AVR ByteForth

versie 2.00

© Willem Ouwerkerk

27 oktober 2003

Inhoudsopgave

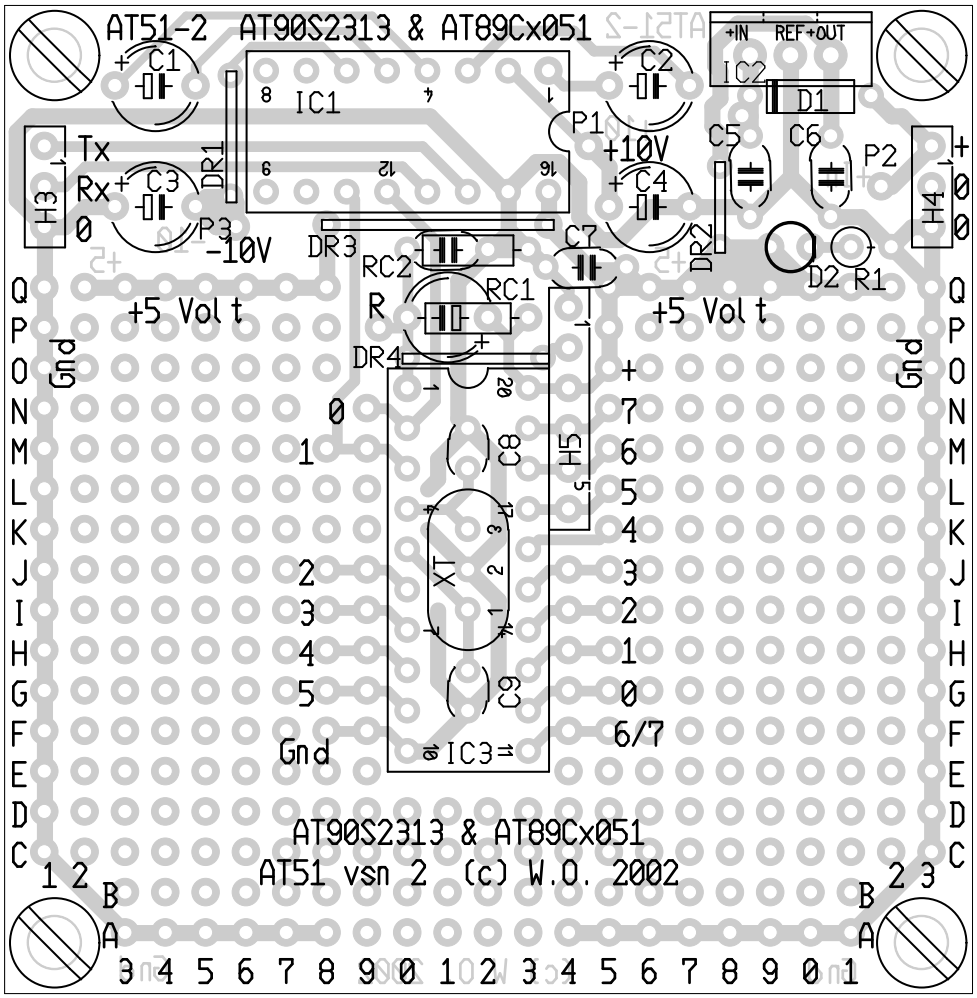
1 Installatie

1.1 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 gelijkspannings voeding 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 17, 18 en 20.



Figuur 1.1: Tekening van de AT51-2 print

1.2 Software

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

- Optimaliserende crosscompiler die code in een buffer genereerd.
- 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.

1.3 Installeren van de software

1) Software van de AVR ByteForth omgeving kan geinstalleerd 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 2.00 (c) W.O. 2003
ISP Flashprogrammer versie 1.31 (c) W.O. 2000-2003
etc.
```

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

1.4 Configureren van de software

AVR ByteForth en de programmer zijn nu geinstalleerd 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 u kunt natuurlijk uw eigen favoriete DOS-editor daarvoor in de plaats zetten. De file is opgesplitst in vijf delen:

1) De paden naar de bibliotheek- en hulpfiles.
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 2.00 Beta versie

\ 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 AT90Sxxxx" 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
```

1.5 ByteForth functietoetsen

De actieve toetscombinaties en functietoetsen van ByteForth zijn:

- F1 Hulpfile bij ByteForth commandline editor.
- F2 Online ByteForth hulp 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 af, ga terug naar DOS/Windows.

