd 'sec', 'min' en 'uur'.

ZET-WEK-DATUM I2C-8583

(dag mnd djaar --)

Zet wek datum 'dag', 'mnd'. Het jaar 'djaar' wordt alleen gebruikt voor de schrikkeljaar correctie.

ZET-WEK-TIJD I2C-8583

(sec min uur --)

Zet wektijd op 'sec', 'min' en 'uur'.

6.4.4 I2C-8591.FRT

ADC I2C-8591

(+n -- u)

Lees ADC ingang '+n', 'u' is het resultaat van de conversie.

DAC I2C-8591

(u --)

Zet de DAC-uitgang op een waarde die overeenkomt met 'u'.

DAC? I2C-8591

(-- vlag)

Deze flag is gezet als de DAC gebruikt is. Zet hem op nul als je de DAC gedurende de ADC conversies uit wilt zetten.

6.4.5 I2C24C02.FRT

LEES-BYTE I2C24C02
(addr -- x)

Lees de data ’x’ vanaf het EEPROM adres ’addr’. ByteForth heeft ook library’s voor de: 24C01 en de 24C02.

SCHRIJF-BYTE I2C24C02
(x addr --)

Schrijf de data ’x’ naar het EEPROM adres ’addr’. ByteForth heeft ook library’s voor de: 24C01 en de 24C02.

6.4.6 I2C24C16.FRT

LEES-BYTE I2C24C16
(d-addr -- x)

Lees de data ’x’ vanaf het dubbele EEPROM adres ’addr’. ByteForth heeft ook library’s voor de: 24C04, 24C08 en de 24C16.

SCHRIJF-BYTE I2C24C16
(x addr --)

Schrijf de data ’x’ naar het dubbele EEPROM adres ’addr’. ByteForth heeft ook library’s voor de: 24C04, 24C08, 24C16 en de 24C65.

6.4.7 I2C24C65.FRT

LEES-BYTE I2C24C65
(addr -- x)

Lees de data ’x’ vanaf het dubbele EEPROM adres ’addr’. Deze ByteForth library is ook bruikbaar voor de: 24C32, 24C64, 24C128 en de 24C256.

SCHRIJF-BYTE I2C24C65
(x addr --)

Schrijf de data ’x’ naar het dubbele EEPROM adres ’addr’. Deze ByteForth library is ook bruikbaar voor de: 24C32, 24C64, 24C128 en de 24C256.

6.4.8 I2C-LM75.FRT

BOVENGRENS I2C-LM75
(dx chip --)

Zet de bovengrens ’d’ van de thermostaatfunctie van ’chip’.

CONFIGURATIE I2C-LM75
(x chip --)

Zet ’x’ in het configuratie register van ’chip’.

ONDERGRENS I2C-LM75

(d chip --)

Zet de ondergrens ’d’ van de thermostaatfunctie van ’chip’.

TEMPERATUUR I2C-LM75

(chip -- d)

Vraag de tenperatuur ’d’ van de chip ’chip’. De temperatuur wordt afgegeven in halve graden per bit, waar 0 nul graden is, en het bereik loopt van -55 tot +125 graden.

6.4.9 ADC549IP.FRT

ADC ADC549IP

(-- u)

Lees de ADC ingang, ’u’ is het resultaat van deze conversie.

SETUP-ADC ADC549IP

(--)

Initialiseer de ADC interface.

6.4.10 TLC834CN.FRT

ADC ADC834CN

(+n -- u)

Lees de ADC ingang ’+n’, ’u’ is het resultaat van deze conversie.

SETUP-ADC ADC834CN

(--)

Initialiseer de ADC-interface voor de ADC834CN.

6.4.11 TASKER.FRT

ACTIVATE Tasker

(xt task-nr --)

Initialiseer een achtergrondtaak, dat moet altijd vanuit de basistaak gebeuren. Vanuit de basistaak, mag een achtergrondtaak altijd vervangen worden (44 tot 64 bytes).

COMPARE-ON Tasker

(--)

Start timer-1 met een prescaler van 8, het timerregister wordt met 500 geladen zodat er elke millisec. een interrupt gegeven wordt. De timer-1 compare-A interrupt wordt hiervoor gebruikt (26).

```
MS1                                                          Tasker
   ( u -- )
   Wacht u milliseconden er wordt een timerinterrupt gebruikt
   om MS1, MS2 en MS3 ook in een multitasking omgeving correct
   te laten functioneren.

PAUSE                                                        Tasker
   ( -- )
   Schakel om naar een volgende taak. De lengte van de
   routine is 36 tot 66 bytes, afhankelijk van het AVR-type.

TIMER                                                        Tasker
   ( -- )
   De timer interrupt routine houdt drie afzonderlijke MS
   timers bij. Hij gebruikt ~0.65% van de cpu-tijd (28).
```

6.4.12 TRACER.FRT

```
.REGISTERS         "dot-registers"                           Tracer
   ( -- )
   Toon de belangrijkste registers van de AVR.

.S                 "dot-s"                                   Tracer
   ( -- )
   Copieer en toon alle gegevens op de datastack.

BREAKPOINT                                                   Tracer
   ( -- )
   Zet een breakpunt, toon de registers en druk de stack af,
   wacht daarna op een toetsaanslag.

TRACER-SETUP                                                 Tracer
   ( -- )
   Configureer de tracer en zet een breakpunt.
```

6.4.13 RS232.FRT & RS232M.FRT

De file RS232.FRT bevat routines voor een standaard AVR met een UART, de file RS232M.FRT bevat dezelfde routines voor ATmega AVR's met een USART.

```
RS232-EMIT                                                   RS232
   ( char -- )
   Verzend het karakter 'char' via de RS232.
```

```
RS232-KEY                                                    RS232
   ( -- char )
   Wacht tot een karakter 'char' via RS232 ontvangen is.

RS232-KEY?                                                   RS232
   ( -- flag )
   Is via RS232 een karakter ontvangen, dan is de 'flag' true.

RS232-RTYPE        "rs232-ram-type"                          RS232
   ( $addr u -- )
   Verstuur 'u' karakters vanaf RAM adres 'addr' via de RS232.

RS232-TYPE                                                   RS232
   ( $addr u -- )
   Verzend 'u' karakters vanaf ROM adres '$addr' via de RS232.

SETUP-RS232                                                  RS232
   Initialiseer de UART op een RS232 baudrate van 9600 baud.
   Er zijn versies voor 4, 8 en 11.059 MHz.
```

6.4.14 RS232S.FRT

Dit is een software RS232 versie, er wordt geen speciale AVR hardware gebruikt zodat ze op elk I/O-bit toegepast kunnen worden. Het nadeel is dat de routines alle CPU-tijd opslokken.

```
BITRATE#                                                     RS232s
   ( -- u )
   Deze constante bepaalt de snelheid van de software RS232.
   Er zijn versies voor 1, 2, 4, 8 en 16 MHz. De basissnelheid
   is 19K2 baud voor alle kristallen.

RS232-EMIT                                                   RS232s
   ( char -- )
   Verzend het karakter 'char' via de RS232.

RS232-KEY                                                    RS232s
   ( -- char )
   Wacht tot het karakter 'char' via RS232 ontvangen is.

RTYPE              "ram-type"                                RS232s
   ( addr u -- )
   Verzend 'u' karakters vanaf RAM adres 'addr' via de RS232.

SETUP-RS232                                                  RS232s
   ( -- )
   Initialiseer de I/O-bits voor RS232.
```

TYPE RS232s
(addr u --)
Verzend ’u’ karakters vanaf ROM adres ’addr’ via de RS232.

6.4.15 GLCD.FRT & LETTERS.FRT

Deze files vormen tezamen een aansturing voor grafisch LCD van 122x32 pixels.

>DISPLAY Glcd
(x --)
Stuur de bitrij ’x’ naar actieve positie op het LC-display.

CR Glcd
(--)
Ga naar de volgende regel op het LC-display, scroll het display een regel omhoog als er geen lege regels meer beschikbaar zijn.

EMIT Glcd
(char --)
Zet ’char’ op het grafisch LC-display. Het lettertype is afhankelijk van de ingestelde letterset.

GL "g-l" Letters
(--)
Activeer grote karakters voor het LC-display.

HOME Glcd
(--)
Zet de cursor linksboven op het LC-display.

INITIALISEER Glcd
(--)
Maak het LC-display klaar voor gebruik.

INSTRUCTIE Glcd
(byte --)
Stuur de instructie ’byte’ naar het hele LC-display.

NL "n-l" Letters
(--)
Activeer karakters van normale grootte voor het LC-display.

PAGE Glcd
(--)
Wis het LC-display en voer de functie HOME uit.

RTYPE "ram-type" Glcd
(a u --)
Zet ’u’ karakters vanaf RAM-adres ’a’ op het LC-display.

SCROLL Glcd
(+n --)
Scroll regel ’+n’ naar de bovenste regel van het LC-display.

STREEP Glcd
(x u --)
Zet het bitpatroon ’x’ ’u’ maal op het LC-display.

TYPE Glcd
(a u --)
Zet ’u’ karakters vanaf ROM-adres ’a’ op het LC-display.

WIS Glcd
(--)
Wis vanaf de cursorpositie tot het eind van de regel.

XY Glcd
(x y --)
Positioneer de cursor op positie ’x’ ’y’ van het LC-display.

6.4.16 LCD.FRT

LCD-AT-XY Lcd
(x y --)
Zet de cursor op position ’x’ ’y’ van het LC-display.
Let op: Dit werkt zo alleen op 40 kar. displays.

LCD-BS Lcd
(--)
Wis het vorige karakter van het LC-display.

LCD-CHAR Lcd
(byte --)
Stuur het karakter ’char’ naar een LC-display.

LCD-CR Lcd
(--)
Doe een CR op een LC-display.
Let op: Dit werkt zo alleen op 40 kar. displays.

LCD-EMIT Lcd
(char --)
Zet het karakter ’char’ op het LC-display.

LCD-HOME Lcd
(--)
Zet de cursor in de linkerbovenhoek van het LC-display.

LCD-INIT Lcd
(--)
Initialiseer een LC-display. Zet 4-bits data bus, maak het display schoon, zet de cursor uit en zet de display richting van links naar rechts.

LCD-INSTR Lcd
(char --)
Stuur de instructie ’byte’ naar een LC-display.

LCD-PAGE Lcd
(--)
Maak het LC-display schoon en zet de cursor in linker bovenhoek.

LCD-RTYPE "lcd-ram-type" Lcd
(addr u --)
Zet ’u’ karakters vanaf het RAM adres ’addr’ naar een LC-display.

LCD-SPACE Lcd
(--)
Zet een spatie op het LC-display.

LCD-SPACES Lcd
(u --)
Zet ’u’ spaties op het LC-display.

LCD-TYPE Lcd
($addr u --)
Zet ’u’ karakters vanaf het ROM adres ’$addr’ naar het LC-display.

6.4.17 NUMBERS.FRT

"number-sign" Numbers
(d1 -- d2)
Deel ’d1’ door het getal in ’BASE’ met als resultaat ’d2’. De rest ’n’ wordt omgezet tot een karakter welke toegevoegd wordt aan het begin van de uitvoerstring. Te gebruiken tussen <# en #>.

#> "number-sign-greater" Numbers
(d -- addr u)

Gooi ’d’ weg en maak de uitvoerstring beschikbaar als ’addr’ ’u’.

#S "number-sign-s" Numbers
(d1 -- d2)

Converteer alle digits in ’d1’ als beschreven bij # todat het resulterende quotient ’d2’ nul is. Te gebruiken tussen <# en #>.

. "dot" Numbers
(n --)

Druk ’n’ af in een vrij formaat.

<# "less-number-sign" Numbers
(--)

Begin een nieuwe getal uitvoerstring.

BASE Numbers
(-- addr)

addr is het adres van de cel die het grondtal voor de getal conversie bevat, geldige waarden zijn 2 t/m 36.

D. "d-dot" Numbers
(d --)

Druk ’d’ af in een vrij formaat.

DECIMAL Numbers
(--)

Zet het grondtal op 10.

DIG-OUT "dig-out" Numbers
(du1 -- du2 char)

Deel ’du1’ door het getal in ’BASE’ tot ’du2’ en ’char’. Wordt gebruikt om een digit te reduceren uit een unsigned dubbel getal, tot zijn equivalente ASCII karakter.

HEX Numbers
(--)

Zet het grondtal op 16.

HOLD Numbers
(char --)

Voeg het karakter ’char’ toe aan het begin van de uitvoerstring in wording. Te gebruiken tussen <# en #>.

SIGN Numbers
(n --)
Als ’n’ negatief is, voeg dan een minteken toe aan de uitvoerstring in wording. Te gebruiken tussen <# en #>.

U. "u-dot" Numbers
(u --)
Druk ’u’ af in een vrij formaat.

6.4.18 ARITH.FRT

*/ "star-slash" Arith
(n1 n2 n3 -- n4)
Vermenigvuldig ’n1’ met ’n2’ met het dubbel tussenresultaat. Deel de dubbel daarna door ’n3’, ’n4’ is het quotient.

*/MOD "star-slash-mod" Arith
(n1 n2 n3 -- n4 n5)
Vermenigvuldig ’n1’ met ’n2’ met het dubbel tussenresultaat. Deel de dubbel daarna door ’n3’, ’n4’ is de rest en ’n5’ is het quotient.

/ "slash" Arith
(n1 n2 -- n3)
Deel ’n1’ door ’n2’, ’n3’ is het quotient.

/MOD "slash-mod" Arith
(n1 n2 -- n3 n4)
Deel ’n1’ door ’n2’, ’n3’ is de rest en ’n4’ het quotient.

FM/MOD "f-m-slash-mod" Arith
(d n1 -- n2 n3)
Deel ’d’ door ’n1’, ’n2’ is de rest en ’n3’ is het ’floored’ quotient.

M* "m-star" Arith
(n1 n2 -- d)
’d’ is het signed produkt van ’n1’ maal ’n2’.

MOD Arith
(n1 n2 -- n3)
Deel ’n1’ door ’n2’, ’n3’ is de rest.

SM/REM "s-m-slash-rem" Arith
(d n1 -- n2 n3)
Deel ’d’ door ’n1’, ’n2’ is de rest en ’n3’ het ’symetric’ quotient.

6.4.19 DOUBLE.FRT

CARRY Double
(-- flag)
High level toegang tot de carry vlag, de 'flag' is true als de carry gezet is.

D* "d-star" Double
(d1 d2 -- d3)
Vermenigvuldig 'd1' en 'd2' met het signed resultaat 'd3'.

D0< "d-zero-less" Double
(d -- flag)
De vlag is true als 'd' kleiner is dan nul.

D0= "d-zero-equals" Double
(dx -- flag)
De vlag is true als 'dx' gelijk is aan nul.

D0> "d-zero-greater" Double
(d -- flag)
De vlag is true als 'd' een positief getal groter dan nul is.

D< "d-less" Double
(d1 d2 -- flag)
De vlag is true als 'd1' kleiner is dan 'd2'.

D<> "d-not-equal" Double
(dx1 dx2 -- flag)
De vlag is true als 'dx1' ongelijk is aan 'dx2'.

D= "d-equals" Double
(dx1 dx2 -- flag)
De vlag is true als 'dx1' is gelijk is aan 'dx2'.

D> "d-greater" Double
(d1 d2 -- flag)
De vlag is true als 'd1' groter is dan 'd2'.

DMAX "d-max" Double
(d1 d2 -- d3)
'd3' is de grotere van 'd1' en 'd2'.

DMIN "d-min" Double
(d1 d2 -- d3)
'd3' is de kleinere van 'd1' en 'd2'.

```
DU<                "d-u-less"                                  Double
   ( du1 du2 -- flag )
   De vlag is true als 'du1' kleiner is dan 'du2'.

DU>                "d-u-greater"                               Double
   ( du1 du2 -- flag )
   De vlag is true als 'du1' groter is dan 'du2'.

DUMAX              "d-u-max"                                   Double
   ( du1 du2 -- du3 )
   'du3' is de grotere van 'du1' en 'du2'.

DUMIN              "d-u-max"                                   Double
   ( ud1 ud2 -- ud3 )
   'du3' is de kleinere van 'du1' en 'du2'.

DUSQRT             "d-u-square-root"                           Double
   ( du -- u )
   Bereken de wortel van 'du', met als resultaat 'u'.
```

6.4.20 MS.FRT

```
/MS                "slash-m-s"                                 Ms
   ( u -- )
   Wacht tenminste 'u' maal 100 microseconden. Er zijn versies
   beschikbaar voor freq. van 1, 2, 4, 8, 10, 11.059, 12 en 16 MHz.

MS                 "m-s"                                       Ms
   ( u -- )
   Wacht tenminste 'u' milliseconden. Er zijn versies beschikbaar
   voor frequenties van 1, 2, 4, 8, 10, 11.059, 12 en 16 MHz.
```

6.4.21 RANDOM.FRT

Deze file bevat twee verschillende pseudorandom generators. Standaard wordt de schuifregister versie geladen.

```
CHOOSE                                                         Rnd
   ( u1 -- u2 )
   Maak het random getal 'u2', 'u2' ligt in het bereik van 0 tot 'u1-1'.

SETUP-RANDOM                                                   Rnd
   ( -- )
   Initialiseer het random getal zaadje.
```

6.4.22 CATCH.FRT

```
ABORT                                                          Catch
   ( i*x -- || R: j*x -- )
   Maak de data-stack schoon en voer de funktie -1 THROW uit (12).

CATCH                                                          Catch
   ( i*x xt -- j*x 0 | j*x n )
   Zet een 'exception-frame' op de return-stack en voer het token
   'xt' uit (net als met EXECUTE). Als er niets fout gaat staat
   er na uitvoering een 'nul' op de stack, bovenop wat door 'xt'
   op de stack is aangebracht. Het 'exception-frame' is verwijderd.
   Zie ook: THROW voor de rest van de beschrijving (20).

ERROR-HANDLER                                                  Catch
   ( -- addr )
   Wijs naar de in gebruik zijnde 'error-handler'.

THROW                                                          Catch
   ( k*x n -- k*x || i*x n )
   Haal het 'exception-frame' van de return-stack en ga door met
   uitvoeren na CATCH, De data- en return-stack zijn teruggebracht in
   de toestand voor de CATCH met daarbovenop het foutnummer (14).
```

6.4.23 KEYB1.FRT

```
AKEY              "a-key"                                      Keyb1
   ( -- char )
   Wacht tot er een toets ingedrukt is, char is de ASCII waarde
   van die toets.

AKEY?             "a-key-question"                             Keyb1
   ( -- Flag )
   Geef een true vlag als een toets ingedrukt is, anders false.
```

6.4.24 KEYB2.FRT

```
KEY                                                            Keyb2
   ( -- char )
   Wacht tot er een toets ingedrukt is, char is de ASCII waarde
   van die toets.

KEY?              "key-question"                               Keyb2
   ( -- flag )
   Geef een true vlag als een toets ingedrukt is, anders false.
```

6.4.25 RC5.FRT

RCKEY "r-c-key" Rc5
(-- x)
Ontvang een RC5 databyte, ’x’ is een code uit de gedefinieerde RC5-code commando set. Bekijk hiervoor de RC5 device documentatie.

RCKEY? "r-c-key-question" Rc5
(-- flag)
Geef true als er een voor dit systeem geldig RC5 commando is.

SETUP-RC Rc5
(--)
Initialiseer de RC5-decoder.

6.4.26 MUSIC.FRT

1/1 Music
(--)
Zet TEMPO zo dat hele noten gespeeld worden.
Er zijn ook: 1/2, 1/4 en 1/8 tempo’s.

A1 Music
(--)
Speel de noot A1 met de lengte van TEMPO millisec. Er zijn nog 22 andere noten, die samen ongeveer 2.3 octaaf vormen van A1 tot C3.

REST Music
(--)
Wacht TEMPO milliseconden.

SETUP-MUSIC Music
(--)
Initialiseer de muziek hardware.

6.4.27 BAMBOE.FRT

BAMBOE! "bamboe-store" Bamboe
(x --)
Stuur de data x naar de bamboe uitgangen.

