Vierte Dimension

4/2006

Das Forth–Magazin

für Wissenschaft und Technik, für kommerzielle EDV,
für MSR-Technik, für den interessierten Hobbyisten

In dieser Ausgabe:

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische Informatik

Feldstrasse 143
D–22880 Wedel
Fon 04103 – 808989 – 0
Fax 04103 – 808989 – 9
mail@tematik.de
www.tematik.de

Gegründet 1985 als Partnerinstitut der FH–Wedel beschäftigten wir uns in den letzten Jahren vorwiegend mit Industrieelektronik und Präzisionsmeßtechnik und bauen z. Z. eine eigene Produktpalette auf.

Know–How Schwerpunkte liegen in den Bereichen Industriewaagen SWA & SWW, Differential–Dosierwaagen, DMS–Messverstärker, 68000 und 68HC11 Prozessoren, Sigma–Delta A/D. Wir programmieren in Pascal, C und Forth auf SwiftX86k und seit kurzem mit Holon11 und MPE IRTC für Amtel AVR.

LEGO RCX–Verleih

Seit unserem Gewinn (VD 1/2001 S.30) verfügt unsere Schule über so ausreichend viele RCX–Komponenten, dass ich meine privat eingebrachten Dinge nun Anderen, vorzugsweise Mitgliedern der Forth–Gesellschaft e. V., zur Verfügung stellen kann.

Angeboten wird: Ein komplettes LEGO–RCX–Set, so wie es für ca. 230,– € im Handel zu erwerben ist.

Inhalt:

1 RCX, 1 Sendeturm, 2 Motoren, 4 Sensoren und ca. 1.000 LEGO Steine.

Anfragen bitte an
Martin.Bitter@t-online.de

Letztlich enthält das Ganze auch nicht mehr als einen Mikrocontroller der Familie H8/300 von Hitachi, ein paar Treiber und etwas Peripherie. Zudem: dieses Teil ist „narrensicher“!

RetroForth
Linux · Windows · Native
Generic · L4Ka::Pistachio · Dex4u
Public Domain
`http://www.retroforth.org`
`http://retro.tunes.org`

Diese Anzeige wird gesponsort von:
EDV–Beratung Schmiedl, Am Bräuweiher 4, 93499 Zandt

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

KIMA Echtzeitsysteme GmbH

Tel.: 02461/690–380
Fax: 02461/690–387 oder –100
Karl–Heinz–Beckurts–Str. 13
52428 Jülich

Automatisierungstechnik: Fortgeschrittene Steuerungen für die Verfahrenstechnik, Schaltanlagenbau, Projektierung, Sensorik, Maschinenüberwachungen. Echtzeitrechnersysteme: für Werkzeug– und Sondermaschinen, Fuzzy Logic.

FORTecH Software
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

Budapester Straße 80 a D–18057 Rostock
Tel.: (0381) 46 13 99 10 Fax: (0381) 4 58 34 88

PC–basierte Forth–Entwicklungswerkzeuge, comFORTH für Windows und eingebettete und verteilte Systeme. Softwareentwicklung für Windows und Mikrocontroller mit Forth, C/C++, Delphi und Basic. Entwicklung von Gerätetreibern und Kommunikationssoftware für Windows 3.1, Windows95 und WindowsNT. Beratung zu Software–/Systementwurf. Mehr als 15 Jahre Erfahrung.

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

Ingenieurbüro
Klaus Kohl

Tel.: 07044/908789
Buchenweg 11
D–71299 Wimsheim

FORTH–Software (volksFORTH, KKFORTH und viele PDVersionen). FORTH–Hardware (z.B. Super8) und Literaturservice. Professionelle Entwicklung für Steuerungs– und Meßtechnik.

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 19 02 25
80602 München
Tel: (0 89) 1 23 47 84
E–Mail: `Secretary@forth-ev.de`
`Direktorium@forth-ev.de`
Bankverbindung: Postbank Hamburg
BLZ 200 100 20
Kto 563 211 208
IBAN: DE60 2001 0020 0563 2112 08
BIC: PBNKDEFF

Redaktion & Layout

Bernd Paysan, Ulrich Hoffmann
E–Mail: `4d@forth-ev.de`

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

Januar, April, Juli, Oktober jeweils in der dritten Woche

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00€ + Porto u. Verpackung

Liebe Leser,

die Beteiligung an der Umfrage fiel diesmal reger aus und zeigt uns, dass die Serie „Forth von der Pike auf“ nach wie vor das Highlight ist. Auch die anderen längeren Artikel sind beliebt; wobei auch mathematisch nicht ganz leichte Kost nicht verachtet wird. Wir werden sehen, ob wir vergleichbare Artikel auch in Zukunft auftreiben können.

Artikel	Stimmen
Ein neuer Direktor stellt sich vor	1 (5,26%)
Forth von der Pike auf	7 (36,84%)
Adventures in Forth	3 (15,79%)
Gforth–R8C erzeugt Sudoku–Rätsel–Vorgaben	5 (26,32%)
Galois–Felder	3 (15,79%)
Das Forth–eV–Wiki	0 (0,00%)
Buchbesprechung: Designing Embedded Hardware	0 (0,00%)

Das Wiki konnte aber keine große Begeisterung erwecken. Das sieht man auch an der Beteiligung: Bis auf die üblichen Verdächtigen beteiligt sich kaum jemand am Wiki (`www.forth-ev.de/wiki`). Zu den üblichen Verdächtigen gehören leider auch missliebige Internetzeitgenossen, die hauptsächlich Aufmerksamkeit für körperteilvergrößernde Substanzen erwecken wollen — auch das Blog lockt solche Typen an. Deshalb bedarf es jetzt leider einer Freischaltung der neuen Accounts durch den Administrator.

Unsere herzlichen Glückwünsche gehen an Carsten Strotmann, der beim jährlichen Software–Wettbewerb der Atari Bit Byter User Group, ABBUC, mit seiner Portierung von volksForth auf die 8Bit–Ataris 800XL und 130XE den ersten Preis gewonnen hat (vgl. Seite 28). Ein phantastisches Ergebnis, Carsten!

Eine betrübliche Nachricht erreicht uns von unseren Forth–Freunden aus den Niederlanden (vgl. Seite 27). Ron Minke, langjähriger Editor des niederländischen Forth–Magazins *Vijgeblaadje*, kann leider die Redaktion nicht mehr weiterführen und seine Nachfolge ist bisher ungeklärt. Wir selbst haben vor einem Jahr vor der gleichen Situation gestanden. Wir sind froh, dass es uns durch das verteilte Arbeiten in einem Redaktions–Team, unterstützt durch unseren Internet–Server `www.forth-ev.de`, bisher gelungen ist, die Vierte Dimension in bewährter Qualität weiter herauszugeben. Vielleicht ist unser Vorgehen ja ein Modell auch für das Produzieren der momentan nicht erscheinenden Forth–Magazine in England und den Niederlanden? Über unsere Erfahrungen, die Vierte Dimension zu produzieren, und über die Technik, die wir dabei einsetzen, haben wir ja auf der Forth–Tagung und auch hier schon berichtet. Weitere Hilfestellungen geben wir gern...

Wie auch immer: Entscheidend für das Funktionieren des Forth–Magazins sind dabei nach wie vor die Artikel, die Ihr, liebe Leser, schreibt. Einsendungen sind immer hochwillkommen.

Bernd Paysan und Ulrich Hoffmann

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
`http://www.forth-ev.de/filemgmt/index.php`

Die Forth–Gesellschaft e. V. wird durch ihr Direktorium vertreten:

Ulrich Hoffmann Kontakt: `Direktorium@Forth-ev.de`
Bernd Paysan
Ewald Rieger

Zu den *Galois-Feldern* von Jens Storjohann (VD-Heft 3/2006)

Der Artikel von Jens Storjohann gefällt mir gut. Jens hat seine ganze Liebe zur Mathematik in diesen Artikel gepackt. Wenn man beiläufig erfährt, dass gewisse nichtalltägliche mathematische Strukturen dazu beitragen, Übertragungen von und zu Zip-Laufwerken durch redundante Codierung fehlerfrei zu halten, dann ist das eine gute und praktische Anwendung auch ausgefallener Seiten der Mathematik. Wenn man von Jens erklärt bekommt, dass Forth dabei helfen kann, solche ungewohnten Strukturen in den Griff zu bekommen, dann ist das eine gute und praktische Anwendung von Forth.

Bei der automatischen Fehlerkorrektur kommen Galois-Felder (endliche Körper) vor. Galois-Felder können über (irreduzible) Polynome eingeführt werden. Algebraische Polynome dürfen nicht mit Polynomfunktionen der Analysis verwechselt werden. Die Frage nach der Bedeutung der Unbestimmten X bei algebraischen Polynomen ist sinnlos. Es kommt nur auf die Koeffizienten an. Die Unbestimmte hat lediglich symbolischen Charakter und erleichtert uns das Merken der Zusammenhänge (beim Multiplizieren so tun, als ob es sich um Polynomfunktionen handelt). Jens erklärt es uns.

Jetzt wird uns auch klar, warum man bei den komplexen Zahlen $a+ib$ nicht nach der Bedeutung der Plusverknüpfung fragen sollte. Und das i als Wurzel aus -1 ist sowieso absurd: Erst erklärt man die Vorzeichenregeln (per definitionem!) so, dass + mal + und auch − mal − das Vorzeichen + ergeben, und dann sucht man nach einer *Zahl*, deren Quadrat -1 ist.