1.6 Adres van de HCC Forth gg

Voor vragen en verdere informatie over AVR of 8051 ByteForth zend 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:	voorz@forth-gg.hobby.nl
Homepage:	http://www.forth.hccnet.nl

2 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 tekens 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 je de draad kwijt bent, doe dat stuk dan weer 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.

2.1 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 1 + 2 uit te rekenen moest je intoetsen 1 <Enter> 2 <+>. Het antwoord verscheen dan op het display. Je plaatst eerst het getal 1 op de stack (ned. stapel), dan het tweede (2) en vervolgens geef je aan welke bewerking op die twee getallen uitgevoerd moet worden. Dit wordt ook wel de Reverse Polish Notation (RPN) genoemd. In Forth gaat dat op een vergelijkbare manier.

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>	.S is een niet destructieve stack print opdracht.
+ . <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 veel woorden om de stack te manipuleren.
	Forth werkt over het algemeen zo: eerst de
	data, dan de operator (actie).

2.2 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, & = ASCII en % = 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 eerst voer je 10*12 uit en trekt 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): -128 ... +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.

2.3 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.

`tel <cr>`	Oeps foutmelding. Forth is een verzameling woorden. En `tel` is geen Forth woord.
`: tel 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 tel . <cr>`	Al het andere tot de punt-comma is de code die uitgevoerd zal worden als het nieuwe woord wordt uitgevoerd.

`: tellus <cr>`
`  100 <cr>`
`  5 0 do tel 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 `tel` er in gecompileerd. Daarna heb je `tellus` uitgevoerd. In ByteForth kun je bijna alles op deze manier uitproberen. Later meer daarover.

2.4 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 beperkt het gebruik van RAM enorm. 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 tegelijkertijd afgewerkt worden (multi-tasking). 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>`	Zet de inhoud van `pils` op de stack, 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.

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

Ook lokale variabelen zijn binnen colon-definities toegestaan, meer daarover in op bladzijde 11.
Gebruik `HELP 'naam' <cr>` om meer uitleg over een Forth woord of begrip te krijgen.

2.5 Controlestructuren

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

```
atom inline$
```
Om `."` te gebruiken moet je de macro `inline$` importeren. Het woord `atom` verzorgd dat importeren.

```
: .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 lus `begin until` controlestructuur met daarin een (geneste) `if`.

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

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

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

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

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

```
      -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`.