BAMBOE@ "bamboe-fetch" Bamboe
(-- x)
Lees de data x van de bamboe ingangen.

```
SETUP-BAMBOE                                                    Bamboe
   ( -- )
   Setup portbits for use with bamboe.
```

6.4.28 7SEGM.FRT

```
#DIGITS                                                         7segm
   ( -- u )
   Bevat het aantal aangesloten digits.

PUNT                                                            7segm
   ( -- addr )
   Bevat de positie van de decimale punt op het display gerekend
   vanaf het rechter display. Nul betekent geen decimale punt.

RTYPE              "ram-type"                                   7segm
   ( addr u -- )
   Stuur het resultaat van een getal conversie 'addr' 'u' met <# # etc. #>
   naar het zeven segmentdisplay. Als er minder dan #DIGITS cijfers
   zijn dan wordt het getal links aangevuld met lege digits.
```

6.4.29 PIR.FRT

```
MENS-GEZIEN?       "mens-gezien-query"                          Pir
   ( -- flag )
   Geef true als de PIR-sensor een warmbloedig wezen heeft gezien,
   anders false. Een meting duurt 80 milliseconden.
```

6.4.30 BCD.FRT

```
>BCD               "to-bcd"                                     Bcd
   ( +n1 -- +n2 )
   Converteer binair getal +n1 naar BCD getal +n2.

BCD>               "bcd-from"                                   Bcd
   ( +n1 -- +n2 )
   Converteer BCD getal +n1 naar binair getal +n2.
```

6.4.31 CP-ADC.FRT

```
ADC                "a-d-c"                                      Cp-adc
   ( -- u )
   Bepaal de analoge spanning op PB3, 'u' is het resultaat.
```

SETUP-ADC Cp-adc
(--)
PB2 en 3 zijn hoogohmige ingangen.

6.4.32 MIDI.FRT

MIDI-EMIT MIDI
(char --)
Verzend het karakter ’char’ via MIDI.

MIDI-KEY MIDI
(-- char)
Wacht tot een karakter ’char’ via MIDI ontvangen is.

MIDI-KEY? "midi-key-question" MIDI
(-- flag)
Geef true als een karakter via MIDI ontvangen is, anders false.

MIDI-RTYPE "midi-ram-type" MIDI
($addr u --)
Verstuur ’u’ karakters vanaf RAM adres ’$addr’ via MIDI.

MIDI-TYPE MIDI
($addr u --)
Verzend ’u’ karakters vanaf ROM adres ’$addr’ via MIDI.

SETUP-MIDI MIDI
Initialiseer de UART op een MIDI baudrate van 31K25 baud.

6.4.33 GP2D02.FRT

AFSTAND GP2D02
(-- afstand)
Geef ’afstand’ tot een object aan (lager is dichterbij).
Een meting duurt ongeveer 75 milliseconden.

SETUP-GP2D02 GP2D02
(--)
Initialiseer I/O-pennen voor de GP2D02.

7 Voorbeeld code

Er zijn verschillende geteste voorbeeld files bij AVR ByteForth gevoegd. Het doel van deze files en de bijzonderheden worden hieronder kort toegelicht. Zowel voor de beginnende als gevorderde gebruiker zal er zeker iets van nut tussen te vinden zijn.

7.1 De voorbeelden op een rij

PINCODE.FRT	Pincode automaat, matrix toetsenbord uitlezing, bit i/o en een voorbeeld van het gebruik van `CATCH` en `THROW`.
AVR-PBM.FRT	8 KHz pulsbreedtemodulatie via timer-1.
AVR-ADC.FRT	Gebruik van de interne ADC op sommige AVR's.
AVR-EEP.FRT	Het opslaan en uitlezen van data via de in alle AVR's ingebouwde EEPROM voor dataopslag.
AVR-WDT.FRT	Demonstratie met behulp van de ingebouwde watchdogtimer.
AVRSLEEP.FRT	Voorbeeld van het gebruik van de sleepmode op avr's.
RS232DEM.FRT	RS232 voorbeeld van eenvoudige uitvoer.
RC5TEST.FRT	Toon RC5-codes op 8 leds.
CREATE1.FRT	Samen met CREATE2.FRT en CREATE3.FRT, voorbeelden van het gebruik van `CREATE` en `DOES>` in AVR ByteForth.
EXEC1.FRT	Met EXEC2.FRT en EXEC3.FRT voorbeelden van executietabellen gemaakt in AVR ByteForth.
EEVAR.FRT	Definiërend woord om tabellen in EEPROM te maken.
HILEVEL1.FRT	Samen met HILEVEL2.FRT tonen ze hoe high-level interrupts in AVR ByteForth gebruikt kunnen worden.
SERVO0.FRT	High-level besturing van twee modelbouwservo's.
BITIO.FRT	High-level voorbeelden van het gebruik van I/O-bitbesturing en het toepassen van bitvlaggen in AVR ByteForth.
SMAGIC.ZIP	Elektuur applicatie, de synchrone besturing van max. tien modelbouwservo's via RS232. Van elke servo kan apart de begin- en eindstand opgegeven worden.
MINI-USG.FRT	Ultrasoon afstand meter, bereik 0 cm tot 200 cm.
LM75MINI.FRT	En LM75LEDS.FRT een I2C-temperatuurmeter op 8 leds.
KWIS2.FRT	Kwisschakeling voor 2 tot 6 deelnemers.
PBM-INT.FRT	Samen met PBM-INT1.FRT, PBM-INT2.FRT en PBM-INT3.FRT vier voorbeelden van het gebruik van PBM via interrupts in AVR ByteForth.
CP-ADC.FRT	ADC via ingebouwde comparator.
HCCDEMO.FRT	Toon looplicht of binaire teller op de leds van een STK200 starterkit en toon de karakterset van een LCD op zijn display.
2313TASK.FRT	Multitasking voorbeeld op een AT90S2313, drie samenwerkende taken die samen een looplicht vormen op een STK200-kit.
MEGADEMO.FRT	Samen met andere files een set voorbeelden speciaal voor megaAVR's waaronder multitasking, realtimeklok, RS232 en PBM.
TINYSLP.FRT	Voorbeeld van de sleepfunctie op een ATtiny22 en AT90S2343.

7.2 'Egelwerkboek

Alle voorbeelden uit het 'Egelwerkboek zijn aangepast zodat ze nu ook werken met de AT90S2313 AVR-processor. Het AT51-printje aangepast om naast de AT89Cx051 nu ook de AT90S2313 te kunnen gebruiken met de HCC Forth gg standaard programmeerinterface. In het directory 'Egel vindt je al deze voorbeeld files uit het 'Egel werkboek.
Elk hoofdstuk in het werkboek is als volgt opgebouwd. Een inleiding, schema, PCB-layout, onderdelenlijst, bouwbeschrijving, software beschrijving en een software listing. Achter in het 'Egelwerkboek vindt je datasheets van alle gebruikte onderdelen en andere aanvullende informatie. Meer over het 'Egel werkboek kun je vinden op de homepage van de HCC Forth gg:

http://www.forth.hccnet.nl/pr-egel.htm.

7.3 'Egel files op een rij

ESW-01.FRT	Binaire teller op 8 leds.
ESW-02.FRT	Looplicht op 8 leds.
ESW-03.FRT	Looplicht op 8 leds met snelheidsregeling.
ESW-04.FRT	Analoog naar digitaal omzetting via TLC549IP.
ESW-05.FRT	Lichtmeter via A/D omzetter.
ESW-06.FRT	RS232 seriële I/O.
ESW-07.FRT	Analoge datalogger (A/D en RS232 gecombineerd.
ESW-08a.FRT	Vermogensregeling via pulsbreedtemodulatie (PBM) met behulp van een timerinterrupt routine.
ESW-08b.FRT	Vermogensregeling via pulsbreedtemodulatie (PBM) met gebruik van de ingebouwde PBM-hardware van de AVR.
ESW-09.FRT	Relais-aansturing.
ESW-10.FRT	Besturing van twee modelbouwservo's via interrupt.
ESW-11.FRT	Unipolaire stappenmotorsturing, via eenfase, tweefase of halvestap aansturing en snelheidsregeling.
ESW-12.FRT	Bipolaire stappenmotorsturing, via eenfase, tweefase of halvestap aansturing en snelheidsregeling.
ESW-13.FRT	Periodetijd- en toerentalmeting van een blokvormig invoersignaal, uitvoer via RS232.
ESW-14.FRT	RC5-decoder via interrupt met 220 Volt solidstate relais uitgang.
ESW-15.FRT	RC5-zender voor besturing van ESW-14.
ESW-16.FRT	Zevensegment displaybesturing zonder speciale decoder.
ESW-17.FRT	LCD-aansturing d.m.v. een 4-bits databus.
ESW-18.FRT	I2C-uitvoer via PCF8574.
ESW-19.FRT	I2C-invoer en uitvoer via PCF8574.
ESW-20.FRT	Dataopslag naar I2C-EEPROM met uitlezing via PCF8574.
ESW-21.FRT	I2C-klok met wekker via PCF8583.

8 Flash programmer

Om de AVR ByteForth omgeving compleet te maken is er een In-System-Programmer toegevoegd die de SPI-interface van de AVR gebruikt. De interface is opgebouwd met een minimum aan hardware. Deze ISP Flash-programmer werkt via de printerpoort. Voor de programmer moet een driver met aangepaste kabel worden gemaakt. De programmer werkt met de alle bekende 'In System' programmeerbare AVR-typen, zoals de AT90S2313, ATtiny26 en ATmega32.

8.1 Programmer commando's

Bij normaal gebruik van de programmer zijn er negen belangrijke commando's. De eerste drie zijn de meest gebruikte:

E	Wis (Erase) de Flash EPROM.
P	Programmeer (Program) de Flash EPROM.
V	Vergelijk (Verify) de Flash EPROM met buffer.
Lock1	Programmeer Lock-bit-1 (programmeren uitgeschakeld).
Lock2	Programmeer Lock-bit-2 (vergelijken uitgeschakeld).
R	Lees (Read) de Flash EPROM naar buffer.
T	Toon cpu type aan de hand van de 'signature bytes'.
EB	Wis (Erase) buffer met $FF.
DB	Druk (Dump) buffer inhoud af.

8.2 Problemen bij de ISP Flash-programmer

De AT90, ATtiny en ATmega Flash EPROM's zijn gegarandeerd duizend maal te programmeren, maar ze kunnen door statische elektriciteit kapot gaan (pas op). Het is echter veel gevaarlijker de chips verkeerd om op het experimenteerbord te steken. Pas ook op met de voedingsspanning, de chips mogen niet meer dan 6 Volt hebben. Als de programmer een chip niet herkent, dan kunnen er vier dingen gebeurd zijn.

[1] De chip is door een van bovenstaande redenen kapot gegaan.
[2] De voedingsspanning is niet (goed) aangesloten.
[3] De AVR ByteForth compiler is beschadigd of de ISP-programmer (de dongle) is stuk.
[4] Een of meer signaturebytes waarmee een AVR geidentificeerd kan worden is beschadigd of gewist.

De AT90S1200, de oudste AVR heeft slechte SPI-ingangen, om hem te kunnen programmeren moet je de snelheid van de ISP-interface drastisch verlagen. Een factor 40 heb ik herhaaldelijk moeten gebruiken, SET-PAUSE staat bij mij normaal op 50 voor de AT90S1200 wordt dat `2000 SET-PAUSE`.

8.3 Werken met EEPROM

Om data uit Eprom of Flash terug lezen, moet de juiste 'switch' in de compiler omgezet zijn. De default stand is FLASH, alle programmer commando's zoals P, V en R werken op het Flash geheugen. Door nu de 'switch' EEPROM te activeren werken alle hiervoor genoemde commando's op het EEPROM geheugen. Hierna een voorbeeld waarin het EEPROM met data gevuld wordt. Ben je klaar met het EEPROM vergeet dan niet de 'switch' FLASH te activeren, om weer Flash geheugen te kunnen bewerken, that's all.

8.4 Hoe vul je EEPROM

Elke AVR heeft intern EEPROM, hierin kan ook vooraf belangrijke data meegegeven worden. We kunnen ByteForth gebruiken om deze data te maken en in EEPROM te zetten. De eerste byte in EEPROM kan instabiel zijn, om problemen te voorkomen adviseerd Atmel om EEPROM byte-0 niet te gebruiken. Een klein voorbeeld:

```
90S2313                                 \ Geef AVR type op
0 0 0 MEMORY                            \ Begin op adres nul
EEPROM                                  \ De data is voor EEPROM
00 C,                                   \ Zet nul in de eerste byte
&W C, &I C, &L C, &L C, &E C, &M C,     \ met daar achter mijn naam
SAVE data.bin                           \ Bewaar de EEPROM data binair
P V                                     \ Vul EEPROM en verifieer data
FLASH  EMPTY                            \ Klaar en maak boel schoon
```

8.5 Lezen en schrijven van binary's

Er zijn in AVR ByteForth twee methoden om binary's op te slaan en/of in te lezen. De hierboven beschreven methode bewaart de gegenereerde EEPROM-data binair. Natuurlijk kun je een executable op dezelfde manier bewaren SAVE `voorbeeld.bin`. Vergeet niet dat de file-extensie zelf moet worden opgegeven.
Inlezen van binaire data doe je met READ `voorbeeld.bin`. Door deze commando's kun je gegevens transporteren naar een andere programmer of debugger. Je kunt natuurlijk ook, de met een andere compiler gegenereerde code in ByteForth inlezen en gebruiken.

8.6 Lezen en schrijven van Intel-Hex files

Veel programmers kunnen niet met binary's werken, maar wel met Intel-Hex files. Een Intel-Hex bevat een ASCII representatie van de gegenereerde binary inclusief checksum. Voor het schrijven is er `WRITE-HEX voorbeeld.hex`. Inlezen doe je met `READ-HEX voorbeeld.hex`. Beide commando's gebruik je op dezelfde manier als de binaire versie.

8.7 Extra programmer-instructies toevoegen

Moderne AVR-chips als de ATmega8 hebben veel meer mogelijkheden. Het gaat daarbij vooral om de 'fusebits'. Ingebouwde programmeerinstructies daarvoor ontbreken in Byteforth. Met de commando's >AVR en AVR> kunnen deze toch gebruikt worden.
Een ATmega8 voorbeeld: Volgens het datasheet zet de volgende string fusebit-3 van de 'Fuse low byte' `$AC $A8 $FF $D1`, het EEPROM wordt bij een wis (E-commando) niet meer gewist. Een fusebit moet op nul gezet worden om een functie te activeren!
In ByteForth is dat `$D1 $FF $A8 $AC >AVR`. Let op, lees het datasheet zeer zorgvuldig, voor je hier zelf mee aan de slag gaat! Als je per abuis de ISP-interface uitschakeld of van de reset-pen een I/O-pen maakt dan ben je de ISP-interface geheel kwijt. Alleen parallel programmeren krijgt de chip weer in het gareel. Meer voorbeelden:

ATmega8535 lees fusebits laag: `$FF $00 $50 AVR> .HEX ( geeft ) $E1`
ATmega8535 schrijf fusebits laag, brownout aan: `$A1 $FF $A0 $AC >AVR`
ATmega8535 schrijf fusebits laag, brownout en osc.: `$AF $FF $A0 $AC >AVR`

8.8 Flash programmer woordenlijst

```
.HELP              "dot-help"                                  ISP
   ( -- )
   Toon de commando's van de programmeer software.

.PAUSE             "dot-pause"                                 ISP
   ( -- )
   Toon klokpuls verlengtijd, default is 100.

>AVR               "to-a-v-r"                                  ISP
   ( b0 b1 b2 b3 -- )
   Stuur de vier bytes 'b0', 'b1', 'b2' en 'b3' naar een AVR chip
   waarmee handmatig een schrijfcommando toegevoegd wordt.

AVR>               "a-v-r-from"                                ISP
   ( b0 b1 b2 -- b3 )
   Stuur de drie bytes 'b0', 'b1' en 'b3' naar een AVR chip om waarmee
   handmatig een leescommando toegevoegd wordt, 'b3' is het resultaat.

DB                 "d-b"                                       ISP
   ( -- )
   Toon hexdump van het gebruikte deel van de ByteForth buffer.

E                  "erase"                                     ISP
   ( -- )
   Wis het gehele FLASH en EEPROM van de microcontroller.

EB                 "e-b"                                       ISP
   ( -- )
   Wis de code buffer door hem met $FF te vullen.

EEPROM             "e-e-prom"                                  ISP
   ( -- )
   De programmer werkt op het data (EEPROM) geheugen van de cpu.

FLASH                                                          ISP
   ( -- )
   De programmer werkt op het code (FLASH) geheugen van de cpu.

INFO                                                           ISP
   ( -- )
   Toon de versie info van de programmeer software.

LOCK1              "lock-one"                                  ISP
   ( -- )
   Zet lock-bit 1, die het verder programmeren van de
   microcontroller verbiedt.
```

LOCK2 "lock-two" ISP
(--)
Zet lock-bit 2, die het uitlezen van de microcontroller verbiedt.

P "program" ISP
(--)
Schrijf het gebruikte deel van de ByteForth buffer
naar de microcontroller. Zie ook: FLASH en EEPROM.

PR "p-r" ISP
(--)
(Wis), programmeer en verifieer een microcontroller.

PRN1 "p-r-n-one" ISP
(--)
De ISP adapter is verbonden met printerpoort 1.
Er zijn ook: PRN2, PRN3 en PRN4

R "read" ISP
(--)
Lees het hele FLASH of EEPROM van de microcontroller naar de
ByteForth buffer. Zie ook: FLASH en EEPROM.

RESTART ISP
(--)
Herstart de via ISP aangesloten cpu door een reset puls.

RUN ISP
(--)
Laat de via ISP aangesloten chip vrij lopen.

SET-PAUSE ISP
(u --)
Zet de klokpuls verlengtijd, default is 100.

STOP ISP
(--)
Stop de via ISP aangesloten chip.

T "cpu-type" ISP
(--)
Toon het typenummer van de aangesloten microcontroller.

V "verify" ISP
(--)
Verifeer het gebruikte deel van de ByteForth buffer met
dat van de microcontroller. Zie ook: FLASH en EEPROM.

9 Geheugenindeling

AVR-microcontrollers hebben drie geheugen gebieden, onderverdeeld in FLASH, RAM en EEPROM. Het RAM geheugen heeft echter ook een onderverdeling op instructieniveau: 32 registers, SFR-registers (ook I/O-registers genoemd) met een maximum van 64 of 224 stuks, 0 tot 4Kbyte RAM. De hardwareregisters varieren sterk in aantal op verschillende AVR-chips. De meeste AVR's komen met het blok van 64 SFR-registers uit. Sommige mega-AVR's hebben extra SFR-registers nodig, maximaal 160 extra zogenaamde extended SFR-registers.