Will man *sauber* arbeiten, dann braucht man ja komplexe Zahlen nur als 2-dimensionale reelle Vektoren aufzufassen. Die Addition ist die übliche Vektoraddition, $(a,b)+(c,d) = (a+c,b+d)$, und als Multiplikation verlangt man $(a,b)*(c,d) = (a*c-b*d, a*d+b*c)$. Da ist nichts Verbotenes dran, nichts Imaginäres, nichts Unvorstellbares. Es gibt andere *Vektormultiplikationen*, die in Mathematik, Physik und Technik eine Rolle spielen, und in der Gaußschen Zahlenebene werden komplexe Zahlen ja sowieso als Punkte in der Ebene (identifizierbar mit Ortsvektoren) dargestellt. Gehen wir also wie Jens mit seinen Polynomen vor und vergessen wir + und i in $a+ib$!

Jens regt uns mit seinem Artikel zum Nachdenken an. Danke Jens.

Fred Behringer

Artikel Galois-Felder

Sehr geehrte Herren, [...]
Den Artikel über Galois-Felder finde ich sehr interessant und verständlich. So etwas könnte man öfter lesen!
Mit freundlichen Grüßen

Paul E. Schey (Mainz)

RCX–Nachfolger NXT

Hallo Martin, liebe Lego–Roboter–Interessierte,

der NXT kostet bei LPE 138,– Euro.

Der NXT (Nachfolger des RCX für das Lego-Mindstorms-Projekt) ist ein 32-Bit–Mikroprozessor mit 256 kB Flash und 64 kB RAM. 4 Eingänge, 3 Ausgänge. Bluetooth-Kommunikation über PC, Mobiltelefon, PDA und über andere NXT–Bausteine. USB-2.0-Anschluss. Programmierbares grafisches Display, Lautsprecher. Stromversorgung: 6 x 1,5–V–AA–Batterien oder Akku 9798 und Netztrafo 9833. Bestellnummer 40309.230.242.

LPE Technische Medien GmbH
Schwanheimer Str. 27
69401 Eberbach

`www.technik-lpe.de`

`info@technik-lpe.com`

An diese Information bin ich über Liesl Rohrmayer gekommen. Sie ist bei LPE seit unserer Vaterstettener Roboter–Aktivitäten *Kundin*.

Über den NXT habe ich, soweit ich mich erinnere, das erste Mal von Henry Vinerts erfahren.

Ich hätte nichts dagegen, wenn wir die Meldung in das Heft 4/2006 der Vierten Dimension aufnehmen könnten.

Herzlichen Gruß

Fred

...

Hallo Fred,

ich sehe da 160.08€, d.h. bei deinem Preis ist keine Mehrwertsteuer dabei. Und das ist nur der Klotz selbst, ohne Peripherie. Das Education-Basisset kostet bei LPE 307,40€, die Standardversion 279€ UVP (gibt's nicht bei LPE). Die Schulversion enthält einen LiIon–Akku, dafür aber deutlich weniger Lego–Bausteine.

Und man muss noch einen Bluetooth–Adapter kaufen, damit man sinnvoll mit den Teilen reden kann (nämlich drahtlos).

> Der NXT (Nachfolger des RCX für das Lego–Mindstorms–Projekt) ist ein 32–Bit–Mikroprozessor mit 256 kB Flash und 64 kB RAM. 4 Eingänge, 3 Ausgänge. Bluetooth-Kommunikation über PC, Mobiltelefon, PDA und über andere NXT-Bausteine. USB–2.0–Anschluss. Programmierbares grafisches Display, Lautsprecher. Stromversorgung: 6 x 1,5–V–AA–Batterien oder Akku 9798 und Netztrafo 9833. Bestellnummer 40309.230.242.

Für alle, die beim von Martin Bitter ausgerichteten Wettbewerb mitgemacht haben: Das NXT berücksichtigt praktisch alle Kritikpunkte. Es ist über Bluetooth

ansprechbar, d. h. kein Übersprechen zwischen verschiedenen Sendern in einem Raum. Die Motoren haben einen Rotationssensor.

Die Software ist Labview–basiert, also von einer Firma, die vor einiger Zeit auch mal Forth (Asyst) gemacht hat.

Der Hauptprozessor ist ein ARM, einen 8–Bit–Koprozessor (AVR) hat das Gerät auch noch drin. Wikipedia weiß mehr:

`http://de.wikipedia.org/wiki/LEGO_Mindstorms`

> Über den NXT habe ich, soweit ich mich erinnere, das erste Mal von Henry Vinerts erfahren.

Ich auf heise Online.

> Ich hätte nichts dagegen, wenn wir die Meldung in das Heft 4/2006 der Vierten Dimension aufnehmen könnten.

Sollen wir nicht ein Projekt daraus machen?

Viele Grüße,

Bernd

Bretonischer Swap

Auf dem Weg durch die Bretagne haben wir nicht schlecht gestaunt, als wir feststellen durften, dass wir auf einem Zeltplatz sozusagen unter den Fittichen von SWAPs Ur–Opa nächtigen konnten. Das Städtchen Guidel dokumentiert die Verbundenheit zur Familie des Swap seit Generationen in seinem Stadtwappen. Die Straße *Rue du 19e Dragons*, die von einem Kreisverkehr direkt zum nahen Atlantik abzweigt, meint sicher weniger den Drachen selbst, als vielmehr jene Soldaten, die sich unter seinen Schutz gestellt haben. Uns war das aber in jedem Fall das anhängende Photo wert.

fep

Elektor–Spezial *Mikro–Controller*

Liebe RC8– und Forth–Interessierte,

das Elektor–Spezial–Projekt–Heft *Mikro–Controller* ist erschienen. 114 Seiten. Auf Seite 3 steht, dass *dieses Heft das erste in einer losen Mikrocontroller–Sonderheft–Reihe ist, über deren Fortführung letztendlich Sie als Leser entscheiden. Deshalb bitten wir Sie, uns mit konstruktiver Kritik, mit neuen Ideen, Entwürfen, Beiträgen und Schaltungen tatkräftig zu unterstützen. Denn: Ein Heft ist nur so gut, wie seine Leser aktiv sind.* — Ende des Zitats.

Ich meine, da könnten wir (die aktiven Forth–Autoren) uns dranhängen?

Die Zeitschrift Elektor ist inzwischen noch viel mehr international geworden. Aus dem Impressum: Elektor erscheint auch in Englisch, Französisch, Niederländisch und weiteren Sprachen. Elektor ist in über 50 Ländern erhältlich.

Ich bin (in zwei Anläufen) gefragt worden, ob ich nicht Lust hätte, Artikel aus dem Holländischen zu übersetzen. Ich habe (vorsichtig) abgelehnt, da das Ganze auf eine Vollzeit–Beschäftigung hinauslaufen würde. — Und ich könnte mich dabei nicht auf Forth beschränken. Das behagt mir nicht.

Vonseiten der holländischen Forth–Freunde wird deren Redakteur, Ron Minke, sich über den starken Bezug auf die ATmega–Reihe auf der dem Heft beiliegenden CD freuen — und sein Forth–System dafür (siehe seine Artikelserie) beschleunigt abschließen.

Mein jahrelanges Ziel, Forth in die hierfür geeignete Heftewelt einzuschleusen (Jörg Plewe hat, soweit ich mich erinnere, das früher schon mal geschafft, wir haben es mit Kleinanzeigen versucht), ist erreicht: Wir (Forth, die FG und die VD) stehen in einer Elektor–Publikation (*Forth auf dem R8C: Blinkende LEDs und Sudoku*), haben einen Fuß in der Tür, und was wir daraus machen, liegt ganz bei uns. Oder? Vielleicht bin ich zu schnell vorgeprescht, aber zu viele Unwägbarkeiten haben mich von langen Absprachen in der FG abgehalten. Und die Elektor–Redaktion sucht ja für Spezial–Heft 2 weitere Artikel von weiteren Autoren. Der oben stehende Aufruf der Elektor–Redaktion kann schließlich nur heißen, dass die Sonderheft–Reihe nur dann fortgeführt wird, wenn genügend Artikel eintrudeln. Und warum nicht auch Forth dort zu einer festen Einrichtung werden lassen?

Hier der Blickfang (Eye–Catcher) und gleichzeitig Zusammenfassungsteil des Gforth–R8C–Artikels auf Seite 52:

Forth gilt unter Entwicklern als Geheimtipp: Interaktiv wie BASIC und gleichzeitig compilierend, mit einfacher Einbindung von Assemblerteilen. Forth eignet sich gut für das Rapid Prototyping. *Ausgangspunkt ist das in Zusammenarbeit mit Heinz Schnitter entwickelte Gforth–R8C–System von Bernd Paysan. Der gesamte Forth–Compiler wird in den R8C platziert! Um zu zeigen, dass man selbst mit den eingeschränkten Mitteln des Elektor–R8C/13–Projektes mit Forth nicht nur einfache Steuerungsaufgaben erledigen, sondern auch anspruchsvollere Ziele verfolgen kann, wird ein Programm gezeigt, mit dessen Hilfe man beliebig viele Sudoku–Rätsel–Vorgaben erzeugen und am Bildschirm anzeigen lassen kann.*

Der Screenshot auf Seite 53 wurde von der Redaktion ohne Rückfrage bei mir in letzter Minute aus der FG–Homesite geholt und eingeschoben, so dass ich keine Gelegenheit mehr sah, in den Text einen Hinweis auf Bernd einzubauen.