2.6 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. Hier een assembler voorbeeld, die je niet gelijk helemaal hoeft te begrijpen.

```
code 6+ ( x1 -- x2 ) <cr>
  r16 x+ ld, <cr>
  r16 6 addi, <cr>
  -x r16 st, <cr>
  ret, <cr>
end-code <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. `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>` `vul . <cr>`	Zie je.

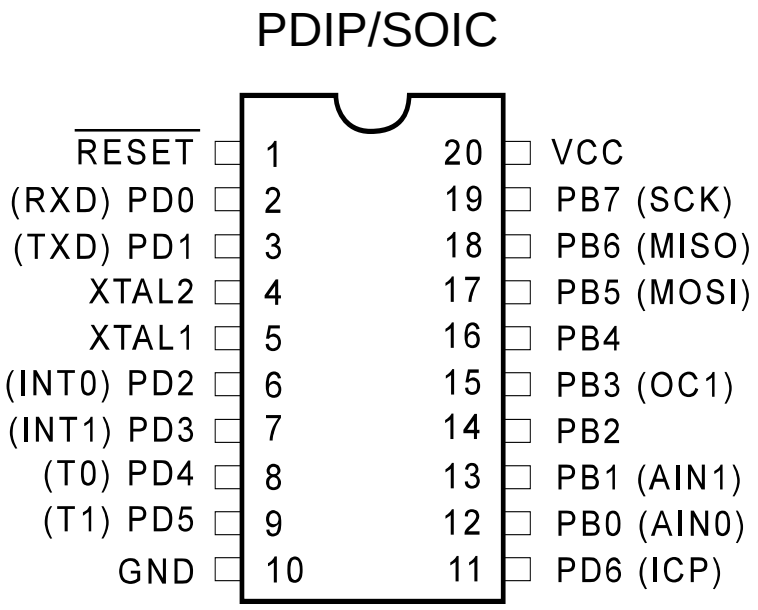
2.7 Special Function Registers (SFR's)

Het definierend 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.

`$18 sfr poortb <cr>`	Definieer 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>`	Definieer 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.

2.8 Penconfiguratie AT90S2313

Voor een volledig datasheet van chips uit de AVR serie, moet je naar de website van ATMEL gaan, de link daarvan vindt je op bladzijde 20.



Figuur 2.1: Pen configuratie van AT90S2313

2.9 Het is een crosscompiler

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

`words <cr>`	Laat alle woorden in het werkgebied zien.
`( zie je uitgang & ingang )`	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 met naam 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?

2.10 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.

2.11 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. Daar merk je nauwelijks iets van. De simulator kun je ook gebruiken als tracer om bugs te vinden in je code.

`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.

2.12 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 net als een variabele 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 !!

2.13 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.

2.14 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.

2.15 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 de gegenereerde code niet helemaal te begrijpen hoor!

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.

2.16 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 toe te voegen.

```
ram <cr>
:: vars <cr>
  create <cr>

    allot align <cr>

  does> d+ ; <cr>

10. vars array <cr>
12 9. array ! <cr>
100 0. array ! <cr>
0. array @ . <cr>
9. array @ . <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 10 in de array,

Zet 100 op positie 1.

Lees positie 1 terug, klopt het?

Lees ook positie 10 terug, klopt die ook?

```
rom <cr>                          Nu een datstructuur in ROM.
:: exec <cr>                      Je maakt een executietabel genaamd EXEC. Op de
  create ( xn x1 n -- ) <cr>      stack verwacht die 'n' executie-tokens in omge-
    dup 1- , align <cr>           keerde volgorde van 'x1' t/m 'xn'.
    0 do  d,  loop <cr>           Het aantal tokens wordt bewaard en de tokens wor-
                                  den in ROM opgeslagen.
  does>  ( n -- i*x ) <cr>        Bij uitvoering wordt op de stack het nummer van
                                  het gewenste token 'n' verwacht.
    2>r  2r@ rom@ <cr>            Na manipulatie en een berekening wordt
    umin 1+ 2* 0 <cr>             het juiste token uit de tabel opgevist door
    2r> d+ <cr>                   2ROM@ en vervolgens uitgevoerd door EXECUTE.
    2rom@ execute ; <cr>          Als 'n' ongeldig is wordt altijd het laatste token
                                  'xn' uitgevoerd.
: nul   0 ; <cr>                  Vier programma's voor in de executietabel.
: een   -1 ; <cr>
: twee  -2 ; <cr>
: drie  -3 ; <cr>

' drie  ' twee <cr>               Zet de vier tokens van de programma's klaar.
' een   ' nul <cr>
4 exec wim <cr>                   Maak een executietabel met vier elementen en de
                                  naam wim.
0 wim . <cr>                      Het eerste token wordt uitgevoerd.
3 wim . <cr>                      Het vierde token wordt uitgevoerd.
5 wim . <cr>                      Omdat 5 geen geldig token opleverd, wordt het
                                  vierde token uitgevoerd. Gesnapt? Maak je geen
                                  zorgen als dat nog niet zo is, CREATE DOES> is
                                  Forth voor gevorderden.
```