9.1 Special Function Registers tabel

Register Summary

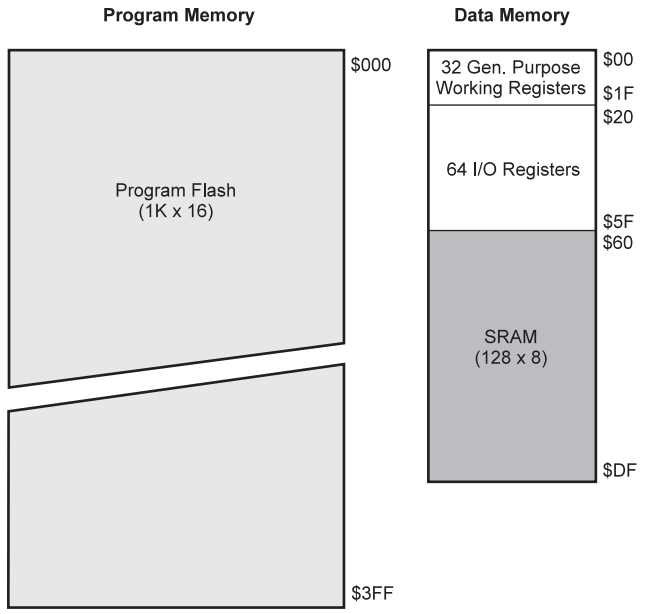
Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	19
$3E ($5E)	Reserved									
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	20
$3C ($5C)	Reserved									
$3B ($5B)	GIMSK	INT1	INT0	-	-	-	-	-	-	25
$3A ($5A)	GIFR	INTF1	INTF0							26
$39 ($59)	TIMSK	TOIE1	OCIE1A	-	-	TICIE1	-	TOIE0	-	26
$38 ($58)	TIFR	TOV1	OCF1A	-	-	ICF1	-	TOV0	-	27
$37 ($57)	Reserved									
$36 ($56)	Reserved									
$35 ($55)	MCUCR	-	-	SE	SM	ISC11	ISC10	ISC01	ISC00	28
$34 ($54)	Reserved									
$33 ($53)	TCCR0	-	-	-	-	-	CS02	CS01	CS00	31
$32 ($52)	TCNT0	Timer/Counter0 (8 Bit)								31
$31 ($51)	Reserved									
$30 ($50)	Reserved									
$2F ($4F)	TCCR1A	COM1A1	COM1A0	-	-	-	-	PWM11	PWM10	33
$2E ($4E)	TCCR1B	ICNC1	ICES1	.	-	CTC1	CS12	CS11	CS10	34
$2D ($4D)	TCNT1H	Timer/Counter1 - Counter Register High Byte								35
$2C ($4C)	TCNT1L	Timer/Counter1 - Counter Register Low Byte								35
$2B ($4B)	OCR1AH	Timer/Counter1 - Compare Register High Byte								36
$2A ($4A)	OCR1AL	Timer/Counter1 - Compare Register Low Byte								36
$29 ($49)	Reserved									
$28 ($48)	Reserved									
$27 ($47)	Reserved									
$26 ($46)	Reserved									
$25 ($45)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								36
$24 ($44)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								36
$23 ($43)	Reserved									
$22 ($42)	Reserved									
$21 ($41)	WDTCR	-	-	-	WDTOE	WDE	WDP2	WDP1	WDP0	38
$20 ($40)	Reserved									
$1F ($3F)	Reserved									
$1E ($3E)	EEAR	-	EEPROM Address Register							40
$1D ($3D)	EEDR	EEPROM Data register								40
$1C ($3C)	EECR	-	-	-	-	-	EEMWE	EEWE	EERE	40
$1B ($3B)	Reserved									
$1A ($3A)	Reserved									
$19 ($39)	Reserved									
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	50
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	50
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	50
$15 ($35)	Reserved									
$14 ($34)	Reserved									
$13 ($33)	Reserved									
$12 ($32)	PORTD	-	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	55
$11 ($31)	DDRD	-	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	55
$10 ($30)	PIND	-	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	55
...	Reserved									
$0C ($2C)	UDR	UART I/O Data Register								44
$0B ($2B)	USR	RXC	TXC	UDRE	FE	OR	-	-	-	45
$0A ($2A)	UCR	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	45
$09 ($29)	UBRR	UART Baud Rate Register								47
$08 ($28)	ACSR	ACD	-	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	48
...	Reserved									
$00 ($20)	Reserved									

Notes:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers $00 to $1F only.

Figuur 9.1: Hardwareregisters van de AT90S2313

9.2 Hoeveel geheugen ?

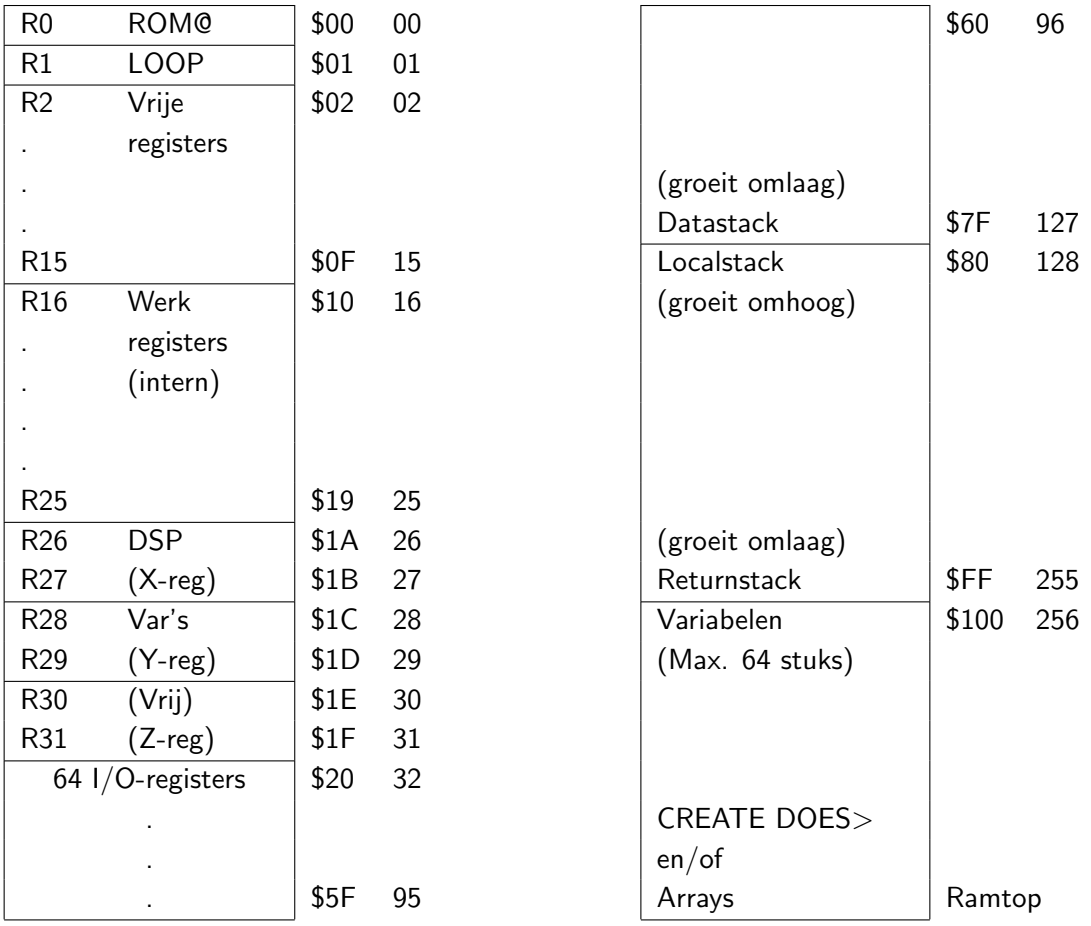
De AT90S2313 heeft 128 bytes aan RAM werkgeheugen en 128 bytes EEPROM datageheugen. Er zijn kleinere broertjes als de AT90S1200 zonder RAM maar EEPROM hebben ze altijd. Er komen steeds meer verschillende AVR-types beschikbaar. Bijvoorbeeld de AT90S8535: deze heeft ook 256 bytes RAM en EEPROM, maar als extra ook een 10-bits ADC met 8 ingangen. Een andere opvolger heet de ATmega8515, deze is gelijk aan de AT90S8515 maar alle bugs zijn er uit gehaald en de klok is opgevoerd van 8 MHz naar 16 MHz. Sommige AVR's hebben ook een brownout-detector ingebouwd, deze zet de controller tijdens een spanningsdip in de reset toestand. De ATmega8 heeft 1 kByte RAM, 8 kByte Flash, 512 Bytes EEPROM, een watchdog timer, ADC, brownout, etc. Voor laatste gegevens, zie de Atmel homepage.



Figuur 9.2: Geheugenmap's van de AT90S2313

9.3 ByteForth geheugengebruik

Onderstaande plaatje toont het data-geheugen van een AVR microcontroller. Het 16-bits X-register is de datastackpointer (DSP), het Y-register is de variabelenpointer (verkorte adresseerwijze) en het Z-register is vrij, maar wordt o.a. gebruikt voor locale variabelen, inline strings en DOES>. De registers R16 t/m R25 zijn als werkruimte gereserveerd voor codedefinities. Ze worden intensief in de macro's gebruikt, ook in uw eigen code definities mogen ze worden toegepast. Register R0 (en R1 bij ATmega's) is gereserveerd voor lezen uit ROM, R1 of R2 is gereserveerd voor de high-level FOR NEXT en DO LOOP lussen. De resterende registers t/m R15 zijn beschikbaar. De fout opvang routines vragen een vrij register aan als pointer. De datastack gebruikt het geheugen van adres 96 t/m 127 en groeit omlaag. Tenslotte komen we bij het geheugen dat met MEMORY gemodificeerd kan worden. Bij gebruik van bit-vlaggen worden één of meer registers aangevraagd net als met REGISTER. Het aantal vrije registers daalt daarom als bit-vlaggen gebruikt worden, beginnend bij R2 of R3, afhankelijk van het AVR-type.
Maximaal 64 variabelen kunnen aangevraagd worden. Het stuk dat overblijft tussen de datastack en de variabelen t/m adres 255 of lager, wordt gebruikt voor de returnstack.
Als de returnstack minder dan 8 nestingen voor subroutines en interrupts kan bevatten, genereert ByteForth een foutmelding. Resterend RAM geheugen vanaf adres 256 kan o.a. gebruikt worden voor array's.



Figuur 9.3: Basis ByteForth geheugen indeling

Voor de AT90S2313 is het configureerbaar geheugen standaard als volgt ingedeeld: 16 bitvlaggen, 32 variabelen en een returnstack van 32 adrescellen. Voor de 90S8515 zijn dit: 16 bitvlaggen, 48 variabelen en een returnstack van 64 adrescellen. Er rest dan nog 304 bytes RAM. De ATmega8 is standaard ingedeeld met: 16 bitvlaggen, 48 variabelen en een returnstack van 64 cellen er rest daar 816 bytes RAM.
Voor alle processoren geldt dat het codegeheugen voor ByteForth begint na de opstartvector en alle interruptvectoren; cel-adres 11 voor de AT90S2313, ermee rekening houdend dat een sprong een lengte heeft van 2 bytes (de RJMP en RCALL). De ATmega8 gebruikt vier bytes voor elke sprong, de code begint daarom op cel-adres 19.
Er is ook nog het woord `MAP`, hiermee is de geheugenindeling van AVR ByteForth compleet overhoop te halen. Zowel de datastack en returnstack kunnen nu ook van grootte veranderd worden, de geheugenvolgorde blijft wel hetzelfde.
De AVR heeft speciale opcode's beschikbaar om zowel de registers R0 t/m R31 als de I/O-registers (SFR's) snel te behandelen.

10 AVR assembler

De AVR-processoren ondersteunen niet altijd alle opcode's uit de AVR-instructieset (zie bladzijde 130). Je kunt er echter van uitgaan dat lezen of schrijven naar extern RAM meestal onzin oplevert. Er kan maar op enkele AVR's externe RAM aangesloten worden. Er is bijna altijd interne-RAM aanwezig, een grote registerset, EEPROM en een blok hardware-registers. Deze hardware-registers worden in ByteForth SFR's genoemd, het zijn registers om de ingebouwde hardware te besturen.
De assembler waarschuwt de gebruiker als hij een opcode en/of adresseermode gebruikt die niet geldig is voor de geselecteerde AVR.

10.1 Adresseermodes en argumenten

Register adressering	`R1 R16 MOV,`	`R3 INC,`
Directe adressering (RAM)	`$44 R16 STS,`	`$0A 1 SBI,`
Register indirecte adressering	`X R16 ST,`	`R16 X LD,`
Immediate adressering (konstanten)	`R16 3 LDI,`	`R16 17 ADDI,`
Indirecte sprong of call	`IJMP,`	`ICALL,`

10.1.1 Argumenten voor de AVR opcodes:

- 0 t/m $DF — Direct register-, SFR- of RAM-adres voor de AT90S2313.
- 0 t/m $3FF — Code adresbereik, voor de AT90S2313 in 16-bits cellen.
- 0 t/m $3F — I/O-register adresbereik, voor de AT90S2313.
- 0 t/m 7 — Impliciete adressering van een vlag in het statusregister.
- `Rn` — Register adressering, n is 0 t/m 31.
- `X+` — Indirecte register adressering autoincrement, X, Y en Z-reg.
- `-X` — Indirecte register adressering autodecrement, X, Y en Z-reg.
- `X` — Indirecte register adressering, via X, Y en Z-reg.
- `n [X]` — Indirecte register adressering, via X, Y en Z-reg. met offset n (0 tot 63).
- `LPM,` — Impliciete register adressering, via het Z-reg.
- `n LDI,` — Immediate adressering, n is 0 t/m 255.

10.2 Instructies zonder argument

```
RET,    RETI,   NOP,    LPM,    WDR,    SLEEP,  CLC,    CLI,
CLN,    CLH,    CLS,    CLT,    CLV,    CLZ,    SEC,    SEI,
SEH,    SES,    SET,    SEV,    SEZ,
```

10.3 Instructies met een argument

Al deze instructies gebruiken als argument één register <Rn>. Alle 32 registers zijn bruikbaar. Bij `BSET,` en `BCLR,` staat `Bit` (in statusreg.) voor een getal van 0 t/m 7.

```
<Rn> ASR,    <Rn> LSL,    <Rn> LSR,    <Rn> ROL,    <Rn> ROR,
<Rn> CLR,    <Rn> SER,    <Rn> COM,    <Rn> NEG,    <Rn> DEC,
<Rn> INC,    <Rn> SWAP,   <Rn> TST,    <Rn> POP,    <Rn> PUSH,

Bit BCLR,    Bit BSET,
```

Enkele voorbeelden van het gebruik:

`R16 INC,`	Verhoog de inhoud van register R16 met een.
`R17 LSL,`	Schuif register R17 een positie naar links.
`R26 PUSH,`	Zet de inhoud van R26 op de returnstack.
`R19 DEC,`	Verlaag de inhoud van R19 met een.

10.4 Instructies met twee argumenten

Als argument <Rn> zijn alle 32 registers toegestaan. `Bit` staat voor een getal van 0 t/m 7 (bitnummer in register). Het I/O-poort adresbereik `Prt` loopt van 0 t/m 31. Let op: De instructies die alleen voor de ATmega AVR's gelden staan in een aparte paragraaf vermeld. Het linker register is de destination (bestemming) en het rechter de source (bron).

```
<Rn> <Rn> ADC,          <Rn> <Rn> ADD,          <Rn> <Rn> AND,
<Rn> <Rn> CP,           <Rn> <Rn> CPC,          <Rn> <Rn> ADC,
<Rn> <Rn> CPSE,         <Rn> <Rn> MOV,          <Rn> <Rn> OR,
<Rn> <Rn> EOR,          <Rn> <Rn> SBC,          <Rn> <Rn> SUB,

<Rn> Bit BST,           <Rn> Bit SBRC,          <Rn> Bit SBRC,
<Rn> Bit BLD,

Prt Bit SBI,            Prt Bit CBI,            Prt Bit SBIC,
Prt Bit SBIS,           <Rn> Prt IN,            Prt <Rn> OUT,
```

Enkele voorbeelden van het gebruik:

`R16 R2 ADD,`	Tel R2 bij de inhoud van R16 op.
`X R16 ST,`	Schrijf R16 naar het adres in X.

10.4.1 Immediate adressering

Bij de immediate adresseermodes is het argument <Rn> een van de hoge 16 registers. Het getalbereik van `n` loopt van 0 tot 255. Alleen bij de `SBIW,` en `ADIW,` opcode's is `n` een getal van 0 tot 63. Register <Rn> is een van vier 16-bits registers-paren, dit zijn R24, R26, R28 en R30. Letop, ook `R24-R25` is als 16-bits register te gebruiken!

```
<Rn> n  ANDI,           <Rn> n  CBR,            <Rn> n  CPI,
<Rn> n  LDI,            <Rn> n  ORI,            <Rn> n  SBCI,
<Rn> n  SBR,            <Rn> n  SUBI,
<Rn> n  ADIW,           <Rn> n  SBIW,
```

10.4.2 Indirecte adressering

Deze opcodes werken indirect via een van de drie hoogste dubbele (16-bits) registers, deze zijn X, Y en Z genaamd. Het register <Rn> is een van de 32 registers in de AVR. De `LDS,` en `STS,` opcodes gebruiken een 16-bits RAM adres als argument en een van de 32 registers genaamd <Rn>.

`X <Rn> ST,`	Schrijf <Rn> naar RAM adres in X reg.
`X+ <Rn> ST,`	Schrijf <Rn> naar RAM adres in X reg. en verhoog X
`-X <Rn> ST,`	Verlaag X en schrijf <Rn> naar RAM adres in X reg.
`Y <Rn> ST,`	Schrijf <Rn> naar RAM adres in Y reg.
`Y+ <Rn> ST,`	Schrijf <Rn> naar RAM adres in Y reg. en verhoog Y

-Y <Rn> ST,	Verlaag Y en schrijf <Rn> naar RAM adres in Y reg.
n [Y] <Rn> ST,	Schrijf <Rn> naar RAM adres in Y reg. verhoogd met n (0-63)
Z <Rn> ST,	Schrijf <Rn> naar RAM adres in Z reg.
Z+ <Rn> ST,	Schrijf <Rn> naar RAM adres in Z reg. en verhoog Z
-Z <Rn> ST,	Verlaag Z en schrijf <Rn> naar RAM adres in Z reg.
n [Z] <Rn> ST,	Schrijf <Rn> naar RAM adres in Z reg. verhoogd met n (0-63)
<adr> <Rn> STS,	Schrijf inhoud <Rn> naar RAM adres <adr>

10.4.3 ATmega instructies

Verschillende ATmega's hebben extra opcode's. Wil je precies weten welke ze extra hebben, raadpleeg dan het datasheet van de betreffende AVR. De opcodes zijn te verdelen in vier groepen, vermenigvuldiginstructies, extra FLASH leesinstructies, FLASH schrijfinstructies en extra spronginstructies. Er zijn nog meer uitzonderingen, dat zijn de MOVW, een instructie die twee registers tegelijkertijd verplaatst, handig voor 16-bits pointers. Enkele nieuwe AVR's (niet ATmega) bezitten ook de SPM, instructie, sla R0 op in het Flash geheugen!

```
<Rn> <Rn> MUL,      <Rn> <Rn> MULS,     <Ri> <Ri> MULSU,
<Ri> <Ri> FMUL,     <Ri> <Ri> FMULS,    <Ri> <Ri> FMULSU,

<Rn> <Rn> MOVW,     <Rd> Z LPM,         <Rd> Z+ LPM,
<Rd> Z ELPM,        Z+ <Rd> ELPM,       ELPM,

SPM,                ESPM,

EIJMP,              EICALL,
Adres JMP,          Adres CALL,
```

10.4.4 Extra instructies

Het AVR databoek vermeld een aantal nagebootste opcode's. Ze staan hieronder op een rijtje. Bijna allen bestaan ze intern uit opcode's met twee argumenten. Er is een extra immediate optelling toegevoegd; dit is een SUBBI, die intern een NEGATE doet op n.

<Rn> LSL,	Is eigenlijk: <Rn> <Rn> ADD,
<Rn> ROL,	Is eigenlijk: <Rn> <Rn> ADC,
<Rn> TST,	Is eigenlijk: <Rn> <Rn> AND,
<Rn> CLR,	Is eigenlijk: <Rn> <Rn> EOR,
<Rn> bitmask SBR,	Is eigenlijk: <Rn> bitmask ORI,
<Rn> bitmask CBR,	Is eigenlijk: <Rn> bitmask INVERT ANDI,
<Rn> SER,	Is eigenlijk: <Rn> $FF LDI,
<Rn> n ADDI,	Is eigenlijk: <Rn> +n $FF XOR 1+ SUBI,
<Rn> n ADCI,	Is eigenlijk: <Rn> +n $FF XOR 1+ SBCI,
Adres GJMP,	Kiest zelf de kortst mogelijke sprong opcode uit.
Adres GCALL,	Kiest zelf de kortst mogelijke call opcode uit.