Dem Heft liegt eine CD bei. Ich gebe (ohne Nachprüfung, da ich das schnellstmöglich erledigen möchte) die Angaben dazu aus dem Heft wieder:

- Software, Informationen und Projekte. Entwicklungsoberfläche mit C–Compiler und Debugger, Datenblätter und Applikationen, das C–Projekt *Ports*.
- FPGA–Entwurf in VHDL: Das komplette Pong–Projekt.
- Das Forth–System für den R8C/13: gforth–r8c.mot .
- Hausautomation mit X–10: Das E–blocks–Projekt JDx10_2.fcf .
- Schaltplan, Platinenlayout und das C–Projekt zur DCF77–Funkuhr mit dem R8C/13.
- Kalibrieren mechanischer Uhren: Schaltplan, Platinenlayout und Software zur Zeitwaage mit dem ATMEGA32.
- Das C–Projekt *freq* mit Frequenzmessung, L/C–Messung und Signalgeneratoren.
- Daten und Software zum Webserver mit dem W3150A von WIZnet.
- Software und Dokumentation zum MicroSPS–Projekt inkl. Eagle 4.16.
- MicroSPS mit dem ATMEGA32: Platinenlayout und Software inkl. aller C–Quelltexte.
- Platinenlayout und Entwicklungssoftware zum CAN–Interface für den PC.
- Das PSoC–Projekt SinusLP mit Sinusgenerator und Tiefpassfilter.
- Starterkit für den ATmega169: Dokumentation und Win–AVR–Quelltexte.

Im Heft steht auch ein Artikel über *Fox on the Run: Das Mikro–Linux–System FOX*.

Herzlichen Gruß

Fred

Elektor Sonderheft

Liebe Leute,

ich hab gestern eher zufällig das Elektor–Sonderheft *Mikro-Controller 1* erstanden — mit EUR 16.90 nicht gerade ein Spezialschnäppchen, aber egal. Und was sehen meine entzündeten Glasaugen? Einen Artikel von Fred Behringer über Forth! Respekt, Herr Behringer! Scheint wohl so zu sein, dass das Gforth–R8C–Projekt doch auf Interesse stößt.

Grüße, Erich Wälde (aus `de.comp.lang.forth`)

COMUS

Some Commonly Used Forth Words
Eine interessante Sammlung von Forthworten die **nicht** ANS sind, hat Leo Wong 2004 zusammengestellt. 68 Beispiele sind inzwischen gesammelt, Dinge wie:

```
," string" ( -- ) \ Reserve space for and store string.
INCLUDE filename ( -- ) \ Include source code
CLEAR ( i*x -- ) \ Empty the stack
RANDOM ( -- u ) \ Return a random integer.
```

um mal einige zu nennen. Viel Vergnügen beim Stöbern auf: `http://www.albany.net/~hello/comus.htm`

mka

Forth und seine Zukunft

Hi Freunde,

mit Friederich Prinz hatte ich eine lebhafte Diskussion über Forth — er hat seine Seite vertreten und ich meine Seite. Jetzt hat ein neues Team das übernommen. Aber ich will eines von neuem klarstellen:

Wer Forth auf SWAP, TUCK, FUCK — sprich: Stack-Operationen reduziert, der hat FORTH einfach NICHT kapiert. Der Vorteil von Forth liegt darin, daß es die CREATE– und DOES>–Konstrukte hat. Und die können sogar beliebig geschachtelt sein.

Ich plädiere hiermit erneut, daß der *Swap-Drachen* in *Create–Does*–Drachen umgenannt wird — alles andere ist Anachronismus!

Und daß Programmieraufgaben als Rätsel gestellt werden, die nur durch die Verwendung von CREATE und DOES> gelöst werden können.

Daneben sollte ein Einführungskurs laufen — einfach *Appetitanreger*, aber keine scholastischen Artikel.

PS: Freigegeben als ein Leserartikel.

CU Uli Paul (Stadtbergen)

Forth von der Pike auf: Praxiseinsatz

Hallo Redaktion,

auch wenn ich die VD eher zufällig im Internet gefunden habe, so hat mich die Artikelreihe zum Thema Forth auf dem AVR–ATmega inspiriert, ein Forth mit den Ideen daraus real umzusetzen. Das Ergebnis (oder besser das laufende Projekt) liegt im Quelltext (Assembler und „self-hosted“ forth) und GNU Lizenz bei `http://sf.net/projects/amforth`. Auch ohne das Ende der Reihe zu kennen, funktionieren Interpreter und Colon-Compiler ;=) Die Ausführungen der Artikelserie haben sehr geholfen, das Grundsystem zum Laufen zu bringen. Die ersten Optimierungen, die diese Grundlage doch deutlich ändern dürften, sind schon in der Planung. Wie etwa die sehr interessanten Ausführungen von Anton Ertl in seiner Dissertation.

Mein Ziel ist es, mit dem Forth kleine Roboter (Stichwort Asuro und ct-bot) und eine Modellbahn über das serielle

Terminal zu steuern. Das stelle ich mir deutlich interaktiver und *schöner* vor, als mit C–Programmen herumzuhantieren.

Das implementierte Forth ist so mehr oder weniger entsprechend dem dpans94– Standards umgesetzt, Abweichungen sind mal Absicht, mal Zufall mal Fehler...

Matthias Trute

Mehr Forth von der Pike auf

Dag Fred,

in deiner zweiten Mail hast du mir vom Leserbrief von Matthias Trute berichtet. Die Welt ist klein...

Ich fand die Ankündigung eines neuen Forth–Systems von Matthias vor einiger Zeit in einem Forum für AVR–Freaks. Über die zu seinem Projekt angegebene URL habe ich ihm eine Mail gesandt, die vom Site–Moderator freundlicherweise weitergeleitet wurde.

Wir haben dann einige Mails über das in der Artikelserie dargestellte System *Forth von der Pike auf* ausgetauscht.

Es ist wirklich faszinierend zu sehen, wie eine Idee über eine Webseite in Norwegen (AVRFreaks) und eine Webseite in Amerika (SourceForge) wieder zu mir zurückfindet.

Ich denke, Matthias wird die Informationen aus den letzten beiden Artikelteilen, die jetzt ins VD–Heft kommen, gut gebrauchen können. Wenn das Heft erschienen ist (Übers.: Mit dem vorliegenden Heft geschehen), werde ich gern mit Matthias in Gedankenaustausch darüber treten, wie man daraus ein brauchbares Forth–System machen kann.

Die ausführliche Dokumentation des Forth–Systems aus der Artikelserie (das auf FIG aus dem Jahre 1982 aufbaut) ist soweit klar. Wenn die Software gut die (angepasste) Testserie von John Hayes bestanden hat, werde ich das Ganze ins Internet setzen. Eine Seite liegt bereits fertig auf der Fgg–Website (Übers.: holländische Website, über `www.forth-ev.de` zu erreichen). Sie enthält jedoch noch keine großen Informationen.

Ich freue mich auch riesig darüber, dass meine Artikelserie so viel Aufmerksamkeit auf sich gezogen hat. Wie es scheint, gibt ein Artikel mit einem so gewählten Ausgangspunkt anderen so viel Anregung, dass sie auf allerniedrigstem Software–Niveau (mit entsprechender Hardware) darüber nachzudenken beginnen, wie ein Forth–System zusammengebaut werden kann. Ich fühle mich sehr geehrt!

Die Ergebnisse der ganzen Anstrengungen werden im Laufe der Zeit auf der Fgg–Website zu lesen und von dort herunterzuladen sein.

(Ich habe da auch noch eine R8C–Platine...)

Bis demnächst, jetzt erst einmal raus mit dieser Mail!

Ich wünsche dir und deinen Angehörigen eine recht frohe Weihnachtszeit und ein gesundes 2007!

Ich hoffe, dass wir uns auch im Jahre 2007 noch häufig auf Internetwegen treffen und unsere Gedanken austauschen können.

Recht herzliche Grüße,

Ron Minke Übersetzt von Fred Behringer

weitere Leserbriefe und Meldungen auf Seite 22

Blick auf die beiden Brücken am Firth of Forth

Adventures in Forth 2

Erich Wälde

Im ersten Teil (1) wurde gezeigt, wie man dem Renesas–R8C13–Kontroller beibringt, mit einem i2c–Bus zu reden. Inzwischen hat Bernd Paysan das r8c–Forth und das zugehörige `terminal.fs` so aufgebohrt, dass man in einem Forth–Programm die Anweisung `include` verwenden kann. Daher kann man ein längeres Programm in mehrere Dateien zerlegen. Man kann die Dateien auch so schreiben, dass sie von mehreren Projekten benutzt werden können, eine angenehme Erleichterung.

Ich möchte mit dem R8C–Kontroller eine Meßstation aufbauen, die permanent laufen soll. Um damit zeitliche Abläufe zu organisieren, wäre eine Art Uhr nicht schlecht. Dann kann ich einfacher sagen, *alle 10 Minuten sollen die Messwerte verschickt werden.*

Eine Forth Uhr

Um das Verstreichen der Zeit zu messen, bedient man sich am besten der Timer, die der Kontroller mitbringt. Man konfiguriert einen Timer so, dass er immer nach einer (oder wenigen) Millisekunden einen Interrupt auslöst. Die dazugehörige Interrupt–Service–Routine zählt einen Zähler hoch und setzt einen *flag*, der dem übrigen Programm anzeigt, dass eine Millisekunde vergangen ist. Jetzt kann man sich im Quelltext von gforth–r8c schlau machen, wie man einem timer eine Interrupt–Service–Routine unterjubelt. Allerdings habe ich das als Anfänger nicht genügend durchschaut. Es geht auch ohne Durchblick.

Bei ebendieser Suche (`arch/r8c/prim.fs`) findet sich eine Variable `timer`, die vom Wort `ms-irq` hochgezählt wird. Es gibt also bereits eine Art Uhren-Tick (TimerC–Zyklus), den ich für die Uhr verwenden kann.