2.17 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
```

Definieer 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` gebruikt 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. Speciaal voor toepassing daarvan zijn enkele speciale commando's opgenomen. Zie daarvoor de files HILEVEL1.FRT en HILEVEL2.FRT als voorbeeld in de EXAMPLES directory.

A Wat is er nieuw t.o.v. 8051 ByteForth

AVR ByteForth 2.00 is geheel nieuw en 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 90S2313 t/m de ATmega64.
- VALUE een 8 bits T0-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 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.
- Tenslotte is dit handboek nog verder uitgebreid en verbeterd.

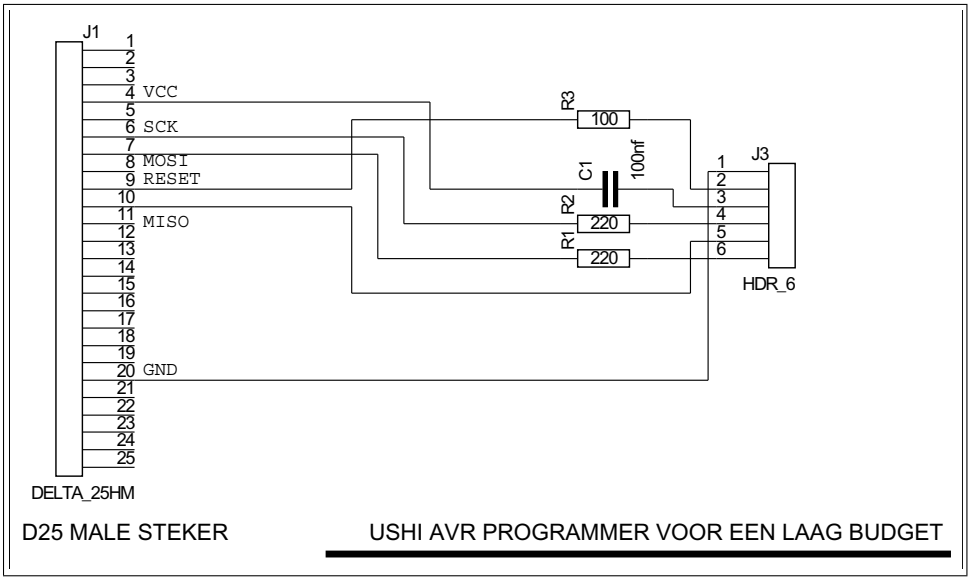
B El Cheapo dongle schema

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 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 El Cheapo



Figuur B.1: Schema El Cheapo

B.3 Bouwbeschrijving 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.

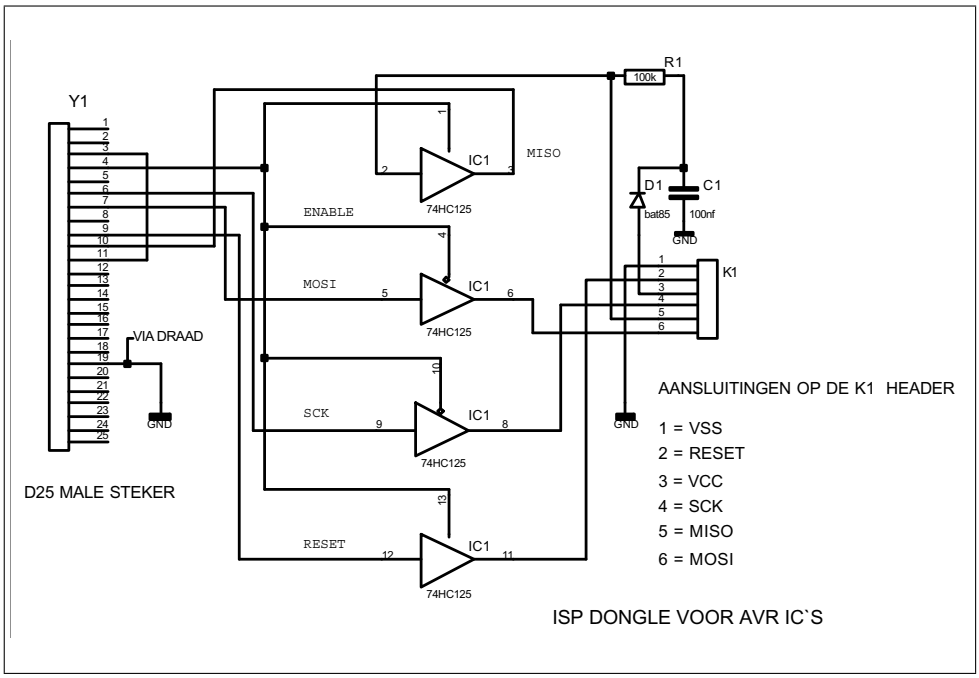
C AVR ByteForth ISP-dongle