10.5 Jump en call instructies

`adres RCALL,`	Relatieve call van + of - 2 k-words
`ICALL,`	Indirecte call via Z-register (16 k-words)
`adres RJMP,`	Relatieve sprong van + of - 2 k-words
`IJMP,`	Indirecte sprong via Z-register (16 k-words)

10.6 Controle structuren

De conditiecodes `bitnr SBS` en `bitnr SBC` verwachten het te testen bitnummer (in het statusregister) vooraf op de stack. Alle andere opcodes testen deze bits impliciet.

```
CS      CC      NZR     LE      ULE     IS      IC
UGE     GE      ZGE     ZER     HS      HC      ZLE
TS      TC      OS      OC      SBS     SBC
```

Nu de schrijfwijze

```
<test> IF,  ...   ELSE,  ...  THEN,
BEGIN,  ...  AGAIN,
BEGIN,  ... <test> UNTIL,
BEGIN,  ... <test> WHILE,  ...  REPEAT,
<reg> <const> ldi, DO,  ...  <reg> LOOP,
Prt bit SBIS, AHEAD,  ... THEN,
AHEAD,  BEGIN, ...  ENTRY,  <test> UNTIL,
```

Enkele voorbeelden van het gebruik:

`R16 20 CPI, ZER IF,`	Ga verder achter `IF,` als R16 20 bevat.
`R16 DEC, NZR IF,`	Verlaag `R16` en ga verder achter `IF,` zolang `R16` niet nul is.
`R16 TST, ZER IF,`	Ga verder achter `IF,` als R16 nul is.
`R16 R17 ADD, CS IF,`	Ga verder achter `IF,` als de carry vlag gezet is.

10.7 De bitinstructies van de AVR

De AVR kent een kleine bit-instructieset voor het manipuleren van I/O-bits. De `IN,` en `OUT,` instructie zorgen voor de rest van de in- en uitvoer manipulatie. Alleen de eerste 64 SFR-registers zijn via de `IN,` en `OUT,` te lezen en schrijven. De `SBI,` en `CBI,` instructie werken helaas alleen op de eerste 32 SFR-registers.

Instructies voor I/O bits:

```
Prt Bit SBI,            Prt Bit CBI,            Prt Bit SBIC,
Prt Bit SBIS,           <Rn> Prt IN,            Prt <Rn> OUT,
```

Er is een aparte set instructies voor bit-vlaggen. Ze hebben een eigen accumulator, de T-vlag in het statusregister. Elk bit in een van de 32 registers van de AVR kan een bit-vlag zijn.

Instructies voor bitvlaggen:

```
<Rn> Bit BST,           <Rn> Bit SBRC,          <Rn> Bit SBRS,
<Rn> Bit BLD,
```

10.8 Conversie operatoren

Er zijn een aantal conversie operatoren in AVR ByteForth opgenomen voor het omzetten van ByteForth formaten naar AVR assembly formaat. Hier een opsomming:

>REG	Zet registernummer om in register opcode data.
F>B	Zet vlagadres om naar een bitnummer.
>FL	Zet vlag adres om naar opcode-data voor BLD, en BST,.
B>M	Zet bitnummer om naar bitmasker.
>REAL	Zet variable offset om naar werkelijk RAM adres.
REAL>	Zet RAM adres om naar offset voor adressering via Y-reg.

10.9 Speciale functies

Er zijn enkele speciale functies in de assembler, zij leveren belangrijke adressen van het ByteForth systeem. Deze adressen kunnen verschillend zijn op een andere AVR, ze zijn afhankelijk van het gekozen type.

DSP0	Adres van de bodem v/d datastack.
RSP0	Adres van de bodem v/d returnstack.
RAMTOP	Hoogste RAM-adres van het ByteForth systeem.

10.10 Het gebruik van de assembler

Vijf code voorbeelden met gebruik van de AVR-assembler, zoals die ingebouwd zit in AVR ByteForth. Het eerste voorbeeld verwijderd het bovenste data item van de stack. Voorbeeld twee zet de constante waarde 10 op de stack, het toont hoe data op de stack geplaatst wordt, etc.

```
CODE DROP ( x -- )          \ Pop de top van stack
      XL INC,               \ Verhoog de stackpointer DSP met een
      RET,                  \ Klaar
END-CODE

CODE TIEN ( -- x )          \ Push getal op de top van stack
      R16 10 LDI,           \ Zet decimaal 10 in de accu (R16)
      -X R16 ST,            \ Verlaag de stackpointer DSP met een
                            \ Zet de accu op de top van de stack
      RET,                  \ Klaar
END-CODE

VARIABLE AAP                \ Reserveer RAM adres met de naam AAP
CODE VERHOOG ( u -- )       \ Verhoog AAP met u van de stack
      R16 X+ LD,            \ Pop u van de stack naar R16
      R17 ADR AAP [Y] LD,   \ Lees AAP naar R17
      R16 R17 ADD,          \ Tel AAP (R17) bij u (R16) op
      ADR AAP [Y] R16 ST,   \ Zet resultaat terug in AAP
      RET,                  \ Klaar
END-CODE

REGISTER TEL                \ Reserveer register met de naam TEL
CODE TELLER ( -- )          \ Verhoog TEL bij elke keer uitvoeren
      ADR TEL INC,          \ Register TEL wordt direct aangesproken
      RET,                  \ Klaar
END-CODE
```

```
FLAG NOOT?                  \ Reserveer bit-vlag met naam NOOT?
CODE NOOTJES ( -- 0|u )     \ Is bit waar, push AAP, anders 0
      R16 CLR,              \ Maak accu 0
      ADR NOOT? BST,        \ Lees status NOOT? in T-vlag
      TS IF,                \ Is NOOT? (T-vlag) Waar ?
        R16 ADR AAP [Y] LD, \ Ja, zet AAP in accu
      THEN,
      -X R16 ST,            \ Push accu op de stack
      RET,                  \ Klaar
END-CODE

PORTB 7 BIT-SFR UITGANG     \ Geef PB.7 de naam UITGANG
CODE PULS ( -- )            \ Zet hoge puls op UITGANG
      ADR UITGANG SBI,      \ Maak UITGANG hoog
      NOP,                  \ Verleng puls
      ADR UITGANG CBI,      \ Maak UITGANG laag
      RET,                  \ Klaar
END-CODE
```

Vergeet niet dat een I/O-poort op de AVR een apart richtingsregister heeft. In dit register moeten de 'uitvoer'-bits hoog gezet zijn en de 'invoer'-bits laag. Anders gebeurt er niets.

10.11 De Forth schrijfwijze

Vergelijk de schrijfwijze van de fabrikant (zie bladzijde 130) met de schrijfwijze die AVR ByteForth voor assembly gebruikt.

Atmel schrijfwijze	ByteForth	Functie
`MOV R31,R1`	`R31 R1 MOV,`	Zet R1 in R31.
`LD R16,X`	`R16 X LD,`	Zet inhoud adres uit X-reg. in R16.
`STS 160,R16`	`160 R16 STS,`	Zet R16 in adres 160.
`PUSH R0`	`R0 PUSH,`	Zet R0 op de stack.
`COM R16`	`R16 COM,`	Inverteer register R16.
`NOP`	`NOP,`	Doe 1 kloktik niets.
`SBI PORTB,5`	`PORTB 5 SBI,`	Zet bit 5 van PORTB hoog.
`ORI R16,5`	`R16 5 ORI,`	Logische OR van R16 en 5.
`CPI R24,20`	`R24 20 CPI,`	Vergelijk R24 met 20.
`ST -X,R16`	`-X R16 ST,`	Sla R16 op in het adres uit het X-reg en verlaag X met 1.
`IN R16,PIND`	`R16 PIND IN,`	Lees de ingangen van poort-D uit en zet ze in R16.
`LDD R20,Y+10`	`R20 10 [Y] LD,`	Gebruik het adres in het Y-register als pointer+10, lees de inhoud van het resulterende adres naar R20.
`LPM`	`LPM,`	Lees de inhoud van het Flashadres waar de Z-pointer naar wijst, naar R0.
`BST R4,5`	`R4 5 BST,`	Lees bitvlag 5 uit R4 naar de T-vlag in het statusregister.
`BLD R4,5`	`R4 5 BLD,`	Zet de T-vlag uit het statusregister naar bitvlag 5 in R4.
`ADIW XL,4`	`XL 4 ADIW,`	Verhoog de 16-bits pointer in het X-register met 4.

A Wat is er nieuw t.o.v. 8051 ByteForth

AVR ByteForth 2.00 is geheel nieuw ontwikkeld en gelijk gemaakt aan de 8051 ByteForth versie 2.00. De verbeteringen zijn een nog verder uitgebreide optimalisator en net zo'n fraaie symbolische disassembler/decompiler als in de 8051 versie. De ByteForth systemen versie 2.00 zijn meer interactief (gedragen zich meer als een gewone Forth). De ISP-programmer werkt via een driver voor de parallele poort. Vanaf deze versie is ByteForth er alleen nog voor de PC en wordt er geen versie meer gemaakt voor het ATS-bord. Een lijst van de wijzigingen:

- Een uitgebreide set CHForth afdrukinstructies is toegevoegd aan de debugger.
- Een ISP programmer is ingebouwd voor de ATtiny13 t/m de ATmega64.
- VALUE een 8 bits TO-variabele is toegevoegd.
- De optimalisator is verder uitgebreid met vele speciaal geval optimalisators.
- Macro's uitgebreid met o.a. high-level interrupt ondersteuning.
- Het is nu ook mogelijk om zelf macro's toe te voegen.
- CREATE en DOES> zijn toegevoegd.
- Het display van de ingebouwde tracer is verbeterd.
- Vanaf ByteForth versie 2.00 zijn er gelijke versies voor 8051- en AVR-reeks.
- Bibliotheek files uitgebreid met o.a. grafisch LCD, ADC, etc.
- Een stuk of tien nieuwe voorbeeld toepassingen bijgevoegd.
- Alle voorbeelden uit het 'Egel boek werken ook op deze versie. (Het 'Egel werkboek is te verkrijgen via de HCC Forth-gg).
- Er zijn verschillende nieuwe ontwikkelsysteempjes bijgekomen, voor de AT90S1200 en AT90S2313 het AT51-2 printje, voor de AT90S8515, ATmega8515 en ATmega161 in 44-pens PLCC behuizing het AT8252 printje. Verder is er nog de Ushi-robot met opsteekprintjes voor de AT90S2313, AT90S4433 en ATmega8, ATtiny26 en voor de ATmega16/32.
- —- **Voor versie 2.07** —-
- Vier nieuwe AVR's toegevoegd waaronder de ATmega48 en ATtiny2313.
- Enkele bugs uit de ISP-programmer verwijderd.
- De decompiler kan nu bijna alle AVRF datastructuren decompileren.
- Naast VARIABLE in al zijn variaties is nu ook VALUE in dezelfde variaties aanwezig, echter in een niet standaard vorm.
- Bouwbeschrijvingen van de nieuwe schakelaar- en LCD-print zijn in dit boek opgenomen, evenals een apart hoofdstuk over AVR bijzonderheden.
- De dongle bouwbeschrijving is aangepast aan de nieuwe dubbelzijdige print en fouten in diverse schema's zijn verbeterd.

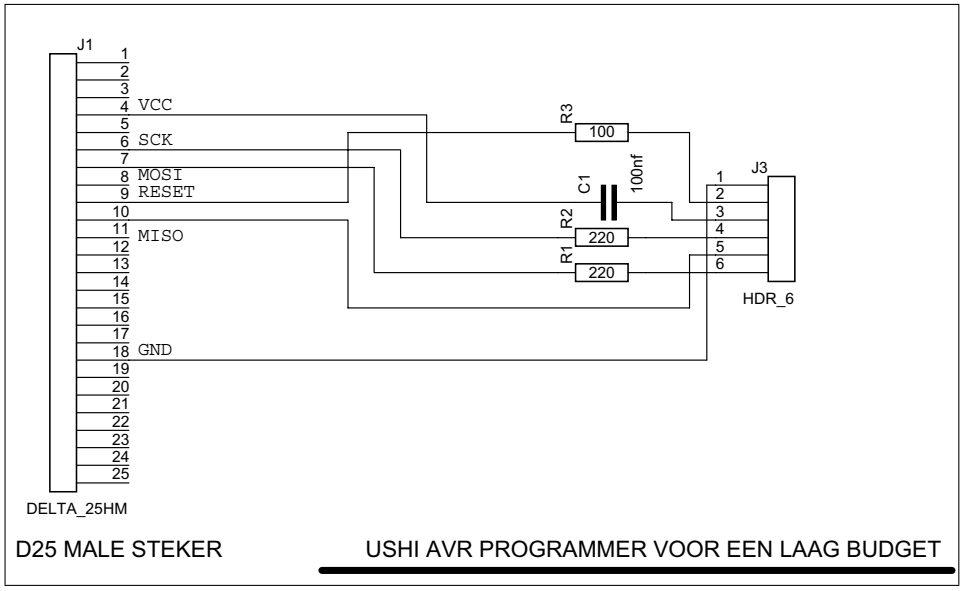
B El Cheapo dongle

De zogenaamde 'El cheapo' interface is de goedkoopste manier om AVR ByteForth uit te proberen. Samen met de demo versie van AVR ByteForth, een AT90S2313 en enkele onderdelen ben je voor ongeveer 10 Euro klaar.

B.1 Componentenlijst van El Cheapo

R1	220Ω	Weerstand 1/10 Watt
R2	220Ω	Weerstand 1/10 Watt
R3	100Ω	Weerstand 1/10 Watt
C1	100nf	Weerstand 1/10 Watt
DR1	bandkabel	6-polige bandkabel van 50 cm
J1	D25-MALE	DB25-male soldeer
J3	HDR6	6 polige female header
H1	Sub-D kap	Kap voor DB25-connector

B.2 Schema van El Cheapo



Figuur B.1: Schema van El Cheapo

B.3 Bouwbeschrijving van El Cheapo

1) Soldeer de weerstanden R1, R2 en R3 aan de DB25-connector J1.
2) Soldeer de condensator C1 daar ook aan.
3) Soldeer de 6 polige kabel DR1 nu vast volgens het schema.
4) Zet de 6 polige female header J3 aan de andere kant v/d kabel, vergeet niet krimpkous om elke draad te doen. De massaansluiting (Gnd) moet van een zwart stukje krimpkous voorzien worden als markering.
5) Plaats de trekontlasting op de kabel en zet de kap H1 op zijn plaats.

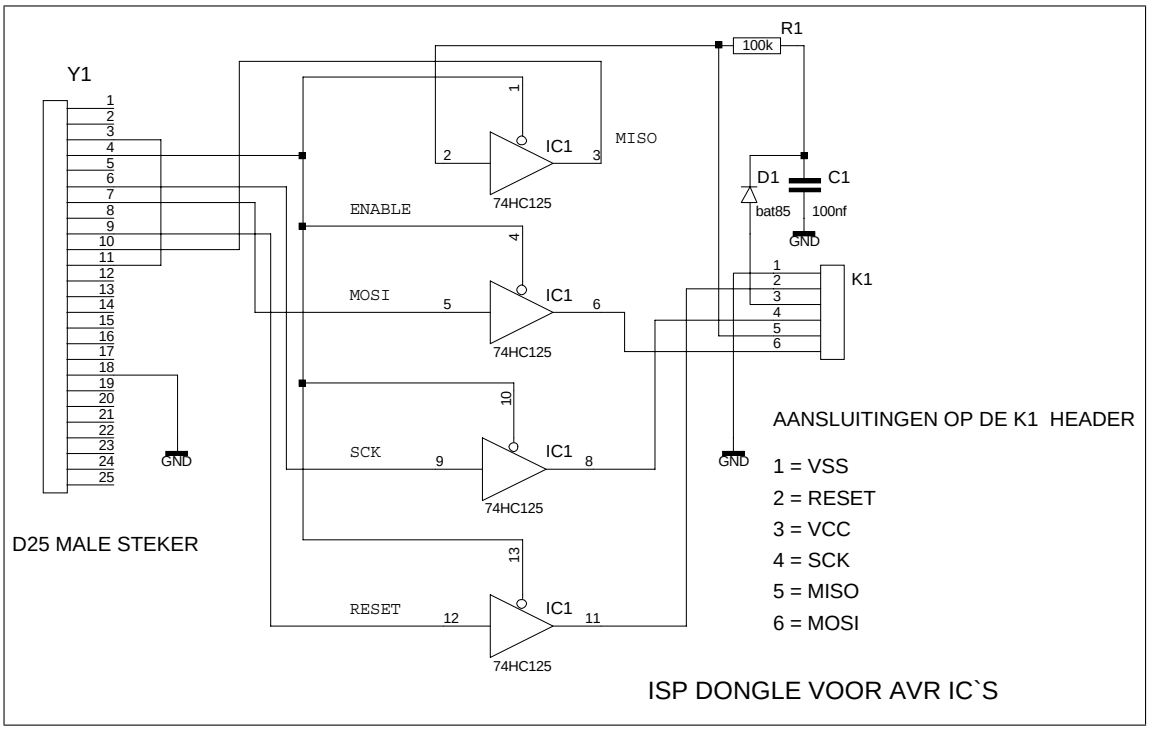
C AVR ByteForth ISP-dongle

De AVR ByteForth dongle is functioneel indentiek aan die van de bekende STK200, STK300 en STK500 starterkits. Onze eigen connector gebruikt echter een andere layout, 6 pennen in lijn, hierdoor kunnen we op zeer klein printjes werken. Zoals b.v. gebruikt voor het Ushi robotproject.

C.1 Componentenlijst van dongle

C1	100nF	Keramische C, steek 2,5mm
D1	BAT85	Diode of 1N4148
IC1	74HC125	14 polig DIL
K1	bandkabel	6-polige bandkabel van 50 cm
K2	HDR6	6 polige female header
R1	100kΩ	Weerstand 1/10 Watt
Y1	D25-MALE	DB25-male soldeer
H1	Sub-D kap	Kap voor DB25-connector

C.2 Schema van dongle

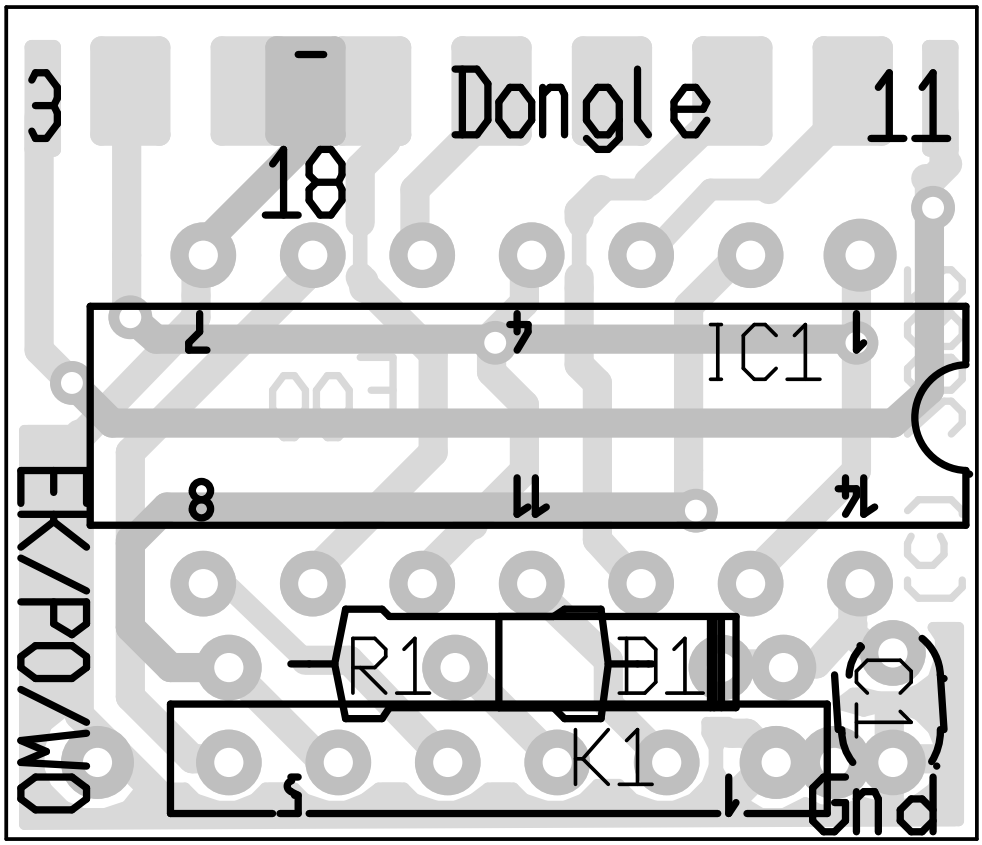


Figuur C.1: Schema van dongle

C.3 Bouwbeschrijving van dongle

1) Soldeer nu de eerst de diode D1, dan weerstand R1 daarbovenop en daarna condensator C1.
2) Plaats IC1 direct (zonder IC-voet) op de print en soldeer hem vast.
3) Verwijder, indien nog aanwezig, eerst de koperen snijlijn rond de print aan de kant van de DB25 connector. Let op die loopt aan beide zijden van de dubbelzijdige print. Hiermee voorkom je kortsluiting op de parallel-poort. Stop de print op de goede plaats tussen de DB25 soldeerpennen, de buitenste pennen zijn 3 en 11. Soldeer de print op beide zijden aan de pennen vast.
4) Soldeer het contact aan de componentenzijde ook aan de DB25 connector vast en soldeer de bandkabel K1 op de print.
5) Zet de 6 polige female header K2 aan de andere kant v/d kabel, vergeet niet krimpkous om elke draad te doen. De massaansluiting (Gnd) moet van een zwart stukje krimpkous voorzien worden als markering.
6) Plaats de trekontlasting op de kabel en zet de kap H1 op zijn plaats.