In einem Buch über PIC–Mikrokontroller (2) findet sich eine geeignete Methode, um eine Zeitzählung zu realisieren. Das Programm läuft durch eine Schleife, in der immer wieder überprüft wird, ob ein Tick abgelaufen ist. Wenn ja, dann werden die Zähler, die die Zeit verwalten, hochgezählt (Ticks oder Millisekunden, Sekunden, Minuten, Stunden . . .). Wenn eine ganze Sekunde verstrichen ist, dann wird der Sekundenzähler ebenfalls erhöht, und ein flag gesetzt, welcher ebendies anzeigt. Diese Routine heißt in dem o.g. Buch `timeup`[1]. Wenn `timeup` zurückkehrt, sind alle Zähler aktualisiert, und die flags gesetzt.

Im darauf folgenden Teil der Schleife werden die flags überprüft, die dazugehörigen Aufgaben (*jobs)* aufgerufen, und danach die flags gelöscht. Wenn mehrere flags gesetzt sind, kann man entweder alle zugehörigen Aufgaben ausführen und erst danach die Schleife von vorne beginnen (*serielle Schleife*[2]) — oder man kehrt in die Schleife nach jeder Aufgabe zurück (*parallele* Form). Das kann von Vorteil sein, wenn man eine Aufgabe hat, die alle n Millisekunden (Tick) auszuführen ist — z.B. die Bedienung einer LED–Ziffernanzeige. Dann kann man zwischen den einzelnen Jobs immer auf den nächsten Tick warten und diese spezielle Aufgabe einschieben.

Der Nachteil, dass beispielsweise eine Stundenaufgabe erst 3 Ticks nach der vollen Stunde aufgerufen wird, ist dagegen klein. Der *Pseudocode* für die serielle Schleife sieht dann etwa so aus:

```
while (true)
{   if (tickover?)
    {   timeup aufrufen
    }
    if ( Tick-flag? )
    {   Tick-Aufgabe aufrufen
        Tick-flag loeschen
    }
    if ( Sekunden-flag? )
    {   Sekunden-Aufgabe aufrufen
        Sekunden-flag loeschen
    }
    if ( Minuten-flag? )
    ...
}
```

Es gilt also, die Funktion `timeup` nachzubilden und die Schleife zu schreiben.

Zeit messen: `timeup`

Das folgende Verfahren hat sich für das Innenleben von `timeup` bewährt: zuerst lesen wir `timer` aus, zählen einen bestimmten Wert (z.B. 500) dazu und speichern das Ergebnis in `newtimer`. Wenn `timer` den Wert von `newtimer` erreicht oder überschritten hat, dann sind 500 ms verstrichen. Dann rufen wir `timeup` auf, um die Zähler zu aktualisieren. Anschließend wird zu `newtimer` wieder 500 addiert. Sollte `timeup` etwas zu spät aufgerufen werden, macht das nichts, weil sich diese Fehler nicht addieren. Wir zählen immer die gleiche Zahl zu `newtimer`. Anders wäre es, wenn wir den Zähler für den Timer löschen und den Timer dann neu starten.

Wir schreiben also erst mal eine abgespeckte Version von `timeup`. TimerC gibt Zyklen vor, die in der Variablen `timer` gespeichert werden. Im Beispiel ist nach 500 Zyklen ein Tick vergangen, nach 2 Ticks eine Sekunde. `timeup` wird aufgerufen, wenn ein Tick verstrichen ist. Die niedrigste Stufe von `timeup` zählt diese Ticks zu Sekunden. Mit Zyklen und Ticks lassen sich auch jobs realisieren, die mehrfach pro Sekunde laufen müssen.

[1] Ich hab's grad wieder nachgelesen: TIMUP heißt sie dort.

[2] Aus der Struktur des zugehörigen Ablaufdiagramms.

```
Variable newtimer
Variable lastsec
Variable tickflag
Variable tick
Variable secflag
Variable sec
Variable minflag
Variable min
500 Constant cycles.tick \ timerC Zykl./Tick
2   Constant ticks.sec   \ Ticks/Sekunde

: tickover? ( -- ) newtimer @ timer @ - 0< ;
: timeup ( -- )
  cycles.tick newtimer +!
  1 tickflag !                  \ tick over!
  1 tick +!
  tick @ ticks.sec >= IF
    0 tick !
    1 secflag !                 \ sec over!
    1 sec +!
  ENDIF
  sec @ 60 >= IF
    0 sec !
    1 minflag !                 \ min over!
    1 min +!
  ENDIF
  min @ 60 >= IF
    0 min !
  ENDIF
;
```

Was schaffen: `run`

Die Hauptschleife ist auch keine Zauberei:

```
: run
  init-loop
  BEGIN
    tickover?  IF timeup ENDIF

    tickflag @ IF job.tick 0 tickflag ! ENDIF
    secflag  @ IF job.sec  0 secflag !  ENDIF
    minflag  @ IF job.min  0 minflag !  ENDIF

  key? UNTIL
;
```

Zu Beginn der Schleife wird lediglich `newtimer` initialisiert. Bleiben nur die Jobs, damit auch was passiert:

```
: led0 ( -- ) 3 port1 bclr ;
: led1 ( -- ) 3 port1 bset ;
: job.tick
  tick @ $01 and IF led0 ELSE led1 ENDIF
;
: job.sec
  timer @ dup lastsec @ - swap dup lastsec !
  sec @ min @
  cr . . . .
;
: job.min
  cr ." running minute job ..."
;
: init-loop ( -- )
  cr ." min sec timer timer-lastsec"
  timer @ cycles.tick + newtimer !
;
```

Der zu einem abgelaufenen Tick gehörige `job.tick` lässt hier die LED an Pin P1.3 im Sekundentakt blinken. Der Sekundenjob gibt die aktuellen Zähler aus, sowie den Abstand seit dem letzten Sekundenjob in TimerC–Zyklen.

```
include adv2_1.fs  ok
58 sec !  ok
run
min sec timer timer-lastsec
0 59 -26991 1000
1 0 -25991 1000
running minute job ...
1 1 -24991 1000
...
```

Die Zähler werden hochgezählt, ganz so, wie man sich das vorstellt. Diese Schleife lässt sich schon mal mit der Stopuhr testen. Man beachte auch, dass die Hauptschleife recht simpel aussieht. Die einzelnen Worte sind auch überschaubar.

Optimierungen 1

Ein komplettes Wort, also 16 Bit, zu belegen, wo auch ein einziges Bit ausreicht, das ist Verschwendung. Unsere flags finden in einer Variablen bequem Platz. Dafür müssen wir uns merken, welches flag in welcher Bitposition wohnt:

```
Variable Flags
2 Flags bset      \ Setze   Flag[Minute]
1 Flags bclr      \ Loesche Flag[Sekunde]
0 Flags btst IF   \ Flag[Tick] gesetzt?
  timeup
ENDIF
```

Gespart werden hier 6 Worte im RAM: die Variablen `tickflag secflag .. yearflag` werden alle in `Flags` untergebracht.

Uhr und Kalender

`timeup` wird zwar länglich, wenn man die Behandlung von Stunden, Tagen, Monaten und Jahren dazupackt, aber nicht komplexer. `timeup` erhält im nächsten Schritt die restlichen Zähler. Sofort braucht man ein Wort, welches die Länge eines bestimmten Monats ausgibt, inklusive Behandlung der Schaltjahre. Ein Schaltjahr ist ein Jahr, welches durch 4 aber nicht durch 100 teilbar ist, oder welches durch 400 teilbar ist. In Stephen Pelcs Buch *Programming Forth* (3) findet sich diese Anweisung in sehr schöner, kompakter Form:

```
: leap_year ( year -- t/f )
  dup    4 mod 0=
  over 100 mod 0<> and
  swap 400 mod 0=  or ;
```

Für die Monate habe ich eine Tabelle `MaxDay` als *byte array* angelegt, weil alle Werte unter 255 liegen. Der gesuchte Monat wird um 1 vermindert und dient dann als Index in dieses array. Dazu kommt die korrekte Behandlung des Februars in Schaltjahren.

```
ram
create MaxDay  31 c, 28 c, 31 c, 30 c, 31 c,
   30 c, 31 c, 31 c, 30 c, 31 c, 30 c, 31 c,
rom
: length_of_month ( year month -- maxday )
  dup 1-                \ array starts at 0
  MaxDay + c@           \ get length
  swap 2 = IF           \ if month == 2
    swap leap_year IF   \ and leap_year
      1+                \   length += 1
    ENDIF
  ELSE
    swap drop           \ remove year
  ENDIF ;
```

Diese Worte lassen sich testen — wie immer in Forth direkt von der Eingabezeile:

```
1900 leap_year . 0  ok
2000 leap_year . -1  ok
2001 leap_year . 0  ok
2004 leap_year . -1  ok
2004 2 length_of_month . 29  ok
2006 2 length_of_month . 28  ok
2006 3 length_of_month . 31  ok
2006 4 length_of_month . 30  ok
...
```

Bei `timeup` muss man noch darauf achten, dass die Zähler für Sekunden, Minuten, Stunden gegen den ersten ungültigen Wert (60, 60, 24) mit `>=` getestet werden. Bei der Länge des Monats produziert `length_of_month` aber den höchsten gültigen Wert, also ist der Vergleich ein `>` und der Startwert 1 und nicht 0. Analog für den `month` Zähler:

```
: timeup ( -- )
...
  day @ year @ month @ length_of_month > IF
    1 day !                     \ offset 1!
    5 Flags bset                \ month over
    1 month +!
  ENDIF
  month @ 12 > IF
    1 month !                   \ offset 1!
    6 Flags bset                \ year over
    1 year +!
  ENDIF
;
```