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 dongle

C1	100nF	Keramische C, steek 5mm
D1	1N4148	Diode
DR1	DRAAD	draadbrug
DR2	DRAAD	draadbrug
DR3	DRAAD	draadbrug
DR4	DRAAD	draadbrug
DR5	DRAAD	draadbrug
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 Dongle schema

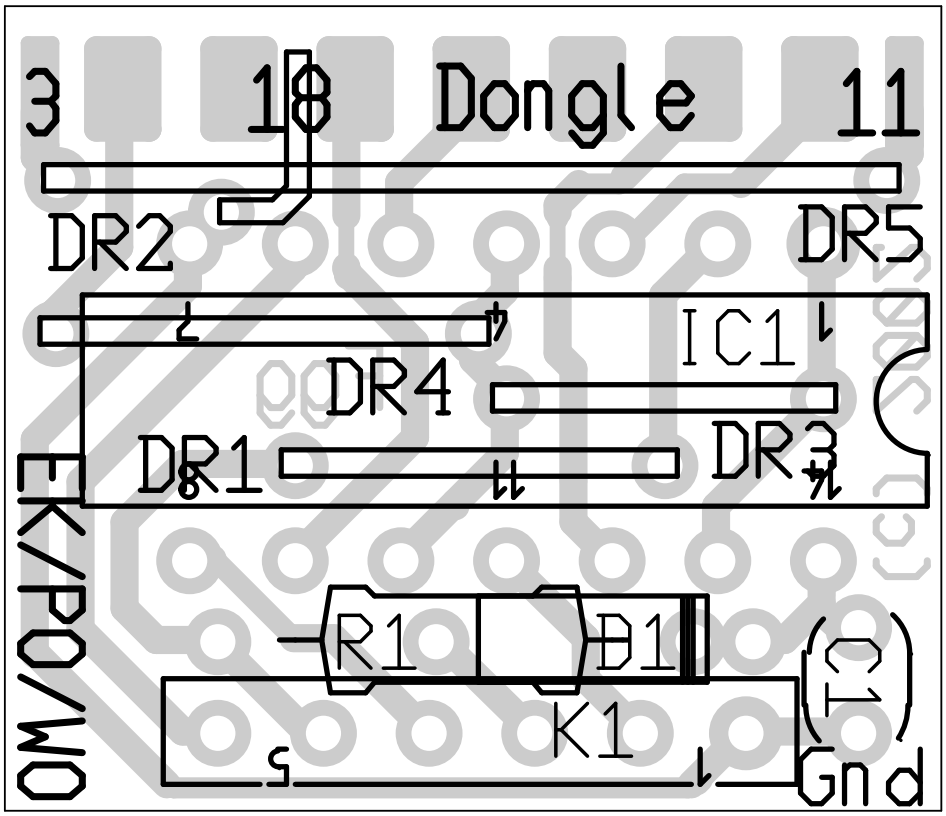


Figuur C.1: Schema dongle

C.3 Bouwbeschrijving dongle

1) Soldeer alle draadbruggen op hun plaats behalve DR2.
2) Soldeer nu de eerst de diode D1, dan weerstand R1 daarbovenop en daarna condensator C1.
3) Plaats IC1 direct (zonder IC-voet) op de print en soldeer hem vast.
4) Stop de print op de goede plaats tussen de DB25 soldeerpennen, de buitenste pennen zijn 3 en 11. Soldeer de print aan de pennen.
5) Plaats en soldeer draad DR2 en de bandkabel K1 op de print.
6) 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.
7) Plaats de trekontlasting op de kabel en zet de kap H1 op zijn plaats.

C.4 Componenten plaatsing dongle



Figuur C.2: Printbezetting dongle

D 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://luna.spaceports.com/~pfleury/	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.