C.4 Componenten plaatsing van dongle

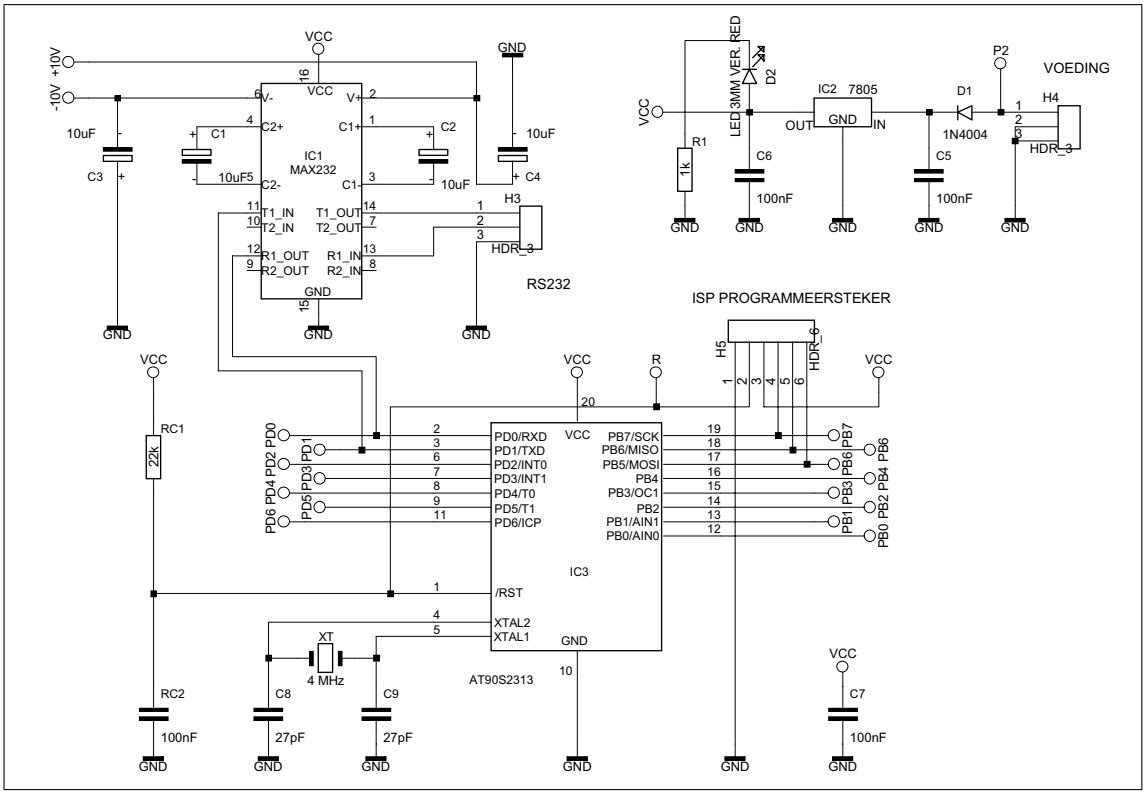


Figuur C.2: Printbezetting van dongle

D AT51 versie-2 print

De AT51-print was in eerste instantie ontwikkeld voor mijn eigen experimenten met 8051 ByteForth voor de AT89C2051. Toen het 'egelwerkboek ontwikkeld werd hebben we het als basis gekozen voor de 21 projecten die daarin beschreven staan. Nu AVR ByteForth beschikbaar komt heb ik de print aangepast zodat hij ook gebruikt kan worden met de AT90S2313 en AT90S1200 van ATMEL.
De belangrijkste wijzigingen zijn; ISP-programmeersteker voor AVR, kristal onder de chip geplaatst zodat meer printruimte beschikbaar is, de 7805 stabilisator is nu zo geplaatst dat hij van een grote koelplaat voorzien kan worden, de coördinaatbenaming uit het 'egelwerkboek is er opgedrukt.

D.1 Schema van de AT51-2

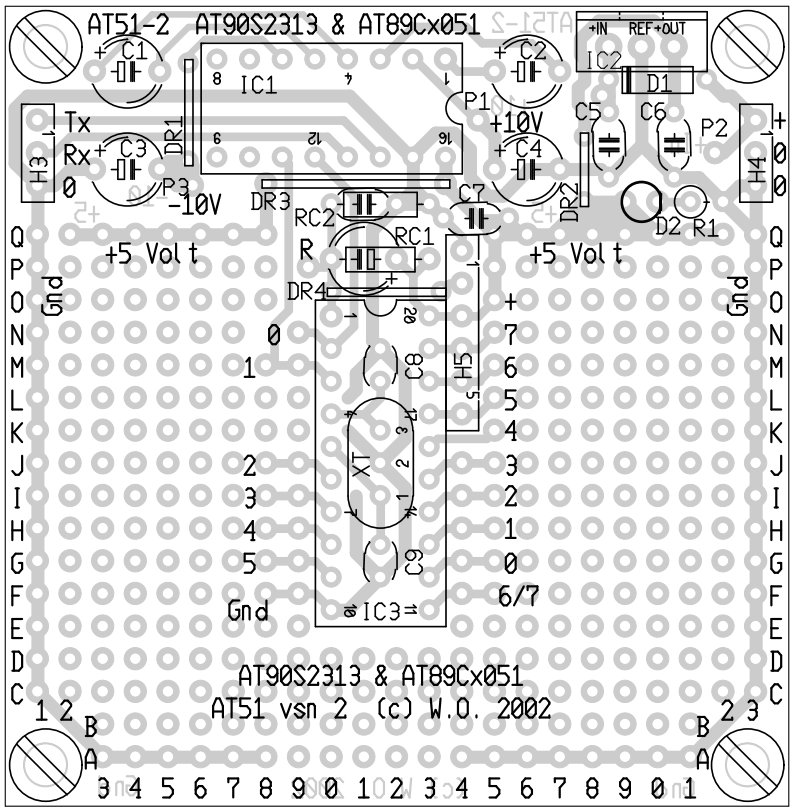


Figuur D.1: Schema van de AT51-2

D.2 Bouwbeschrijving van de AT51-2

1) Breng eerst alle draadbruggen aan.
2) Daarna de weerstanden.
3) Let op, RC1 is voor de AT90S2313 een 22kΩ-weerstand, en RC2 is een condensator van 100nf steek 5mm.
4) De IC-voeten, let op zaag de middenbrug uit het 20-polige IC-voetje anders past het kristal er niet onder.
5) Het kristal en de condensatoren.
6) Tenslotte de 78(S)05.

D.3 Componenten plaatsing van de AT51-2



Figuur D.2: Tekening van de AT51-2 print

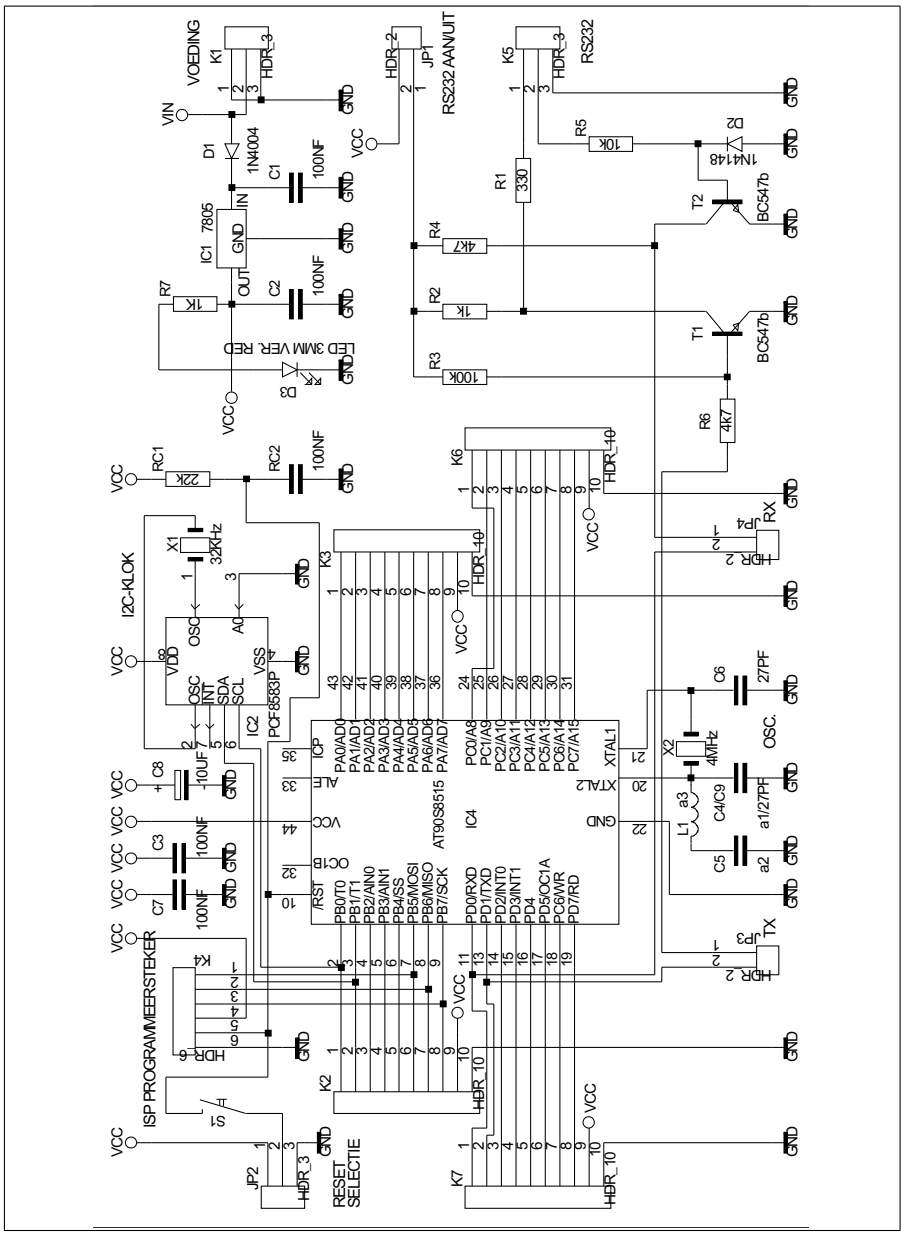
D.4 Componentenlijst van de AT51-2

C1	10μF	Staande elco
C2	10μF	Staande elco
C3	10μF	Staande elco
C4	10μF	Staande elco
C5	100nF	Keramische C, steek 5mm
C6	100nF	Keramische C, steek 5mm
C7	100nF	Keramische C, steek 5mm
C8	27pF	Keramische C, steek 2,5mm
C9	27pF	Keramische C, steek 2,5mm
D1	1N4001	Diode
D2	LED	3mm led
DR1	DRAAD	draadbrug
DR2	DRAAD	draadbrug
DR3	DRAAD	draadbrug
DR4	DRAAD	draadbrug
H3	HDR3	3 polige male header
H4	HDR3	3 polige male header
H5	HDR6	6 polige male header
IC1	MAX232	16 polig DIL-chip en losse voet
IC2	7805	TO220
IC3	AT90S2313	20 polig DIL-chip en losse voet
R1	1kΩ	1/10 Watt
RC1	22kΩ	1/10 Watt
RC2	100nF	Keramische C, steek 5mm
XT	XT of RES	Kristal of resonator

E AT8252 print (versie-1)

De AT8252-print is in eerste instantie ontwikkeld voor de AT89S8252 en familieleden als de AT89S51, etc. voor gebruik met 8051 ByteForth versie 2.00 voor de 8051-familie. Omdat zowel de AT90S8515 en de ATmega161 een zelfde pinout hebben heb ik het bord aangepast (alleen de reset werkt anders). Daarom is de print nu bruikbaar met zowel 8051- als AVR ByteForth.
De print bevat een RS232-interface, reset drukknop, vier 10-polige headers voor evenzovele I/O-poorten, ISP-connector, brownout detector, I2C RTC of EEPROM, flexibele kristal oscillator, etc.

E.1 Schema van de AT8252

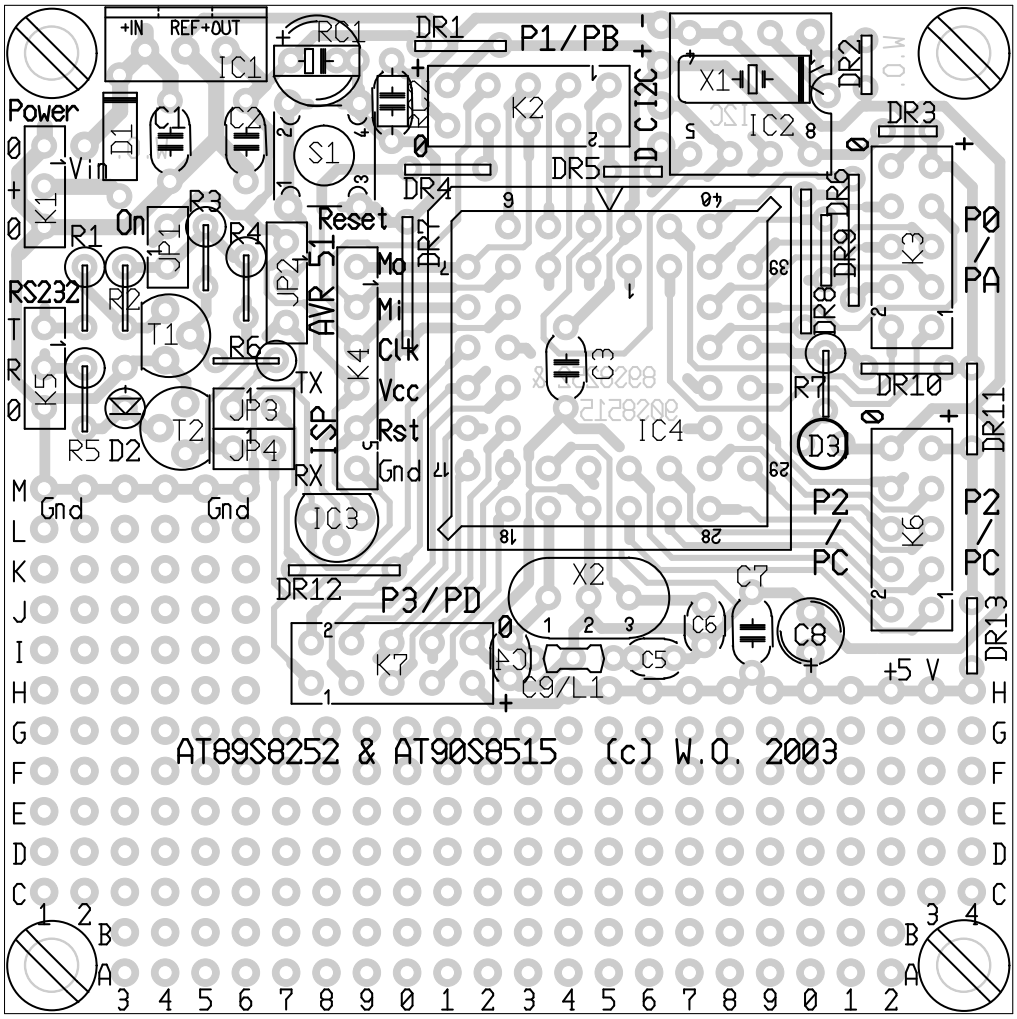


Figuur E.1: Schema van de AT8252

E.2 Bouwbeschrijving van de AT8252

1) Breng eerst alle draadbruggen aan, behalve DR2. Leg een draad tussen de gaten van JP2 waar AVR naast staat.
2) Daarna de weerstanden.
3) Let op, RC1 is voor de ATmega8515 een 22kΩ-weerstand, en RC2 is een condensator van 100nf steek 5mm.
4) Nu de IC-voeten, let op zaag de gaten uit de 44-polige IC-voet anders past de condensator C3 er niet onder.
5) Het kristal en de condensatoren.
6) Tenslotte de connectoren en dan de 78(S)05.

E.3 Componenten plaatsing van de AT8252



Figuur E.2: Tekening van de AT8252-print

E.4 Componentenlijst van de AT8252

C1	100nF	Keramische C, steek 5mm
C2	100nF	Keramische C, steek 5mm
C3	100pF	Keramische C, steek 5mm
C4	a1nF	Keramische C, steek 2.5mm
C5	a2pF	Keramische C, steek 2.5mm
C6	27pF	Keramische C, steek 2.5mm
C7	100nF	Keramische C, steek 5mm
C8	10μF	Staande elco
C9	27pF	Keramische C, steek 2.5mm
L1	a3μH	Printspoel, steek 5mm
D1	1N4004	Diode
D2	1N4148	Diode
D3	LED2MM	Led 3mm
DR1	DRAAD	Draadbrug
DR2	DRAAD	Draadbrug
DR3	DRAAD	Draadbrug
DR4	DRAAD	Draadbrug
DR5	DRAAD	Draadbrug
DR6	DRAAD	Draadbrug
DR7	DRAAD	Draadbrug
DR8	DRAAD	Draadbrug
DR9	DRAAD	Draadbrug
DR10	DRAAD	Draadbrug
DR11	DRAAD	Draadbrug
DR12	DRAAD	Draadbrug
DR13	DRAAD	Draadbrug
IC1	78S05	TO220
IC2	PCF8583	8 polige DIL & voet
IC3	TL(C)7757	TO92 (niet nodig voor ATmega8515)
IC4	ATmega8515	44 polige PLCC & voetje
JP1	HDR2	2 polige male header
JP2	HDR3	3 polige male header
JP3	HDR2	2 polige male header
JP4	HDR2	2 polige male header
K1	HDR3	3 polige male header
K2	HDR2X5	2x5 polige male header
K3	HDR2X5	2x5 polige male header
K4	HDR-6	6 polige male header
K5	HDR3	3 polige male header
K6	HDR2X5	2x5 polige male header
K7	HDR2X5	2x5 polige male header
R1	330Ω	1/10 Watt
R2	1kΩ	1/10 Watt
R3	100kΩ	1/10 Watt
R4	47kΩ	1/10 Watt
R5	10kΩ	1/10 Watt
R6	4k7	1/10 Watt
R7	1kΩ	1/10 Watt
RC1	22kΩ	1/10 Watt
RC2	100nF	Keramische C, steek 5mm
S1	DRUKKNOP	6x6mm, type 3305
T1	BC547b	TO92
T2	BC547b	TO92
TP1	PIN	Soldeerpen
X1	32KHz	Horlogekristal
X2	XT of RES	Kristal of resonator

TEMIC
Semiconductors

ANM032

How to use a Third Overtone Crystal with a 80C51 Family Microcontroller

Description

For cost reason using an overtone crystal is 5 to 6 times cheaper than a fundamental one. Using this type of crystal is slightly different comparing to a fundamental one . The frequency of an overtone crystal is adjusted on the fundamental one and this one must be trapped by a LC pass–band filter. The typical schematic is shown below.