Der komplette Programmtext findet sich in `adv2_2.fs` und produziert etwa folgende Ausgabe:

```
include adv2_2.fs  ok
23 hour ! 59 min ! 57 sec ! 0 tick !  ok
2006 year ! 12 month ! 31 day !  ok
run
year month day hour min sec timer...
2006 12 31 23 59 57 7812 7812
2006 12 31 23 59 58 8813 1001
2006 12 31 23 59 59 9813 1000
2007 1 1 0 0 0 10813 1000
running minute job ...
running hour job ...
running day job ...
running month job ...
running year job ...
2007 1 1 0 0 1 11813 1000
2007 1 1 0 0 2 12813 1000
2007 1 1 0 0 3 13813 1000
...
```

Optimierungen 2

Der Block in `run`, welcher die Flags prüft und die dazugehörigen Jobs aufruft, ist ein Kandidat für eine Schleife.

```
: run
  ...
  BEGIN
   tickover? IF timeup ENDIF
   0 Flags btst IF job.tick 0 Flags bclr ELSE
   1 Flags btst IF job.sec  1 Flags bclr ELSE
   2 Flags btst IF job.min  2 Flags bclr ELSE
   3 Flags btst IF job.hour 3 Flags bclr ELSE
   4 Flags btst IF job.day  4 Flags bclr ELSE
  5 Flags btst IF job.month 5 Flags bclr ELSE
   6 Flags btst IF job.year 6 Flags bclr
   ENDIF ENDIF ENDIF ENDIF ENDIF ENDIF ENDIF
  key? UNTIL
;
```

Für eine Schleife benötigt man lediglich eine Möglichkeit, die Jobs über den Schleifenindex aufzurufen. Dazu erzeugt man eine Tabelle `Jobs`, in der die Einsprungadressen der einzelnen `job`–Worte aufgehoben werden. Das Wort `'` produziert diese Adresse für das nachfolgende Wort:

```
' job.tick . 8842  ok
' job.sec  . 8880  ok
' job.min  . 8958  ok
```

liegt eine solche Adresse auf dem Stack, dann kann diese Adresse mit dem Wort `execute` ausgeführt werden:

```
job.sec
2007 1 1 0 0 4 1581 -2641  ok
' job.sec execute
2007 1 1 0 0 4 10299 8718  ok
```

Die `Jobs` Liste und die Schleife sehen dann so aus:

```
ram
create Jobs ' job.tick ,
  ' job.sec , ' job.min , ' job.hour ,
  ' job.day , ' job.month , ' job.year ,
rom
: run
  init-loop
  BEGIN
```

```
    tickover? IF timeup ENDIF

    7 0 DO
      I Flags btst IF
      I cells Jobs + @ execute
      I Flags bclr
      ENDIF
    LOOP

  key? UNTIL
;
```

Dieser Programmtext ist etwas kürzer als der ursprüngliche Block. Wenn man wissen will, wie groß ein Programm ist, dann macht man mit `dump` einen Speicherabzug. Das Programm wohnt im Datenflash, welcher bei der Adresse `$2000` beginnt und 4 kByte (`$1000`) groß ist. Der freie Speicher enthält lediglich `FF` Werte. Im folgenden Beispiel sind die Zeilen für den Druck umgebrochen:

```
empty  ok
include adv2_2.fs  ok
$2000 $1000 dump
    2000: D4 FD E5 46  6C 61 67 73 -
          68 C0 FF FF  6A 04 00 20
          ...Flagsh...j..
    2010: E8 6E 65 77  74 69 6D 65 -
          72 20 68 C0  FF FF 6C 04
          .newtimer h...l.
...
    2510: 5C C1 1E 25  D8 23 24 C5 -
          06 00 08 20  FC C9 36 DC
          \..%.#$.... ..6.
    2520: 5C C1 7C 24  12 C1 FF FF -
          FF FF FF FF  FF FF FF FF
          \.|$............
$2526 $2000 - . 1318  ok
```

`adv2_2.fs` belegt also 1318 Byte. Das Programm mit der geänderten Hauptschleife (`adv2_3.fs`) belegt 1208 Byte. Damit hat man immerhin 110 Byte oder 8 % eingespart. Ich finde, das ist eine ganze Menge, auch wenn das durch 14 Byte mehr belegtes RAM bezahlt wird.

Uhrzeit auf's LCDisplay

Schön wär's jetzt, wenn die Uhrzeit auf dem Display erscheint und nicht (nur) im Terminal–Fenster. Dazu verwenden wir ein paar einfache Worte, die die Angaben formatieren und anzeigen. Das genaue Format ist natürlich Geschmacksache. Die dafür notwendigen Worte haben wir uns aus dem ersten Artikel ausgeliehen und stehen in den Listings (`adv2_lcd.fs`)

`show.DT` schreibt Datum und Uhrzeit in meinem bevorzugten Format auf das LCDisplay. Ich habe entschieden, `show.DT` nur jede Minute aufzurufen, und jede Sekunde lediglich die Sekundenanzeige zu aktualisieren:

```
: job.sec ( -- )
  ...
  0 13 lcdpos sec @ p2! lcdtype
;
```

```
: job.min ( -- )
  ...
  0 0 lcdpos show.DT
;
```

Außerdem wird `show.DT` vor der Schleife einmal aufgerufen. Die Anzeige funktioniert mit diesen Ergänzungen wunderbar. Allerdings muss nach dem Einschalten des Kontrollers die Zeit erst gestellt werden. Das ist nichts für faule Kerle wie meinereiner. Also habe ich die i2c–Bus Funktionen vom letzten Mal hergenommen und eine batteriegepufferte PCF8583 Uhr an den i2c–Bus angeschlossen.

Wider das Vergessen: i2c–Uhr

Die Uhr anzusprechen, ist keine große Kunst mehr, hier etwa um die Zeit zu lesen. Der einzige Trick hier ist der, dass die Uhr selbst keinen Jahreszähler hat. Lediglich 4 Jahre werden gezählt, um die (meisten) Schaltjahre zu behandeln. Daher habe ich an Adresse 0x10 im EEPROM der Uhr einen 2–Byte–Zähler (nicht BCD, low Byte zuerst) für das ganze Jahr verwendet.

```
: get.rtc
  $01                    \ start address to rtc
  1 i2c_addr_rtc NB>i2c  \ send start address
  6 i2c_addr_rtc NB<i2c  \ read 6 Bytes
  $10                    \ YEAR address
  1 i2c_addr_rtc NB>i2c  \ start address to rtc
  2 i2c_addr_rtc NB<i2c  \ read 2 Bytes: xl xh
  8 lshift +             \ convert to YYYY
;
```

Allerdings handelt man sich bei diesem Modell Zahlen im BCD–Format ein (*binary coded decimal*), d.h. die obere und die untere Hälfte eines Bytes (*nibbles* enthalten je eine Ziffer von 0 bis 9. Die Konvertierung könnte evtl. in Assembler gemacht werden, wenn der Kontroller einen entsprechenden Befehl beinhaltet. Im gforth–r8c hab ich aber kein Wort gefunden. Also basteln wir uns das geschwind selber.

```
: bcd>dec ( n.bcd -- n.dec )
  $ff and
  dup
  4 rshift 10 *    \ extract high nibble as 10s
  swap
  $0f and          \ extract low nibble as 1s
  +                \ add
;
: dec>bcd ( n.dec -- n.bcd )
  &100 mod         \ 99 is largest number for
  &10 /mod         \    8 bit bcd
  4 lshift
  +
  $ff and          \ truncate to 8 bit
;
```

Meine Lösungen sind ganz sicher verbesserungsfähig, aber für diesen Zweck ausreichend.

```
include adv2_4.fs  ok
2006 8 15 18 35 55 0 set.rtc  ok
run
year month day hour min sec timertimer-lastsec
2006 8 15 18 35 56 25877 -25261
2006 8 15 18 35 57 26353 476
2006 8 15 18 35 58 27353 1000
2006 8 15 18 35 59 28353 1000
2006 8 15 18 36 0 29353 1000
running minute job ...
2006 8 15 18 36 1 30353 1000
2006 8 15 18 36 2 31353 1000
...
```

Die Uhrzeit wird zunächst an die i2c–Uhr übertragen, die letzte Zahl vor `set.rtc` sind die Hundertstelsekunden. So ausgerüstet überlebt die Uhr jetzt auch das Ausschalten des Kontrollerboards. Ein 0.47 F–GoldCap Kondensator kann meine Uhr länger als 2 Wochen betreiben. Als Zugabe enthält das Programm noch das LM75–Thermometer von der ersten Folge. Der vollständige Programmtext findet sich in `adv2_4.fs` und den darin eingeschlossenen Dateien.

Optimierungen 3

In der Funktion, äh, im Wort `timeup` sind für jeden Zähler die gleichen Dinge zu erledigen: Zähler erhöhen, falls Grenzwert erreicht: Zähler zurücksetzen und den nächsthöheren Zähler erhöhen, sowie den flag für den nächsthöheren Zähler. Um diese Funktion vollständig in eine Schleife umzuwandeln, sind allerhand Vorarbeiten nötig.

Die Zähler–Variablen `tick` bis `year` werden in einen Datenblock mit dem Namen `Counts` gesteckt. Aus `year @` wird dann `6 cells Counts + @` und analog für alle übrigen Zähler.

Ebenso werden die Grenzwerte in ein Bytearray `Limits` verwandelt. Das klappt zunächst für alle außer dem Monat. Aus `60 >= IF` wird dann eben `2 Limits + c@ >= IF`.