CP1 and CP2 are the parasitic capacitors due to the packaging and the PCB lay–out. L1 and C1 is the passe–band filter used to trap the fundamental frequency. C2 is a small capacitor to increase a little bit the open–loop gain given by:

$$A \times B = A \times \frac{CP2 + C2}{CP1}$$

where A is the gain a the operating frequency and B is the gain of the feed–back . The frequency of the filter is given below:

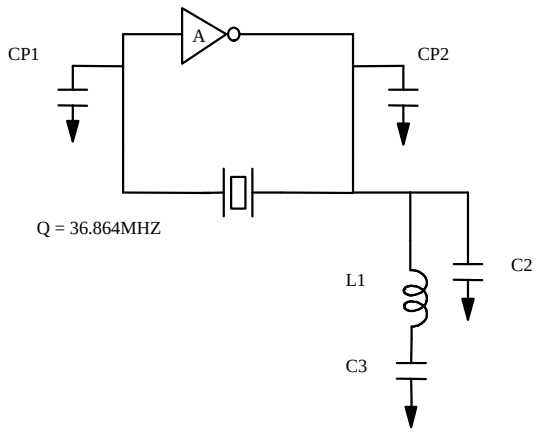
$$f_T = \frac{F_Q}{3} = \frac{36.864}{3} = 12.288\ MHz$$

$$L_1 = \frac{1}{(2 \times \pi \times f_T)^2 \times C3}$$

Where C3 = 33 pF

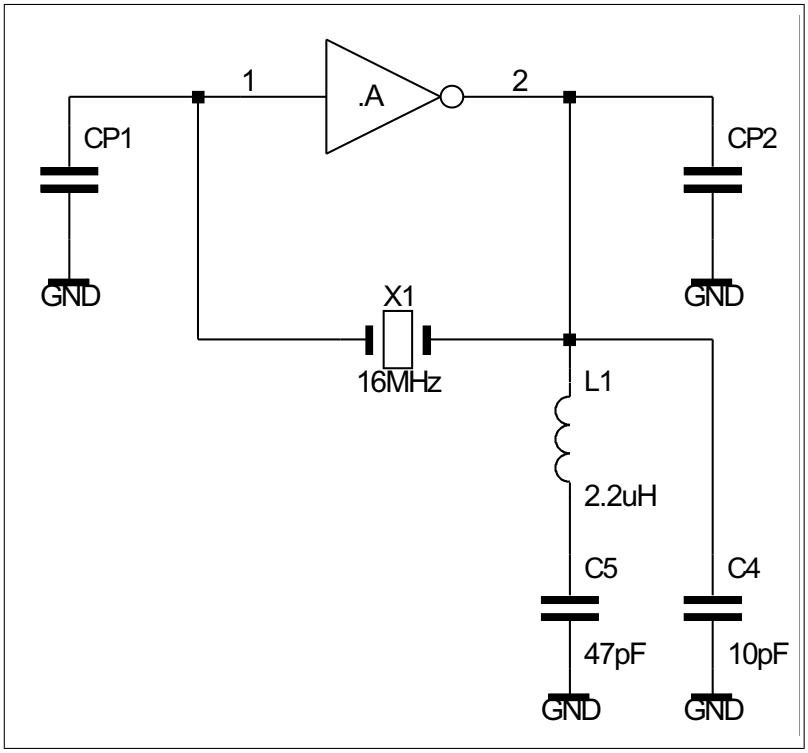
$$L_1 = \frac{1}{(2 \times \pi \times 12.288 \times 10^6) \times (39 \times 10^{-12})} = 4.3\mu H$$

The standard one is 4.7µH and not critical because the bandwidth is large enough . C2 is chosen to be equal to 10pf (a larger value break–down the amplifier and the open loop gain) .



Figuur F.1: Derde boventoon datasheet van Atmel

F.1 Oscillator schema voor AT8252-print



Figuur F.2: Oscillatorschema aangepast aan de AT8252-print

F.2 Formule aangepast voor de AT8252-print

De laatste formule in het datasheet is niet correct afgedrukt, het kwadraat ontbreekt achter de linker berekening onder de streep. Daarom hier de formule nogmaals. De formule is voor het gemak aangepast aan de componentennummering op de AT8252-print.
De condensatoren CP1 en CP2 zijn de capaciteiten van de oscillator .A in de AVR-microcontroller en de printlayout.

$$L1 = \frac{1}{(2 * \pi * fX1)^2 * C5}$$

Uitgewerkt voor het 16 MHz kristal is dit:

$$L1 = \frac{1}{(2 * \pi * 16 * 10^6)^2 * 47 * 10^{-12}} = 2.105 \mu H$$

Afgerond naar een standaard component is L1 dan 2.2 μH.

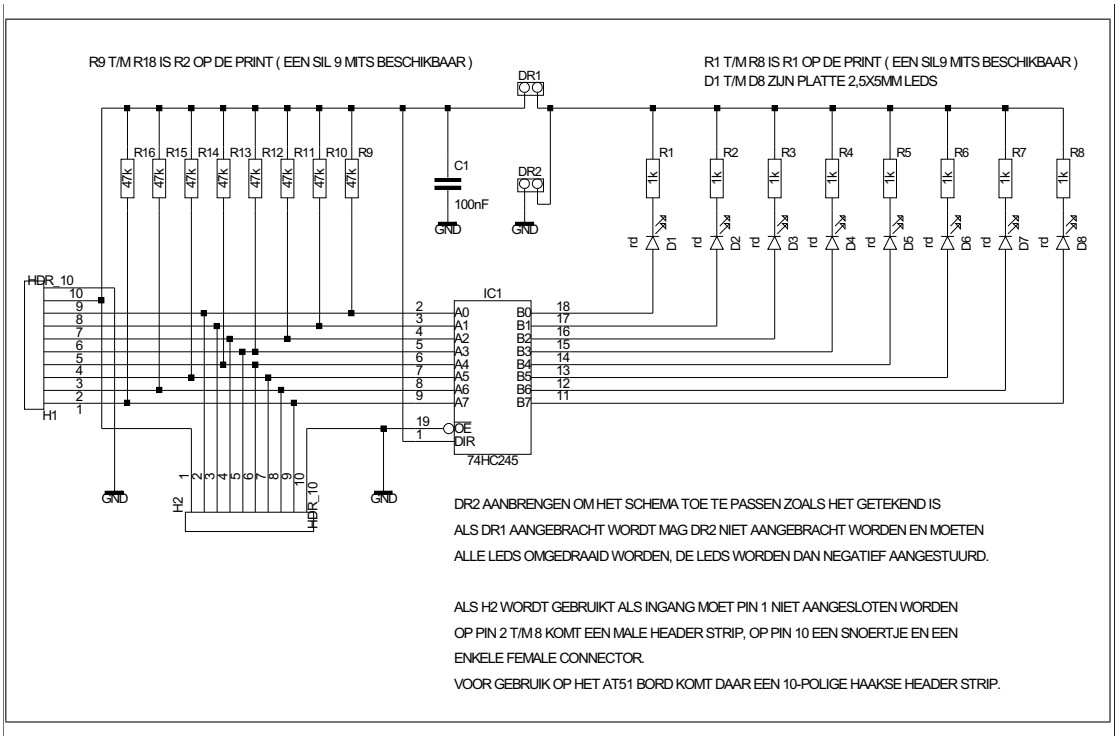
F.3 Enkele voorbeelden

Kristalfreq. (X1)	L1	C4	C5
16 MHz	2.2μH	10pF	47pF
24 MHz	1.5μH	10pF	27pF
33 MHz	1.5μH	10pF	15pF

G LED-print

De LED-print is ontwikkeld voor experimenten op het ATS-bord. De nieuwe print is echter breder inzetbaar, er kan gekozen worden tussen positieve en negatieve aansturing. Positieve aansturing is standaard. Door een stukje van de print af te knippen kan er een haakse sil-10 connector op gezet worden. Hiermee is de ledprint ook bruikbaar op experimenteerborden met verende contacten.

G.1 Schema van de LED-print

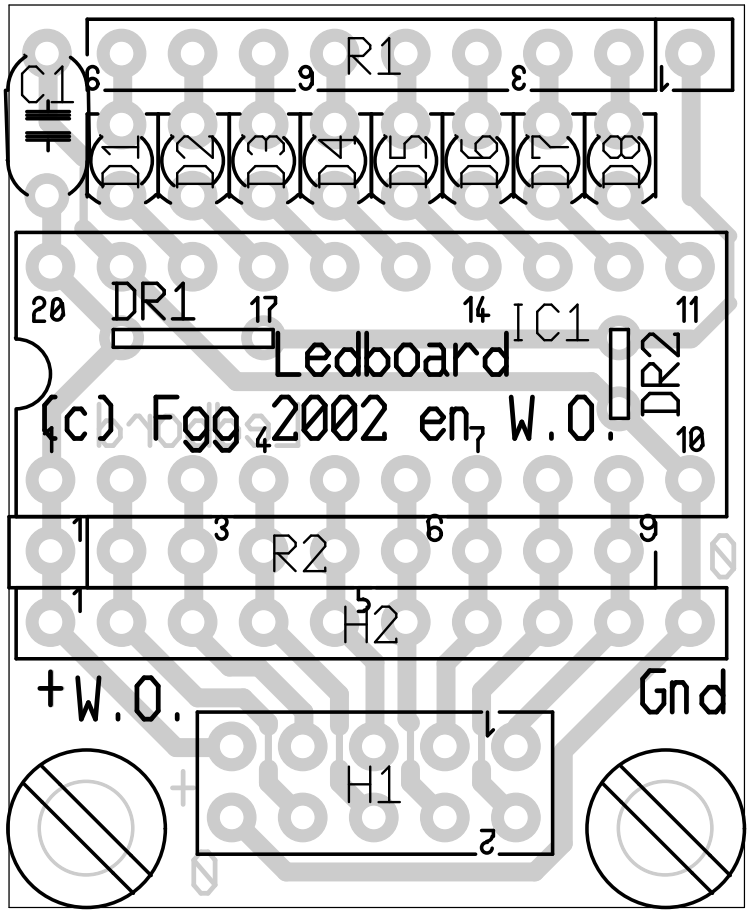


Figuur G.1: Schema van de LED-print

G.2 Bouwbeschrijving LED-print

1) Breng eerst de gewenste draadbrug aan.
2) Daarna de weerstanden.
3) De IC-voet.
4) De condensator.
5) Tenslotte de headers.
6) Maak een draadje van 5cm met aan een uiteinde een enkel female contact. Soldeer het andere uiteinde in pin 10 van H2.
7) Tenslotte de acht leds, i.p.v. 8 losse leds, kan ook de Conrad component 185760-44 gebruikt worden, die bevat 8-leds tesamen en past zo op de print. Let bij de leds goed op de doorlaat richting!

G.3 Componenten plaatsing LED-print



Figuur G.2: Tekening van de LED-print

G.4 Componentenlijst LED-print

C1	100nF	Keramische C, steek 5mm
D1	LED	2x5mm led
D2	LED	2x5mm led
D3	LED	2x5mm led
D4	LED	2x5mm led
D5	LED	2x5mm led
D6	LED	2x5mm led
D7	LED	2x5mm led
D8	LED	2x5mm led
DR1	DRAAD	draadbrug
DR2	DRAAD	draadbrug
H1	HDR10	2x5 polige male header
H2	HDR9	9 polige male header
IC1	74HC245	20 polig DIL-chip en losse voet
R1	1kΩ	sil-9
R2	47kΩ	sil-9

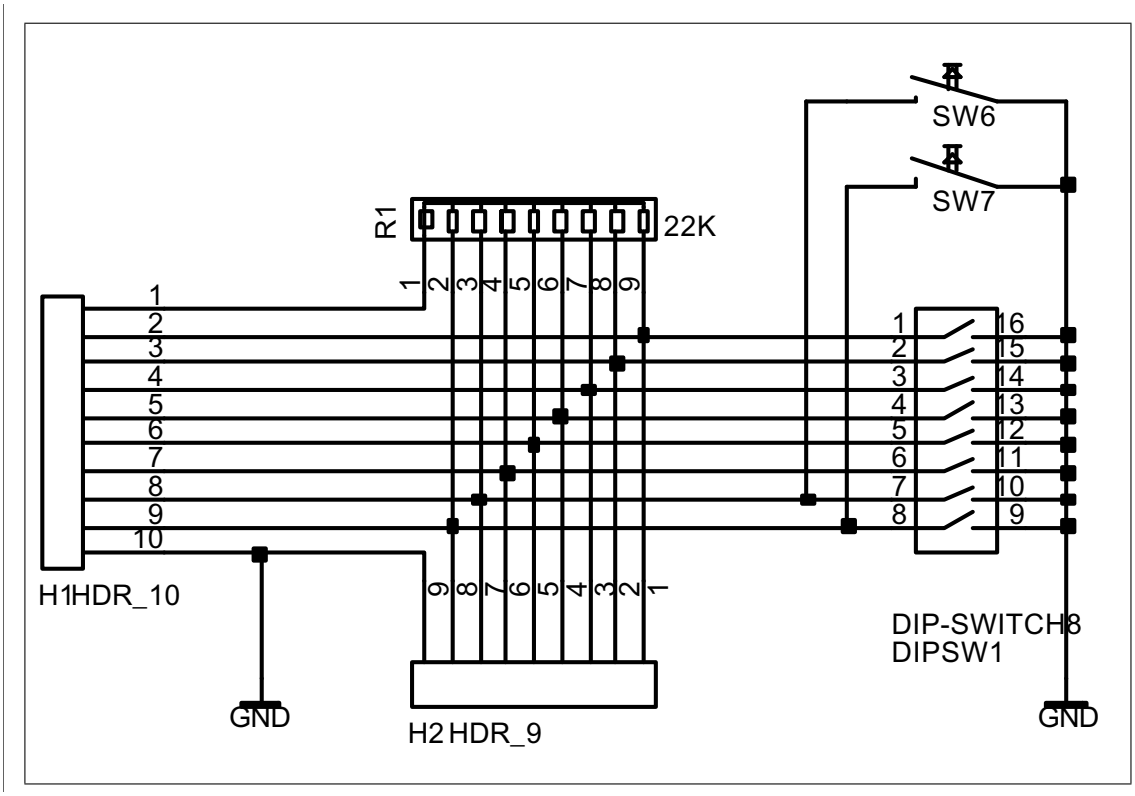
H Een print voor schakelaars

Voor zowel het ATS-bord als de S8252-print en AVRex zijn intussen enkele experimenteerprintjes vervaardigd. Deze schakelaarprint is er een van, de anderen zijn de ledprint en LCD-print. Grotere experimenteerprinten zijn de 'Egel I2C- en zevensegement printen.

H.1 De schakelaarprint

Met behulp van de schakelaaraanpassingsprint kan een standaard DIP-switch en twee druktoetsen op een simpele manier op ATS- of S8252-print aangesloten worden.

H.1.1 Schema van de schakelaarprint

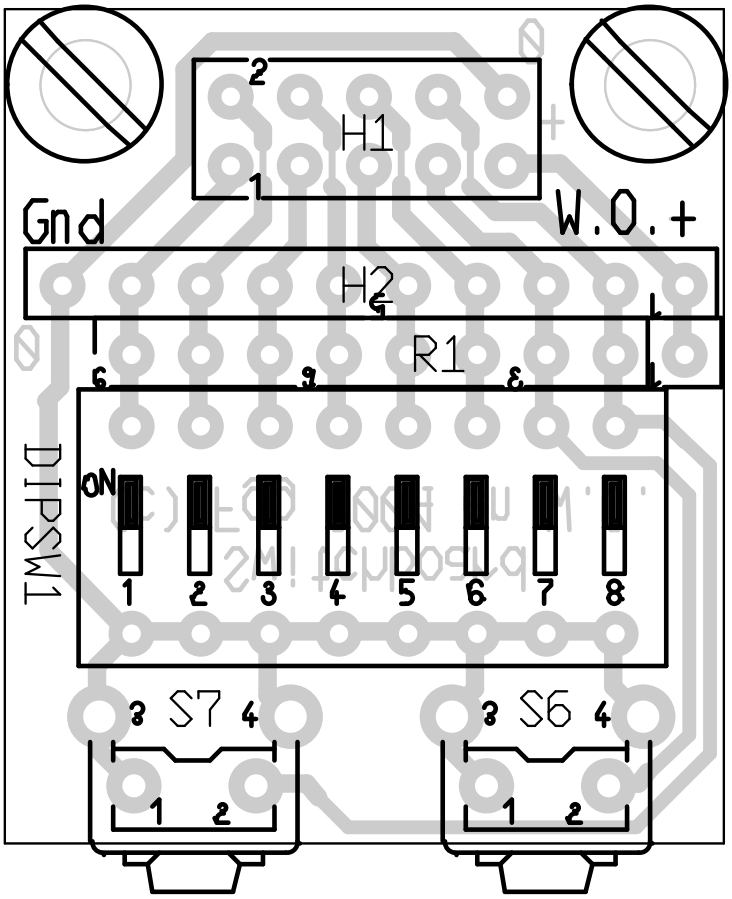


Figuur H.1: Schema van de Schakelaar aanpassingsprint

H.1.2 Bouwbeschrijving van de schakelaarprint

1) Soldeer SIL-weerstand R1.
2) Soldeer de dipswitch DIPSW1 op zijn plek.
3) De 10-polige male headerstrip H1.
6) Tenslotte de twee druktoetsen S6 en S7.

H.1.3 Componenten plaatsing van de schakelaarprint



Figuur H.2: Tekening van de schakelaarprint

H.1.4 Componentenlijst voor de schakelaarprint

R1	47kΩ	SIL 1/10 Watt
H1	HDR 2x5	2x5 polige male header
DIPSW1	Dipsitch	8 polige dipswitch
S6	Druktoets	Haakse druktoets voor print
S7	Druktoets	Haakse druktoets voor print

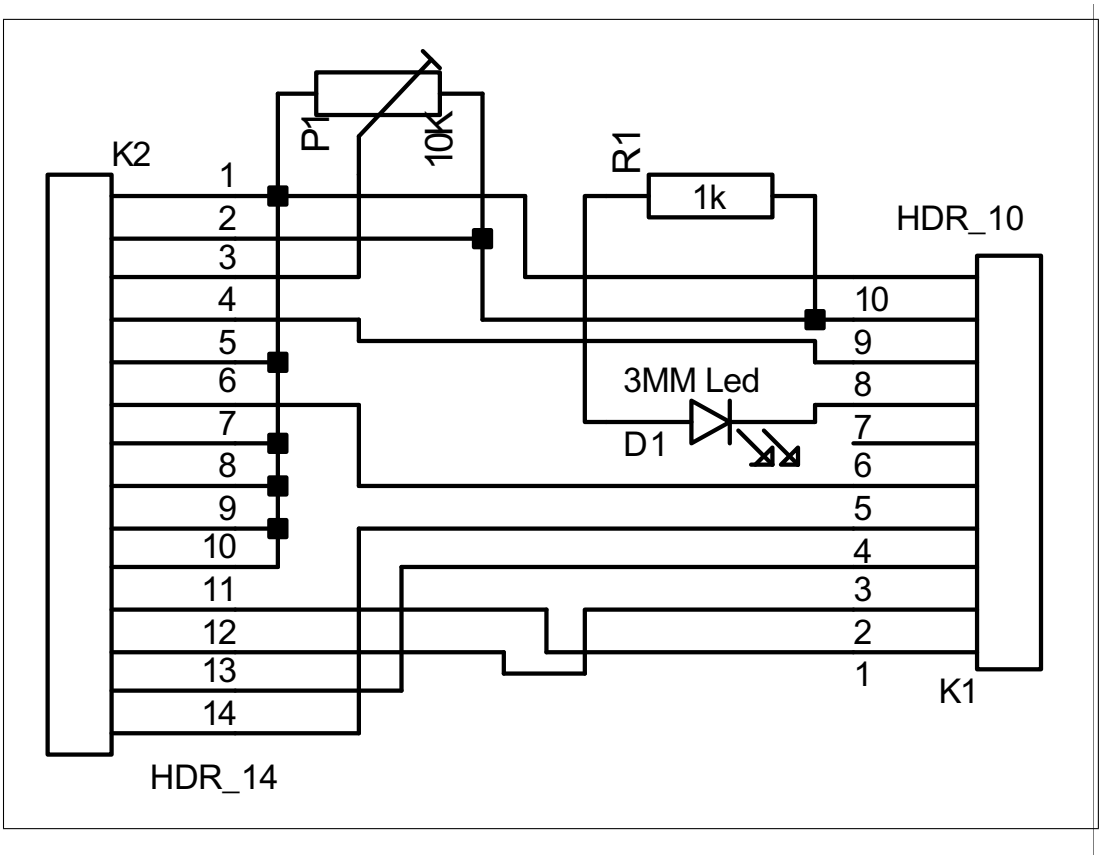
I De LCD aanpassingsprint

Voor zowel het ATS-bord als de S8252-print en AVRex zijn intussen enkele experimenteerprintjes vervaardigd. Deze LCD-print is er een van, de anderen zijn de ledprint en schakelaarprint. Grotere experimenteerprinten zijn de 'Egel I2C- en zevensegement printen.

I.1 Een LCD adapter

Met behulp van de LCD-aanpassingsprint kan een standaard karakter-LCD op een simpele manier op ATS- of S8252-print aangesloten worden. Dit vereenvoudigd het op gang komen van deze speciale hardware sterk.

I.1.1 Schema van de LCD-print

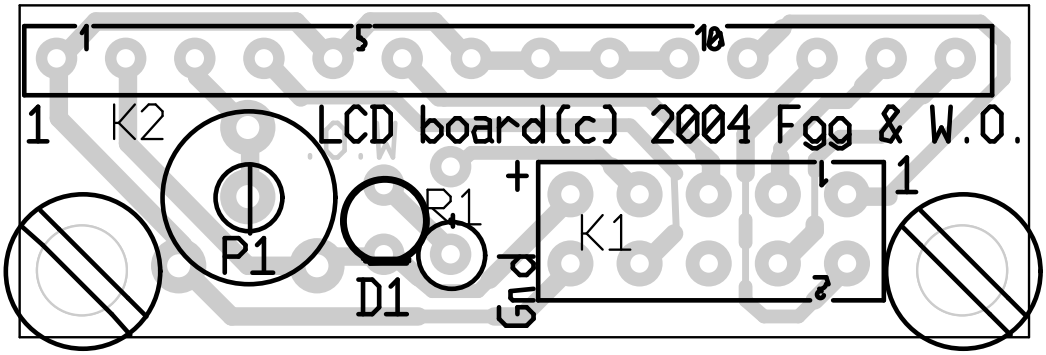


Figuur I.1: Schema van de LCD aanpassingsprint

I.1.2 Bouwbeschrijving van de LCD-print

1) Soldeer de potmeter P1 en de weerstand R1.
2) Soldeer de led D1 op zijn plek.
3) De 10-polige male headerstrip K1.
6) Tenslotte 14-polige male headerstrip K2.

I.1.3 Componenten plaatsing van de LCD-print



Figuur I.2: Tekening van de lcd aanpassingsprint

I.1.4 Componentenlijst voor de LCD-print

R1	1kΩ	1/10 Watt
K1	HDR 2x5	2x5 polige male header
K2	HDR 1x14	1x14 polige male header
D1	Led	Rode of groene 3mm Led

J De Ushi robot

Ushi is een robot opgebouwd uit gangbare materialen zoals servomotoren, standaard electronicacomponenten, een accuset en b.v. een AVR microcontroller.

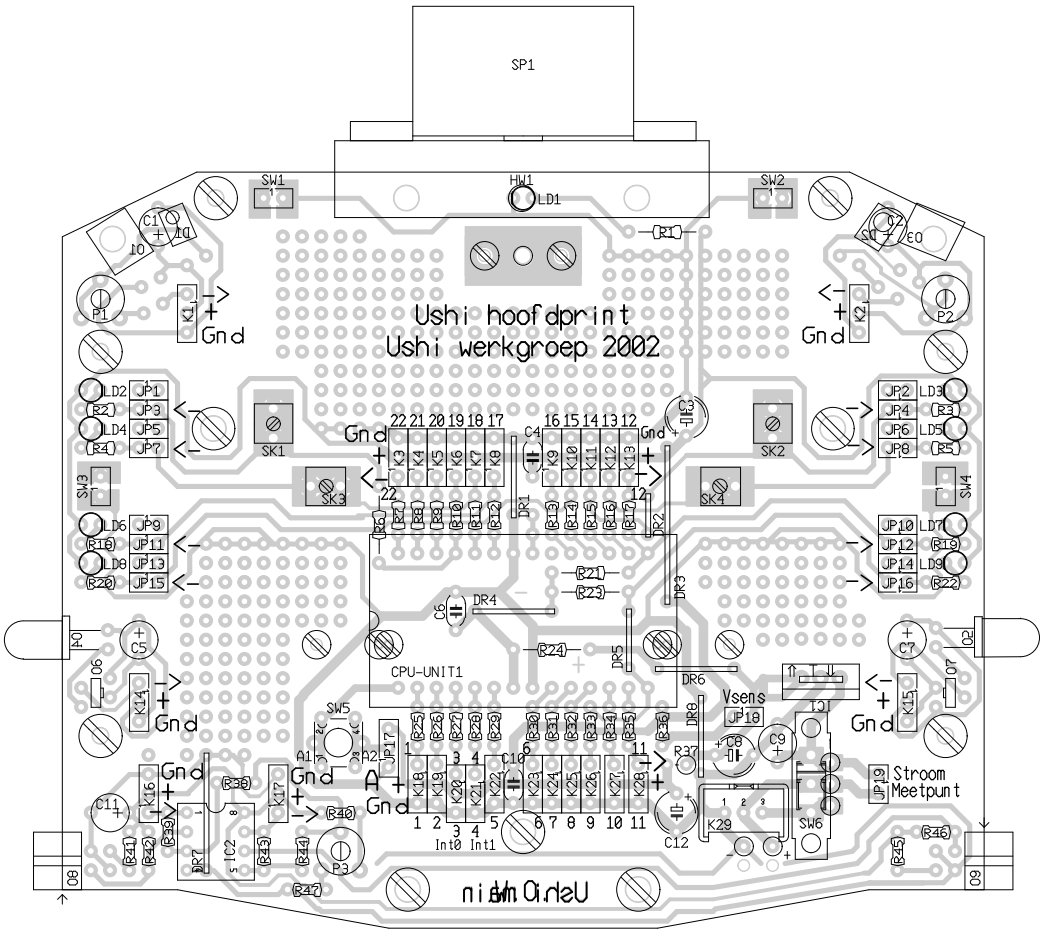
J.1 Wat kun je met Ushi?

Ushi is bedoeld om spelenderwijs de programmeertaal Forth te leren d.m.v. het spelen van spelletjes, alleen of in competitie met andere Ushi's. Al experimenterend verschillende AVR processoren leren kennen, etc. Ushi is qua pinout van het CPU-opsteek printje gebaseerd op de AT90S4433 en ATmega8 microcontroller van de firma Atmel. Op Ushi zijn ook andere microcontroller-systemen van de HCC-Forth-gg bruikbaar. Naast de bovengenoemde types zijn ook de oudgedienden als de AT89C2051 en het ATS-bord met de 80C535 bruikbaar.

J.2 Wat biedt de HCC Forth-gg

Een printenset voor de robot en opsteek printen voor verschillende microcontrollers inclusief een bouwbeschrijving en onderdelenlijsten. De HCC Forth-gg biedt ook ondersteuning via het internet en op bijeenkomsten van de club.

J.3 Plaatje van Ushi



Figuur J.1: Tekening van de Ushi hoofdprint

K Interessante AVR adressen

Op het internet zijn er veel interessante websites te vinden, die (deels) gewijd zijn aan de door AVR ByteForth gebruikte microcontroller serie. Vindt je hier niet wat je zoekt probeer dan de AVR-webring, de meeste van de onderstaande sites zijn hierbij aangesloten.

http://www.avrfreaks.net	Een zeer uitgebreide website geheel gewijd aan de AVR microcontroller. Opgedeeld in pagina's over hardware, chip overzicht, software, application notes, Academy (cursussen), design notes (ideen), diverse forums, etc.
http://jump.to/~fleury/	Heeft een leuke pagina over een doe-hetzelf starterkit, het idee om de 74HC125 toe te passen voor de ISP-dongle komt van deze site. Voor de starterkit gebruikt hij een verend breadboard.
http://shop.kanda.com/shopnav/shop.php3	De ontwikkelaars van de bekende STK200(+) & STK300 starterkits. De STK200+ is nog in de handel en kost $ 65,- inclusief CD-ROM, ISP-dongle en een AVR microcontroller.
http://www.atmel.com/products/avr/	De maker van de AVR- en AT89-microcontrollers. Een degelijke website waar al de datasheets van de diverse ATMEL microcontrollers te vinden zijn. Ook hebben ze voor elke controllertype een FAQ en veel application notes (voorbeelden) vaak met code.
http://www.dontronics.com/atmel.html	Ontwikkelaar van de zogenaamde Simmsticks, de Simmsticks zijn compacte microcontrollerprinten met gestandaardiseerde aansluitingen. Een zeer uitgebreide site met tal van links naar weer andere interessante AVR-sites.
http://www.hth.com/loaa/	Initiatief van Christer Johansson, houdt een lijst bij van openbaar beschikbaar gestelde (AVR) code(voorbeelden).
http://www.olimex.com/dev/	Site van printfabrikant Olimex, verkopen prototype-printen voor 8, 20, 28 en 40-polige DIL AVR chips. Daarnaast hebben ze een Kanda compatible dongle (alleen een erg kort snoertje). Verder nog een tweetal ontwikkelprintjes voor de AT90S2313 eentje met relais er op, RS232, optokoplers, etc. de ander met LCD, RS232, toetsenbord, buzzer, etc.

L AVR specials

L.1 I/O-poort structuur

De AVR heeft een andere I/O-poort structuur dan de 8051, per poort zijn er drie adressen genaamd PINx, DDRx en PORTx. De letter 'x' staat voor de poortnaam. Om I/O-poorten simpel te kunnen gebruiken is een idee uit een Java-compiler voor de AVR overgenomen. Er zijn twee prefixen gedefinieerd die het PORTx adres gebruiken om PINx (FROM) en DDRx (SETDIR) te kunnen benaderen. Let op er PINx kan op deze manier alleen gelezen worden en DDRx alleen geschreven! Als we meer willen dan moet voor elk een apart SFR gedefinieerd worden. We hebben op PORTB vier leds en vier schakelaars zitten, een voorbeeld:

```
PORTB SFR I/O           \ Definieer hi-level toegang tot PORTB
$0F SETDIR I/O          \ Initialiseer DDRB (4-in/4-uit)
$08 TO I/O              \ Zet $08 op PORTB
FROM I/O                \ Lees PINB uit
```

L.2 Over de registers

De AVR-microcontroller heeft 32 registers aan boord. Van deze 32 zijn er maximaal 14 beschikbaar voor de programmeur. Registers werken hetzelfde als VALUE's en zijn vooral goed bruikbaar op plaatsen waar snelheid belangrijk is. Ga er daarom zuinig mee om. Met het woord REGISTER wordt een registervariabele gedefiniëerd. Je gebruikt hem als volgt:

	Alle hieronder getoonde commando's genereren slechts een AVR-opcode en kosten een kristal-tik.
`REGISTER TELLER`	Definiëer een registervariabele met de naam TELLER.
`CLEAR TELLER`	Zet teller op nul.
`INCR TELLER`	Verhoog teller met een.
`PUSH TELLER`	Zet teller op de returnstack.
etc.	

Er bestaat ook nog het woord (REGISTER) met dit woord kun je elk van de 32-registers een naam geven. Het is opgenomen om het uiterste uit de AVR te kunnen persen. Je gebruikt het als volgt:

`25 (REGISTER) AAP`	Geef register 25 de naam AAP.
`CLEAR AAP`	Zet register 25 op nul.
etc.	Register 16 t/m 25 zijn gereserveerd als werkgeheugen voor de AVRF-machinecode. Er zijn echter slechts enkele macro's die de hoogste registers gebruiken. Als je weet dat die niet toegepast worden kun je ze met (REGISTER) toch gebruiken.

L.3 Configureerbare hardware

Een aantal bijzonderheden van de AVR's worden ingesteld d.m.v 'fuse-bits'. Zo kunnen b.v. de klok-oscilator en/of de brownout detector hiermee geconfigureerd worden. Deze configuratie verschilt van de andere AVR-hardware als SPI en ADC dat de instelling gebeurt tijdens het

programmeren van de AVR en niet door een van de I/O-registers te beschrijven. Als je van gedachten veranderd moet je de fuse-bits van de AVR opnieuw programmeren.
Het programmeermechanisme waarmee de hardware (fuses) aangepast wordt beschreven op bladzijde 89. Een AVR heeft wel twee tot 18 of meer fuse-bits. Fuse-bits worden niet gewist door een wiscommando omdat ze in een ander geheugengebied staan dan het programmageheugen.

L.3.1 Fuse byte voorbeelden

De ATmega8 heeft twee fuse bytes, beide fuse bytes zijn hier uitgewerkt als voorbeeld. Een fuse-byte bevat maximaal 8 fuse-bits, een fuse-bit is één als hij niet geprogrameerd is en nul als hij geprogrameerd is. De laatste kolom in de tabel toont de staat van een fuse-bit zoals de fabrikant hem aflevert.

Fuse hoge byte	**Bit nr.**	**Beschrijving**	**Default toestand**
RSTDISBL	7	PC6 is I/O-pen of resetpen	1 = PC6 is RESET-pen
WDTON	6	Watchdog altijd aan	1 = WDT aan door WDTCR
SPIEN	5	SPI interface aan/uit	0 = SPI interface aan
CKOPT	4	Oscillator uitgang	1 = Low power oscillator
EESAVE	3	EEPROM niet wissen met E-commando	1 = EEPROM ook wissen
BOOTSZ1	2	Kies grootte van het BOOTblok	0 = Zie aparte tabel
BOOTSZ0	1	Kies grootte van het BOOTblok	0 = Zie aparte tabel
BOOTRST	0	Kies de plaats v/d RESET-vector	1 = Flash adres 0

Fuse lage byte	**Bit nr.**	**Beschrijving**	**Default toestand**
BODLEVEL	7	Brown out detector trigger niveau	1 = 2,7 Volt of 4 Volt (0)
BODEN	6	Brown out aan/uit	1 = Brown out uit
SUT1	5	Kies start-up tijd	1 = Stel start-up methode in
SUT0	4	Kies start-up tijd	1 = Stel start-up methode in
CKSEL3	3	Kies klok type	0 = Stel AVR klok in
CKSEL2	2	Kies klok type	0 = Stel AVR klok in
CKSEL1	1	Kies klok type	0 = Stel AVR klok in
CKSEL0	0	Kies klok type	1 = Stel AVR klok in

L.3.2 Mogelijkheden

Wat kan er op AVR's via deze 'fuses' zoal ingesteld worden?

1) Er kan een opstart (boot) blok ingesteld worden inclusief de afmeting ervan.
2) De AVR kan toegestaan worden om zichzelf te herprogrammeren.
3) De klokoscillator kan gekozen worden, intern RC, extern kristal, etc. Wordt een interne RC-osc. gekozen, dan kunnen de kristalaansluitingen meestal als extra I/O gebruikt worden.
4) Er kan een extra klokdeler geactiveerd worden.
5) De opstarttijd (resetduur) kan ingesteld worden.
6) De brownout detector kan ge(de)activeerd worden en afgeregeld.
7) Het EEPROM kan beveiligd worden tegen ISP-wisacties.
8) De watchdog-timer kan aan/uit geschakeld worden.
9) De ISP-programmer kan gedeactiveerd worden.
10) Een ééndraads debugWIRE interface kan aan/uit gezet worden.

11) De RESET-pen kan tot I/O-pen omgebouwd worden.
12) De soms aanwezige JTAG-interface kan ook aan of uit gezet worden, dit levert dan weer wat extra I/O-pennen op.

L.3.3 Let op de fuses!

Een veel voorkomende fout is de fuses geheel te vergeten. Lees goed wat de default instelling van je AVR is, anders kun je rare fouten krijgen. Als je geluk hebt staat de AVR in de door jou gewenste toestand, als dat niet zo is zul je een of meerdere fuse-bits moeten wijzigen.
Zo staat bij de nieuwe AVR's de interne oscilator aan. Als je een extern kristal aan sluit en er van uit gaat dat de AVR op die frequentie gaat werken dan heb je het goed mis. De AVR blijft rustig op zijn interne RC-oscillator werken tot je de fuse-bits goed ingesteld hebt.
Een UART zal dan op de verkeerde frequentie werken zodat de seriële verbinding niet werkt. De AVR kan ook een de compatibiliteits mode staan om een ouder type te ondersteunen of een I/O-poort werkt niet omdat de JTAG-interface die op de nieuwe chips aangebracht is default aan staat. Dus controleer goed in welke toestand de AVR staat en hoe je hem in de door jou gewenste toestand krijgt.

L.4 Extra I/O-poort functies

Bijna elk I/O-bit op AVR's heeft meer dan een functie, soms wel zes verschillende mogelijkheden. Als voorbeeld worden de I/O-poorten van de AT90S2313 getoond. De I/O-bits op deze AVR hebben nooit meer dan twee verschillende functies, zijn opvolger de ATtiny2313 heeft maximaal vier functies op een I/O-bit. Nemen we de nieuwe ATtiny13 daar zitten op I/O-bit PB0 maar liefst zes verschillende functies.

L.4.1 Functies van poort-B

Port Pin	Alternate Functions
PB0	AIN0 (Analog comparator positive input)
PB1	AIN1 (Analog comparator negative input)
PB3	OC1 (Timer/Counter1 Output compare match output)
PB5	MOSI (Data input line for memory downloading)
PB6	MISO (Data output line for memory uploading)
PB7	SCK (Serial clock input)

When the pins are used for the alternate function the DDRB and PORTB register has to be set according to the alternate function description.

Figuur L.1: De speciale funkties van Poort-B op de AT90S2313

L.4.2 Functies van poort-D

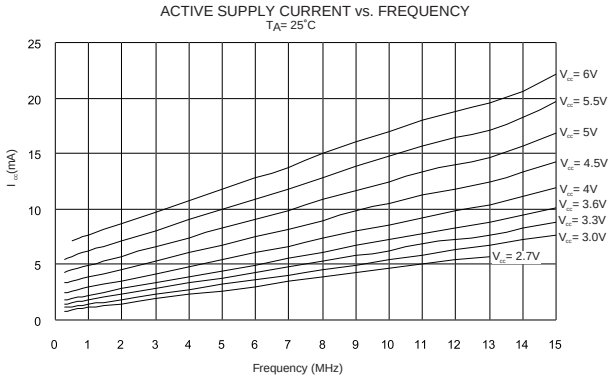
Port Pin	Alternate Function
PD0	RXD (Receive data input for the UART)
PD1	TXD (Transmit data output for the UART)
PD2	INT0 (External interrupt 0 input)
PD3	INT1 (External interrupt 1 input)
PD4	TO (Timer/Counter0 external input)
PD5	T1 (Timer/Counter1 external input)
PD6	ICP (Timer/Counter1Input Capture pin)

When the pins are used for the alternate function the DDRD and PORTD register has to be set according to the alternate function description.