Um den Block für den Monat ebenfalls in die Schleife aufzunehmen, habe ich entschieden, dass ich dafür sorge, dass im Feld 4 von `Limits` immer die Länge des aktuellen Monats steht. Also muss vor Beginn der Schleife, wenn die Uhrzeit aus der i2c–Uhr gelesen wurde, die Länge des aktuellen Monats nach `Limits` kopiert werden:

```
year @ month c@ length_of_month 4 Limits + c!
```

Außerdem muss `job.month` den Eintrag für den neuen Monat aktualisieren. Damit ist unter normalen Umständen die korrekte Behandlung des Monats gesichert.

Verbleibt noch der *Offset 1* Fehler in Tag und Monat. Für beide Zähler habe ich den Startwert auf 0 gesetzt. Damit hat etwa der Januar Tage von 0 bis 30. Die 31 in `Limits` ist jetzt der erste ungültige Wert und der Vergleich im entsprechenden Block von `timeup` wird von `>` zu `>=`, wie bei den anderen Blocks. An allen Stellen, an denen Tag und Monat benutzt werden, muss jetzt 1 dazugezählt oder abgezogen werden. Das ist unübersichtlich, aber es funktioniert. Die Änderungen betreffen auch das Lesen der i2c–Uhr und die Anzeige auf dem LCDisplay.

Nach diesen Veränderungen sehen jetzt alle Blocks gleich aus, bis auf den Index. Jetzt kann man `timeup` tatsächlich enorm reduzieren:

```
: timeup ( -- )
  cycles.tick newtimer +!
  0 Flags bset                  \ tickflag++
  1 Counts +!                   \ tick++
  \ for i in tick sec min hour day month
  6 0 DO
    I cells Counts +  @
    I       Limits + c@
    >= IF                \ if C[i] >= L[i]
      0 I cells Counts + !      \  C[i]=0;
      I 1+ Flags bset           \  F[i+1]++;
      1 I 1+ cells Counts + +! \  C[i+1]++;
    ENDIF                       \ endif
  LOOP
;
```

Das Programm `adv2_4.fs` belegt 2708 Bytes, das neue Programm `adv2_5.fs` 2686 Bytes. Wenn man die Worte `tick` bis `year` konsequent durch `n cells Counts +` ersetzt, dann spart man erstaunlicherweise nochmal $2686 - 2652 = 34$ Bytes. Danach ist das Programm aber deutlich unleserlich geworden.

Fazit: Das Programm wird nicht deutlich kleiner, wenn man `timeup` in eine Schleife umwandelt, weil die nötige Infrastruktur die Ersparnis auffrisst.

Uhr als Task

Jetzt bleibt nur noch, die Uhr als Hintergrunds–*Task* laufen zu lassen, so wie das in den Beispielen auf der Wiki–Seite schon geschieht. Dazu habe ich bei Rafael Deliano (4) eine Erklärung gefunden, wie ein *PAUSE–Tasker* generell funktioniert. An der Stelle, an der ein Task unterbrechbar ist, ruft er die Funktion `pause` auf. Dann läuft der nächste Task. Der hier verwendete Tasker kann genau zwei Tasks verwalten. In den Beispielen ist der zweite Task eine Variante von `key`, der unsere Eingaben am Terminal bedient. Man kann also mit dem System reden, obwohl die Uhr läuft.

Die Umwandlung von `run` in ein Task ist fast trivial: `run` wird als `task` gekennzeichnet, die `BEGIN ... UNTIL` Schleife wird in eine `BEGIN ... AGAIN` Schleife umgewandelt, und vor dem Ende der Schleife wird `pause` aufgerufen. Dort darf `run` unterbrochen werden. Ausgehend von `adv2_4.fs` muss man nur `run` ersetzen und den tasker bekannt machen:

```
include adv2_tasker.fs
...
: run task
  init-loop
  BEGIN
    tickover? IF timeup ENDIF

    7 0 DO
```

```
      I Flags btst IF
      I cells Jobs + @ execute
      I Flags bclr
      ENDIF
    LOOP

    pause
  AGAIN
;
```

Wenn man jetzt noch dafür sorgen will, dass die Show sofort nach dem Einschalten losgeht, dann helfen die folgenden Zeilen:

```
include adv2_6.fs
rom
' run IS bootmessge
ram
savesystem
```

Die Adresse des ausführbaren Teils von `run` wird `bootmessage` untergeschoben (`is`). `savesystem` rettet alle Variableninhalte aus dem RAM ins ROM und hinterlegt weiteren Code, der beim Start die Inhalte wieder ins RAM transferiert, bevor das Anwenderprogramm startet. Sehr feine Sache das. Nachdem `adv2_6.fs` geladen und gesichert wurde, belegt es 3094 Byte. Allerdings kann man die Ausgaben am Terminal noch sparen, wenn die Uhr alleine laufen soll.

Fast überflüssig zu erwähnen ... mit den `job`-Worten kann ich jetzt sehr bequem regelmäßige Aufgaben verteilen. Die Anzeige der Sekunde jede Sekunde, das Löschen und Wiederbeschreiben des LCDisplays jede Minute, das Ein-/Ausschalten der LED und die Anzeige der Temperatur alle halbe Sekunde (slow tick) sind die derzeit eingebauten regelmäßigen Aufgaben. Mit etwas Fantasie lässt sich der Kontroller nun zum Schalten zeitlich bestimmter Vorgänge (Rolläden, Licht, Kaffeemaschine) oder sensorgesteuerter Dinge (Gewächshauslüftung, Dachfenster) gebrauchen. Der Haussteuerung in Forth steht nichts mehr im Weg außer dem nun doch etwas begrenzten Speicher von ROM ...

Ausblick

Der damit erreichte Stand der Dinge ist ganz ordentlich. Dennoch verbleiben eine Menge Dinge für einen weiteren Artikel — falls ich die Leser nicht schon über Gebühr strapaziert habe.

Der aufmerksame Leser hat längst erkannt, dass `job.year` die Kopie von `year` im RAM der i2c–Uhr aktualisieren sollte. Die Möglichkeiten und Grenzen des gewählten Verfahrens sollten ausgelotet werden. Messwerte wollen verschickt werden, am besten über die andere serielle Schnittstelle (uart0). Und wie kann man nun dem Forth–System eine neue Interrupt Service Routine unterjubeln, etwa um die Flanken eines DCF77–Empfängers (Zeitsignal) zu detektieren und bedienen? Unbelegte Pins für Experimente sind noch genügend vorhanden.

Als praktische Anwendung möchte ich einen Sensor aufbauen, der die Sonnenscheindauer erfasst — bitte ohne irgendwelche mechanisch bewegten Teile. Das Auslesen von 4 Fotodioden an einem AD–Wandler über den i2c–Bus funktioniert schon. Messungen am Sonnenlicht und das Entwickeln der Datenauswertung fehlen noch.

Ergänzungen, Kommentare, Korrekturen sind ausdrücklich erwünscht. Sie erreichen mich unter `ew.forth@nassur.net`

Referenzen

1. E. Wälde, Adventures in Forth, Die 4. Dimension 3/2006, Jahrgang 22
2. A. König und M. König, Das PICmicro Profi Buch, Franzis Verlag 1999, ISBN 3-7723-4284-1
3. Stephen Pelc, Programming Forth, `http://www.mpeforth.com/arena/ProgramForth.pdf`
4. `http://www.embeddedforth.de/emb3.pdf` S. 9