Figuur L.2: De speciale funkties van Poort-D op de AT90S2313

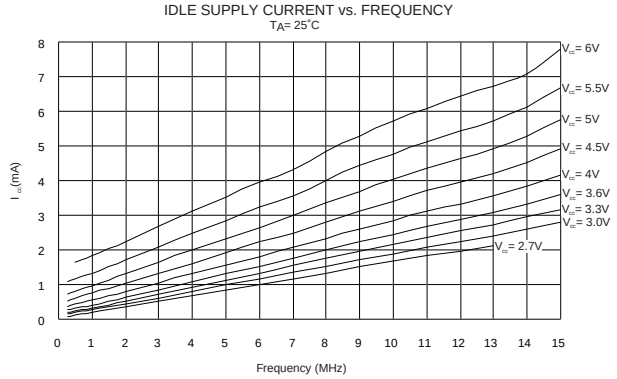
M Stroomverbruik van AT90S2313

M.1 Normaal stroomverbruik



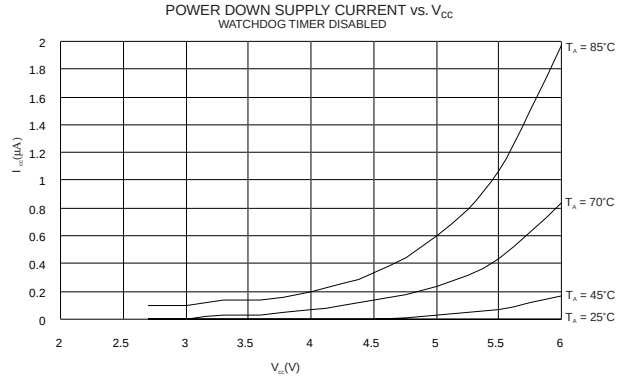
Figuur M.1: AT90S2313 stroomverbruik in actieve toestand

M.2 Idle stroomverbruik



Figuur M.2: AT90S2313 stroomverbruik in idle toestand

M.3 Powerdown stroomverbruik



Figuur M.3: AT90S2313 stroomverbruik in powerdown toestand

Features

- Utilizes the AVR® RISC Architecture
- AVR - High-performance and Low-power RISC Architecture
  - 118 Powerful Instructions - Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Up to 10 MIPS Throughput at 10 MHz
- Data and Nonvolatile Program Memory
  - 2K Bytes of In-System Programmable Flash
    Endurance 1,000 Write/Erase Cycles
  - 128 Bytes of SRAM
  - 128 Bytes of In-System Programmable EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - Programming Lock for Flash Program and EEPROM Data Security
- Peripheral Features
  - One 8-bit Timer/Counter with Separate Prescaler
  - One 16-bit Timer/Counter with Separate Prescaler,
    Compare, Capture Modes and 8-, 9- or 10-bit PWM
  - On-chip Analog Comparator
  - Programmable Watchdog Timer with On-chip Oscillator
  - SPI Serial Interface for In-System Programming
  - Full Duplex UART
- • Special Microcontroller Features
  - Low-power Idle and Power Down Modes
  - External and Internal Interrupt Sources
- • Specifications
  - Low-power, High-speed CMOS Process Technology
  - Fully Static Operation
- Power Consumption at 4 MHz, 3V, 25°C
  - Active: 2.8 mA
  - Idle Mode: 0.8 mA
  - Power Down Mode: <1 µA
- I/O and Packages
  - 15 Programmable I/O Lines
  - 20-pin PDIP and SOIC
- Operating Voltages
  - 2.7 - 6.0V (AT90S2313-4)
  - 4.0 - 6.0V (AT90S2313-10)
- Speed Grades
  - 0 - 4 MHz (AT90S2313-4)
  - 0 - 10 MHz (AT90S2313-10)

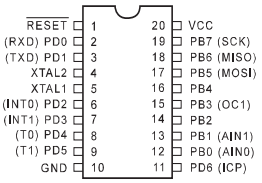
Description

The AT90S2313 is a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture. By executing powerful instructions in a single clock cycle, the

(continued)

Pin Configuration

PDIP/SOIC



8-bit AVR® Microcontroller with 2K bytes In-System Programmable Flash

AT90S2313

Rev. 0839ES–04/99

Note: This is a summary document. For the complete 87 page document, please visit our web site at *www.atmel.com* or e-mail at *literature@atmel.com* and request literature #0839E.

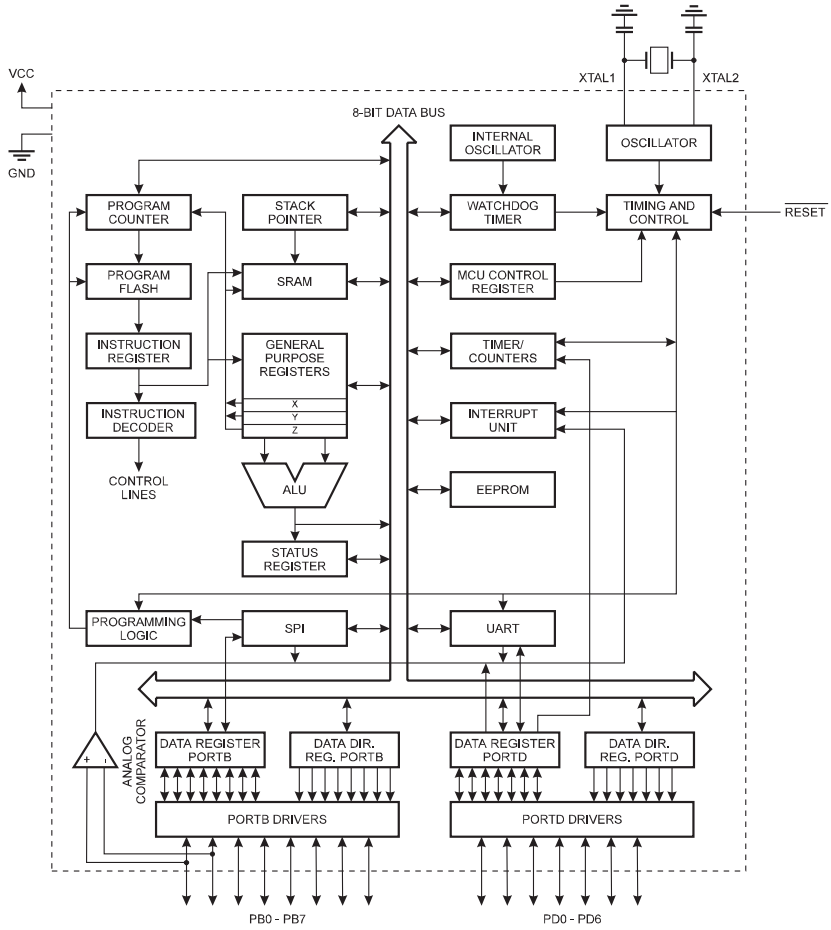
N.1 Datasheet AT90S2313 blokdiagram

AT90S2313 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

Block Diagram

Figure 1. The AT90S2313 Block Diagram



N.2 Datasheet AT90S2313 pen beschrijving

The AT90S2313 provides the following features: 2K bytes of In-System Programmable Flash, 128 bytes EEPROM, 128 bytes SRAM, 15 general purpose I/O lines, 32 general purpose working registers, flexible timer/counters with compare modes, internal and external interrupts, a programmable serial UART, programmable Watchdog Timer with internal oscillator, an SPI serial port for Flash Memory downloading and two software selectable power saving modes. The Idle Mode stops the CPU while allowing the SRAM, timer/counters, SPI port and interrupt system to continue functioning. The power down mode saves the register contents but freezes the oscillator, disabling all other chip functions until the next external interrupt or hardware reset.

The device is manufactured using Atmel's high density nonvolatile memory technology. The on-chip In-System Programmable Flash allows the program memory to be reprogrammed in-system through an SPI serial interface or by a conventional nonvolatile memory programmer. By combining an enhanced RISC 8-bit CPU with In-System Programmable Flash on a monolithic chip, the Atmel AT90S2313 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The AT90S2313 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

Pin Descriptions

VCC

Supply voltage pin.

GND

Ground pin.

Port B (PB7..PB0)

Port B is an 8-bit bi-directional I/O port. Port pins can provide internal pull-up resistors (selected for each bit). PB0 and PB1 also serve as the positive input (AIN0) and the negative input (AIN1), respectively, of the on-chip analog comparator. The Port B output buffers can sink 20mA and can drive LED displays directly. When pins PB0 to PB7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not active.

Port D (PD6..PD0)

Port D has seven bi-directional I/O port with internal pull-up resistors, PD6..PD0. The Port D output buffers can sink 20 mA. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not active.

$\overline{\text{RESET}}$

Reset input. A low level on this pin for more than 50 ns will generate a reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

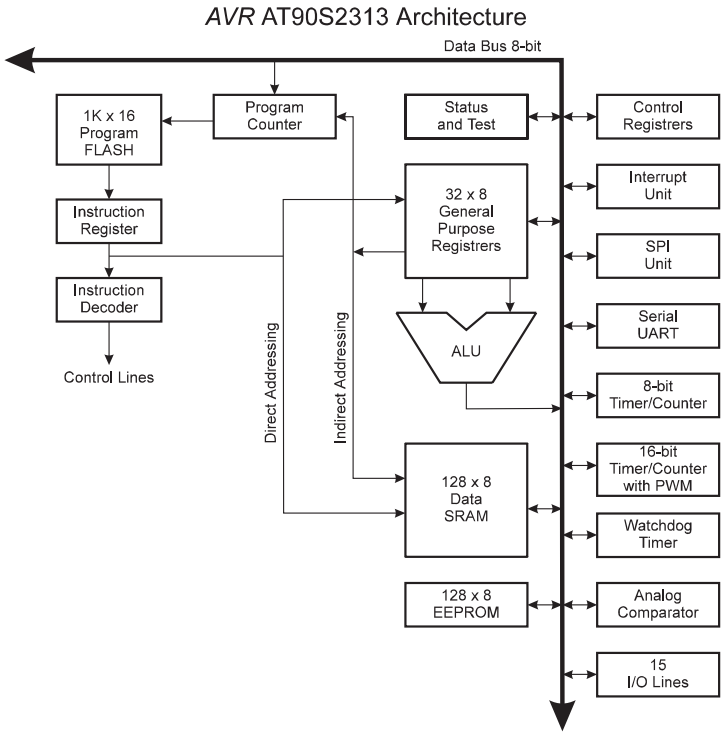
XTAL2

Output from the inverting oscillator amplifier

Architectural Overview

The fast-access register file concept contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This means that during one single clock cycle, one ALU (Arithmetic Logic Unit) operation is executed. Two operands are output from the register file, the operation is executed, and the result is stored back in the register file - in one clock cycle.

Figure 2. The AT90S2313 AVR RISC Architecture



Six of the 32 registers can be used as three 16-bits indirect address register pointers for Data Space addressing - enabling efficient address calculations. One of the three address pointers is also used as the address pointer for the constant table look up function. These added function registers are the 16-bits X-register, Y-register and Z-register.

The ALU supports arithmetic and logic functions between registers or between a constant and a register. Single register operations are also executed in the ALU. Figure 2 shows the AT90S2313 AVR RISC microcontroller architecture.

In addition to the register operation, the conventional memory addressing modes can be used on the register file as well. This is enabled by the fact that the register file is assigned the 32 lowermost Data Space addresses ($00 - $1F), allowing them to be accessed as though they were ordinary memory locations.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, Timer/Counters, A/D-converters, and other I/O functions. The I/O memory can be accessed directly, or as the Data Space locations following those of the register file, $20 - $5F.

The AVR has Harvard architecture - with separate memories and buses for program and data. The program memory is accessed with a two stage pipeline. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-system Programmable Flash memory.

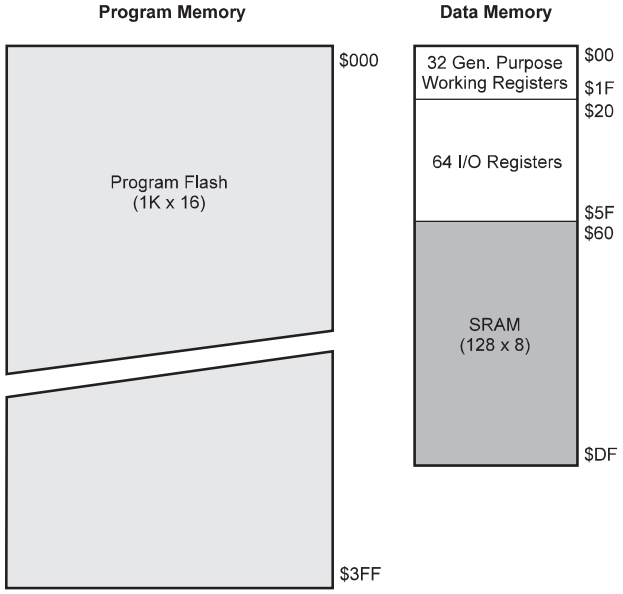
With the relative jump and call instructions, the whole 1K address space is directly accessed. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction.

During interrupts and subroutine calls, the return address program counter (PC) is stored on the stack. The stack is effectively allocated in the general data SRAM, and consequently the stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the reset routine (before subroutines or interrupts are executed). The 8-bit stack pointer SP is read/write accessible in the I/O space.

The 128 bytes data SRAM + register file and I/O registers can be easily accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

Figure 3. Memory Mapss



A flexible interrupt module has its control registers in the I/O space with an additional global interrupt enable bit in the status register. All the different interrupts have a separate interrupt vector in the interrupt vector table at the beginning of the program memory. The different interrupts have priority in accordance with their interrupt vector position. The lower the interrupt vector address the higher the priority.

N.5 Datasheet AT90S2313 SFR registers

Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	19
$3E ($5E)	Reserved									
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	20
$3C ($5C)	Reserved									
$3B ($5B)	GIMSK	INT1	INT0	-	-	-	-	-	-	25
$3A ($5A)	GIFR	INTF1	INTF0							26
$39 ($59)	TIMSK	TOIE1	OCIE1A	-	-	TICIE1	-	TOIE0	-	26
$38 ($58)	TIFR	TOV1	OCF1A	-	-	ICF1	-	TOV0	-	27
$37 ($57)	Reserved									
$36 ($56)	Reserved									
$35 ($55)	MCUCR	-	-	SE	SM	ISC11	ISC10	ISC01	ISC00	28
$34 ($54)	Reserved									
$33 ($53)	TCCR0	-	-	-	-	-	CS02	CS01	CS00	31
$32 ($52)	TCNT0	Timer/Counter0 (8 Bit)								31
$31 ($51)	Reserved									
$30 ($50)	Reserved									
$2F ($4F)	TCCR1A	COM1A1	COM1A0	-	-	-	-	PWM11	PWM10	33
$2E ($4E)	TCCR1B	ICNC1	ICES1	.	-	CTC1	CS12	CS11	CS10	34
$2D ($4D)	TCNT1H	Timer/Counter1 - Counter Register High Byte								35
$2C ($4C)	TCNT1L	Timer/Counter1 - Counter Register Low Byte								35
$2B ($4B)	OCR1AH	Timer/Counter1 - Compare Register High Byte								36
$2A ($4A)	OCR1AL	Timer/Counter1 - Compare Register Low Byte								36
$29 ($49)	Reserved									
$28 ($48)	Reserved									
$27 ($47)	Reserved									
$26 ($46)	Reserved									
$25 ($45)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								36
$24 ($44)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								36
$23 ($43)	Reserved									
$22 ($42)	Reserved									
$21 ($41)	WDTCR	-	-	-	WDTOE	WDE	WDP2	WDP1	WDP0	38
$20 ($40)	Reserved									
$1F ($3F)	Reserved									
$1E ($3E)	EEAR	-	EEPROM Address Register							40
$1D ($3D)	EEDR	EEPROM Data register								40
$1C ($3C)	EECR	-	-	-	-	-	EEMWE	EEWE	EERE	40
$1B ($3B)	Reserved									
$1A ($3A)	Reserved									
$19 ($39)	Reserved									
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	50
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	50
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	50
$15 ($35)	Reserved									
$14 ($34)	Reserved									
$13 ($33)	Reserved									
$12 ($32)	PORTD	-	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	55
$11 ($31)	DDRD	-	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	55
$10 ($30)	PIND	-	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	55
...	Reserved									
$0C ($2C)	UDR	UART I/O Data Register								44
$0B ($2B)	USR	RXC	TXC	UDRE	FE	OR	-	-	-	45
$0A ($2A)	UCR	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	45
$09 ($29)	UBRR	UART Baud Rate Register								47
$08 ($28)	ACSR	ACD	-	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	48
...	Reserved									
$00 ($20)	Reserved									

Notes: 1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.

2. Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers $00 to $1F only.

Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd – Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd – K	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl – K	Z,C,N,V,S	2
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd – Rr – C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd – K – C	Z,C,N,V,H	1
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← $FF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← $00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FF – K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← $FF	None	1
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1 / 2
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1 / 2
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1 / 2
SBIS	P, b	Skip if Bit in I/O Register is Set	if (R(b)=1) PC ← PC + 2 or 3	None	1 / 2
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC + k + 1	None	1 / 2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC + k + 1	None	1 / 2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2

N.7 Datasheet AT90S2313 opcodes-2

Instruction Set Summary (Continued)

Mnemonics	Operands	Description	Operation	Flags	#Clocks
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X – 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y – 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	3
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1

N.8 Datasheet AT90S2313 bestel informatie

Ordering Information

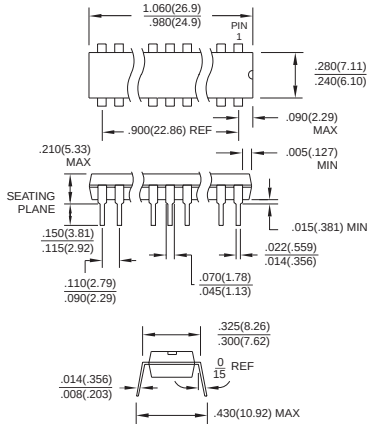
Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
4	2.7 - 6.0V	AT90S2313-4PC AT90S2313-4SC	20P3 20S	Commercial (0°C to 70°C)
		AT90S2313-4PI AT90S2313-4SI	20P3 20S	Industrial (-40°C to 85°C)
10	4.0 - 6.0V	AT90S2313-10PC AT90S2313-10SC	20P3 20S	Commercial (0°C to 70°C)
		AT90S2313-10PI AT90S2313-10SI	20P3 20S	Industrial (-40°C to 85°C)

Package Type	
20P3	20-lead, 0.300" Wide, Plastic Dual In-Line Package (PDIP)
20S	20-lead, 0.300" Wide, Plastic Gull-Wing Small Outline (SOIC)

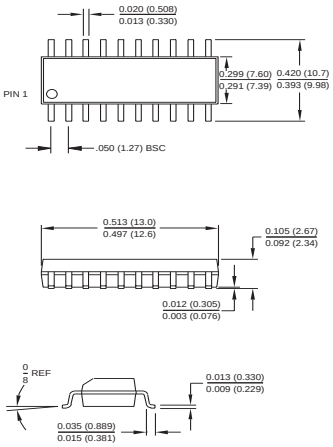
N.9 Datasheet AT90S2313 behuizingen

Packaging Information

20P3, 20-lead, 0.300" Wide,
Plastic Dual Inline Package (PDIP)
Dimensions in Inches and (Millimeters)
JEDEC STANDARD MS-001 BA



20S, 20-lead, 0.300" Wide,
Plastic Gull-Wing Small Outline (SOIC)
Dimensions in Inches and (Millimeters)



N.10 Datasheet ATMEL adressen

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel U.K., Ltd.
Coliseum Business Centre
Riverside Way
Camberley, Surrey GU15 3YL
England
TEL (44) 1276-686-677
FAX (44) 1276-686-697

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs
1150 E. Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

0839ES–04/99/xM

Atmel wordt in Nederland vertegenwoordigd door ALCOM electronics bv in Capelle aan den IJssel. Tel: 010 - 2882500, fax: 010 - 2882525. Internet: http://www.alcom.nl

O Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z