Listings

```
\ 2006-08-16 EW adv2_1.fs
Variable newtimer
Variable lastsec
Variable tickflag
Variable tick
Variable secflag
Variable sec
Variable minflag
Variable min
500 Constant cycles.tick \ timerC Zykl./Tick
2   Constant ticks.sec   \ Ticks/Sekunde
: tickover? ( -- ) newtimer @ timer @ -  0< ;
: timeup ( -- )
  cycles.tick newtimer +!
  1 tickflag !                  \ tick over!
  1 tick +!
  tick @ ticks.sec >= IF
    0 tick !
    1 secflag !                 \ sec over!
    1 sec +!
  ENDIF
  sec @ 60 >= IF
    0 sec !
    1 minflag !                 \ min over!
    1 min +!
  ENDIF
  min @ 60 >= IF
    0 min !
  ENDIF
;
: led0 ( -- ) 3 port1 bclr ;
: led1 ( -- ) 3 port1 bset ;
: job.tick
  tick @ $01 and IF led0 ELSE led1 ENDIF
```

```
;
: job.sec
  timer @ dup lastsec @ - swap dup lastsec !
  sec @ min @
  cr . . . .
;
: job.min
  cr ." running minute job ..."
;
: init-loop ( -- )
  cr ." min sec timer timer-lastsec"
  timer @ cycles.tick + newtimer !
;
: run
  init-loop
  BEGIN
    tickover?  IF timeup ENDIF
   tickflag @ IF job.tick 0 tickflag ! ENDIF
    secflag @ IF job.sec  0 secflag !  ENDIF
    minflag @ IF job.min  0 minflag !  ENDIF
  key? UNTIL
;
```

```
\ 2006-08-16 EW adv2_2.fs
rom
include adv2_timeup.fs
: led0 ( -- ) 3 port1 bclr ;
: led1 ( -- ) 3 port1 bset ;
: job.tick ( -- )
  tick @ $01 and IF led0 ELSE led1 ENDIF
;
: job.sec ( -- )
  timer @ dup lastsec @ - swap dup lastsec !
  sec @ min @ hour @ day @ month @ year @
  cr . . . . . .
;
: job.min ( -- )
  cr ." running minute job ..."
;
: job.hour ( -- )
  cr ." running hour job ..."
;
: job.day ( -- )
  cr ." running day job ..."
;
: job.month ( -- )
  cr ." running month job ..."
;
: job.year ( -- )
  cr ." running year job ..."
;
: init-loop ( -- )
  cr ." year month day hour min sec timer"
  ." timer-lastsec"
  timer @ cycles.tick + newtimer !
  job.sec
;
: run
 init-loop
 BEGIN
  tickover? IF timeup ENDIF
  0 Flags btst IF job.tick 0 Flags bclr ENDIF
  1 Flags btst IF job.sec  1 Flags bclr ENDIF
  2 Flags btst IF job.min  2 Flags bclr ENDIF
  3 Flags btst IF job.hour 3 Flags bclr ENDIF
  4 Flags btst IF job.day  4 Flags bclr ENDIF
  5 Flags btst IF job.month 5 Flags bclr ENDIF
  6 Flags btst IF job.year 6 Flags bclr ENDIF
 key? UNTIL
;
ram
```

```
\ 2006-08-16 EW adv2_timeup.fs
Variable Flags
Variable newtimer
Variable lastsec
Variable tick
Variable sec
Variable min
Variable hour
Variable day
Variable month
Variable year
500 Constant cycles.tick \ timerC Zykl./Tick
2   Constant ticks.sec    \ Ticks/Sekunde
ram
create MaxDay 31 c, 28 c, 31 c, 30 c, 31 c,
  30 c, 31 c, 31 c, 30 c, 31 c, 30 c, 31 c,
rom
: tickover? ( -- ) newtimer @ timer @ -  0< ;
: leap_year ( year -- t/f )
  dup    4 mod 0=
  over 100 mod 0<> and
  swap 400 mod 0=  or
;
: length_of_month ( year month -- maxday )
  dup 1-                \ array starts at 0
  MaxDay + c@
  swap 2 = IF           \ if month == 2
    swap leap_year IF   \   if leap_year
      1+                \     month += 1
    ENDIF
  ELSE                  \ else
    swap drop           \   remove year
  ENDIF
;
: timeup ( -- )
  cycles.tick newtimer +!
  0 Flags bset          \ tick over
  1 tick +!
  tick @ ticks.sec >= IF
    0 tick !
    1 Flags bset        \ sec over
    1 sec +!
  ENDIF
  sec @ 60 >= IF
    0 sec !
    2 Flags bset        \ min over
    1 min +!
  ENDIF
  min @ 60 >= IF
    0 min !
    3 Flags bset        \ hour over
    1 hour +!
  ENDIF
  hour @ 24 >= IF
    0 hour !
    4 Flags bset        \ day over
    1 day +!
  ENDIF
  day @ year @ month @ length_of_month > IF
    1 day !             \ offset 1!
```

```
    5 Flags bset        \ month over
    1 month +!
  ENDIF
  month @ 12 > IF
    1 month !           \ offset 1!
    6 Flags bset        \ year over
    1 year +!
  ENDIF
;
```

```
\ 2006-08-18 EW adv2_3.fs
rom
include adv2_timeup.fs
: led0 ( -- ) 3 port1 bclr ;
: led1 ( -- ) 3 port1 bset ;
: job.tick ( -- )
  tick @ $01 and IF led0 ELSE led1 ENDIF
;
: job.sec ( -- )
  timer @ dup lastsec @ - swap dup lastsec !
  sec @ min @ hour @ day @ month @ year @
  cr . . . . . .
;
: job.min ( -- )
  cr ." running minute job ..."
;
: job.hour ( -- )
  cr ." running hour job ..."
;
: job.day ( -- )
  cr ." running day job ..."
;
: job.month ( -- )
  cr ." running month job ..."
;
: job.year ( -- )
  cr ." running year job ..."
;
: init-loop ( -- )
  cr ." year month day hour min sec timer"
  ." timer-lastsec"
  timer @ cycles.tick + newtimer !
  job.sec
;
ram
create Jobs ' job.tick ,
  ' job.sec , ' job.min , ' job.hour ,
  ' job.day , ' job.month , ' job.year ,
rom
: run
  init-loop
  BEGIN
    tickover? IF timeup ENDIF
    7 0 DO
      I Flags btst IF
      I cells Jobs + @ execute
      I Flags bclr
      ENDIF
    LOOP
  key? UNTIL
;
ram
```

```
\ 2006-08-16 EW adv2_4.fs
rom
include adv2_timeup.fs
include adv2_lcd.fs
: show.DT ( -- )
  year  @ p4! lcdtype
  month @ p2! lcdtype
  day   @ p2! lcdtype s" -" lcdtype
  hour  @ p2! lcdtype
  min   @ p2! lcdtype
  sec   @ p2! lcdtype
;
0     Constant PinSCL
1     Constant PinSDA
port1 Constant PortI2C
$E3   Constant PddrI2C
$9e   Constant i2c_addr_lm75
$a0   Constant i2c_addr_rtc
include adv2_i2c.fs
include adv2_i2c_rtc.fs
: clock-init ( -- )
  get.rtc
  year !
  bcd>dec month !
  bcd>dec day !
  bcd>dec hour !
  bcd>dec min !
  bcd>dec sec !
  drop \ Sec/100
;
include adv2_i2c_lm75.fs
: led0 ( -- ) 3 port1 bclr ;
: led1 ( -- ) 3 port1 bset ;
: job.tick
  tick @ $01 and IF led0 ELSE led1 ENDIF
  1 0 lcdpos show.T
;
: job.sec
  0 13 lcdpos sec @ p2! lcdtype
  timer @ dup lastsec @ - swap dup lastsec !
  sec @ min @ hour @ day @ month @ year @
  cr . . . . . .
;
: job.min
  cr ." running minute job ..."
  0 0 lcdpos show.DT
;
: job.hour
  cr ." running hour job ..."
;
: job.day
  cr ." running day job ..."
;
: job.month
  cr ." running month job ..."
;
: job.year
  cr ." running year job ..."
;
: init-loop ( -- )
  cr ." year month day hour min sec timer"
  ." timer-lastsec"
  clock-init
  timer @ cycles.tick + newtimer !
  lcdpage show.DT
  job.sec
;
ram
create Jobs ' job.tick ,
```

```
    ' job.sec , ' job.min , ' job.hour ,
    ' job.day , ' job.month , ' job.year ,
rom
: run
  init-loop
  BEGIN
    tickover? IF timeup ENDIF
    7 0 DO
      I Flags btst IF
      I cells Jobs + @ execute
      I Flags bclr
      ENDIF
    LOOP
  key? UNTIL
;
ram
```

```
\ 2006-08-15 EW adv2_lcd.fs
\ always display a sign
: sign! 0 < IF 45 ELSE 43 ENDIF hold ;
\ always display 4,2 digits
: p4!  s>d <# # # # # #> ;
: p2!  s>d <# # # #> ;
\ start next lcdtype at lcdposition
\ row (0,1) col (0..15)
: lcdpos ( row col -- )
  swap $40 * + $80 + lcdctrl! &1 ms
;
```

```
\ 2006-07-09 EW  i2c bus master
\  No Error Checking whatsoever !
\
\ expects:
\  PinSDA PinSCL
\  PortI2C PddrI2C
\ provides:
\   NB>i2c ( xN .. x1.msB N addr -- )
\     send N bytes to i2c device at addr
\   NB<i2c ( N addr -- X1.msB .. xN )
\     read N bytes from i2c device at addr
\     expects previously sent addr/controlbyte
\     since REPEATED START condition is used

: sda0 PinSDA PortI2C bclr ;
: sda1 PinSDA PortI2C bset ;
: scl0 PinSCL PortI2C bclr ;
: scl1 PinSCL PortI2C bset ;
\ get Bit Nr. i from Byte x
: getBit ( x i -- b ) rshift 1 and ;
: sdaInput  ( -- ) PinSDA PddrI2C bclr ;
: sdaOutput ( -- ) PinSDA PddrI2C bset ;
: readSDA   ( -- f )
  PinSDA PortI2C btst IF 1 ELSE 0 ENDIF ;
: i2c_tick 2 us ;
: 2i2c_tick i2c_tick i2c_tick ;
\ send START
: i2c_start ( -- )
  i2c_tick sda0 2i2c_tick scl0 i2c_tick ;
\ send STOP
: i2c_stop  ( -- )
  i2c_tick scl1 2i2c_tick sda1 i2c_tick ;
\ send REPEATED START
: i2c_rstart
  sda1 i2c_tick scl1 i2c_tick
  sda0 i2c_tick scl0 i2c_tick
;
```

```
\ clock out 1 Bit
: bit>i2c ( f -- )
  IF sda1 ELSE sda0 ENDIF
  i2c_tick scl1 2i2c_tick scl0 i2c_tick
;
\ send 1 Byte, 8 Bit, msb first
: >i2c ( x -- )
  8 0 DO
    dup 8 I 1+ - getBit
    bit>i2c
  LOOP
  drop
;
\ read 1 Byte, 8 Bit, msb first
: <i2c ( -- x )
  sdaInput
  0
  8 0 DO
    1 lshift
    i2c_tick scl1 i2c_tick readSDA
    i2c_tick scl0 i2c_tick
    +
  LOOP
  sdaOutput
;
\ read ACK|NACK from device
: ack<i2c ( -- f )
  sdaInput
  i2c_tick scl1 i2c_tick readSDA
  i2c_tick scl0 i2c_tick
  sdaOutput
;
\ send n bytes from stack to device
: NB>i2c ( x1 .. xN.msB N addr -- )
  i2c_start
  >i2c ack<i2c drop   \ send address
  0 DO
    >i2c ack<i2c drop \ send next byte
  LOOP
  i2c_stop
;
\ read n Bytes from device to stack, assume
\ address and opt. controlbytes were sent
: NB<i2c ( N addr -- xN.msB .. x1 )
  i2c_rstart
  1+ >i2c ack<i2c drop \ send addr
  1- dup 0 > IF
    0 DO \ loop N-1 times
      <i2c 0 bit>i2c \ read byte, send ACK
    LOOP
  ENDIF
  <i2c 1 bit>i2c \ read last byte, send NACK
  i2c_stop
;
```

```
\ 2006-07-25 EW adv2_i2c_rtc.fs
\ pcf8583:
\ addr
\ 0x00    control register 3: mask_flag
\ 0x01    sec/100.bcd
\ 0x02    sec.bcd
\ 0x03    min.bcd
\ 0x04    7: 0=24h clock, 6: am/pm flag
\         5-0: hour.bcd
\ 0x05    7-6: year%4 5-0: day.bcd
\ 0x06    7-5: weekdays unless maskbit,
```

```
\         4-0: month.bcd
\ eeprom:
\ 0x10,0x11 full year, not BCD

: bcd>dec ( n.bcd -- n.dec )
  $ff and
  dup
  4 rshift 10 * \ extract high nibble as 10s
  swap
  $0f and       \ extract low  nibble as 1s
  +             \ add
;
: dec>bcd ( n.dec -- n.bcd )
  &100 mod      \ 99 is largest for 8 bit bcd
  &10 /mod
  4 lshift
  +
  $ff and       \ truncate to 8 bit
;
: get.rtc
  $01                   \ start address
  1 i2c_addr_rtc NB>i2c \ send > rtc
  6 i2c_addr_rtc NB<i2c \ read 6 Bytes
  $10                   \ YEAR's address
  1 i2c_addr_rtc NB>i2c \ send > rtc
  2 i2c_addr_rtc NB<i2c \ read 2 Bytes
  8 lshift +            \ convert to YYYY
;
\ convert human readable numbers to expected
\ BCD numbers and compounds
: format.rtc
  \ year month day hour min sec sec/100 --
  \ year weekday|month.bcd year%4|day.bcd
  \ hour.bcd min.bcd sec.bcd sec/100.bcd
  dec>bcd >R \ sec/100.bcd
  dec>bcd >R \ sec.bcd
  dec>bcd >R \ min.bcd
  dec>bcd >R \ hour.bcd
             \ year%4<<6 | hour.bcd
  dec>bcd 2 pick $03 and 6 lshift + >R
  dec>bcd    \ month.bcd
  R> R> R> R> R>
;
: set.rtc ( year ... sec/100.bcd -- )
  format.rtc
  $80 $00               \ stop rtc
  8 i2c_addr_rtc NB>i2c \ send all args
                        \ start rtc
  $08 $00 2 i2c_addr_rtc NB>i2c

  \ year is still on stack
  dup $00ff and         \ low byte
  swap
  8 rshift $00ff and    \ high byte
  swap                  \ low byte first
  $10                   \ start addr
  3 i2c_addr_rtc NB>i2c \ send > rtc
;
```

```
\ 2006-07-25 EW adv2_i2c_lm75.fs
\ read temp sensor, controlbyte: 0x00
: get.T ( -- xh xl )
  0 1 i2c_addr_lm75 NB>i2c
  2 i2c_addr_lm75 NB<i2c
;
\ bake 2 Bytes into T scaled 10
\ "*/" takes care of the sign :-)
: decode.T ( xh xl -- T*10 )
  $80 and     \ ( xl & 0x80 )
  swap
  8 lshift    \ ( xh << 8 )
  +           \ ( xh<<8 + (xl & 0x80) )
  5 128 */    \ *10 / 256 ( scale to 1/10 C )
;
\ format T for display (type or lcdtype)
: format.T ( T*10 -- )
  dup >R      \ <# expects unsigned double,
  abs s>d     \ so create one
  <# # 46 hold # # R> sign! #>  \ %+5.1f
  \ get sign back from return stack
;
\ assume lcdpos has been issued before
: show.T ( -- )
  get.T decode.T format.T lcdtype
;
```

```
\ 2006-08-15 EW adv2_5.fs
rom
Variable Flags
Variable newtimer
Variable lastsec
500 Constant cycles.tick \ timerC cycles/tick
2   Constant ticks.sec   \ ticks/second
ram
create Counts &7 cells allot
create Limits ticks.sec c, 60 c, 60 c,
  24 c, 31 c, 12 c,
create MaxDay
  31 c, 28 c, 31 c, 30 c, 31 c, 30 c,
  31 c, 31 c, 30 c, 31 c, 30 c, 31 c,
rom
: tick  ( -- addr )         Counts   ;
: sec   ( -- addr ) 1 cells Counts + ;
: min   ( -- addr ) 2 cells Counts + ;
: hour  ( -- addr ) 3 cells Counts + ;
: day   ( -- addr ) 4 cells Counts + ;
: month ( -- addr ) 5 cells Counts + ;
: year  ( -- addr ) 6 cells Counts + ;
: tickover? ( -- ) newtimer @ timer @ -  0< ;
: leap_year ( year -- t/f )
  dup    4 mod 0=
  over 100 mod 0<> and
  swap 400 mod 0=  or
;
: length_of_month ( year month -- maxday )
  dup 1-                \ array starts at 0
  MaxDay + c@
  swap 2 = IF           \ if month == 2
    swap leap_year IF   \ and leap_year
      1+                \   month += 1
    ENDIF
  ELSE
    swap drop           \ remove year
  ENDIF
;
: timeup ( -- )
  cycles.tick newtimer +!
  0 Flags bset                  \ tickflag++
  1         Counts   +!         \ tick++
  \ for i in tick sec min hour day month
  6 0 DO
    I cells Counts +  @
```

```
    I Limits + c@ >= IF    \ if C[i] >= L[i]
      0 I cells Counts + !     \   C[i]=0;
      I 1+ Flags bset          \   F[i+1]++;
      1 I 1+ cells Counts + +! \   C[i+1]++;
    ENDIF                      \ endif
  LOOP
;
include adv2_lcd.fs
: show.DT ( -- )
  year  @ p4! lcdtype
  month @ 1+ p2! lcdtype
  day   @ 1+ p2! lcdtype s" -" lcdtype
  hour  @ p2! lcdtype
  min   @ p2! lcdtype
  sec   @ p2! lcdtype
;
0     Constant PinSCL
1     Constant PinSDA
port1 Constant PortI2C
$E3   Constant PddrI2C
$9e   Constant i2c_addr_lm75
$a0   Constant i2c_addr_rtc
include adv2_i2c.fs
include adv2_i2c_rtc.fs
: clock-init ( -- )
  get.rtc
  year !
  bcd>dec 1- month ! \ offset 1 correction!
  bcd>dec 1- day !   \ offset 1 correction!
  bcd>dec hour !
  bcd>dec min !
  bcd>dec sec !
  drop \  Sec/100
;
include adv2_i2c_lm75.fs
: led0 ( -- ) 3 port1 bclr ;
: led1 ( -- ) 3 port1 bset ;
: job.tick
  tick @ $01 and IF led0 ELSE led1 ENDIF
  1 0 lcdpos show.T
;
: job.sec
  0 13 lcdpos sec @ p2! lcdtype
  timer @ dup lastsec @ - swap dup lastsec !
  7 1 DO I cells Counts + @ LOOP
  cr . 1+ . \ month offset 1 correction
  1+ .      \ day   offset 1 correction
  . . . . .
;
: job.min
  cr ." running minute job ..."
  0 0 lcdpos show.DT
;
: job.hour
  cr ." running hour job ..."
;
: job.day
  cr ." running day job ..."
;
: job.month
  cr ." running month job ... "
  year @
  month @ 1+ \ month offset 1 correction
  length_of_month
  dup . ." new month limit"
  4 Limits + c!
;
: job.year
  cr ." running year job ..."
;
: init-loop
  cr ." year month day hour min sec timer"
  ." timer-lastsec"
  clock-init
  timer @ cycles.tick + newtimer !
  lcdpage show.DT
  job.sec
;
ram
create Jobs ' job.tick ,
  ' job.sec , ' job.min , ' job.hour ,
  ' job.day , ' job.month , ' job.year ,
rom
: run
  init-loop
  BEGIN
    tickover? IF timeup ENDIF
    7 0 DO
      I Flags btst IF
      I cells Jobs + @ execute
      I Flags bclr
      ENDIF
    LOOP
  key? UNTIL
;
ram
```

```
\ gforth/arch/r8c/tasker.fs
Variable bgtask ram $20 cells allot rom
:noname  bgtask @ 0= ?EXIT
  rp@ bgtask @ sp@ cell+ bgtask ! sp! rp! ;
IS pause
: task r> bgtask $20 cells + !
  bgtask $20 cells + bgtask $10 cells + !
  bgtask $10 cells + bgtask !
;
:noname echo @ IF
    BEGIN pause key? UNTIL THEN (key) ;
is key
```

Beispiele für Gforth–R8C

Bernd Paysan

Auf der Wiki–Seite für das Gforth–R8C haben sich in den letzten paar Monaten einige Beispiele zusammengefunden, die wir hier abdrucken.

Lauflicht im R8C

Ein einfaches Lauflicht, das nach dem Reset gleich losläuft, bis man über's Terminal eine Taste drückt:

```
rom
: licht!  led!  &100 ms ;
: lauf  1 licht! 2 licht! 4 licht! 8 licht!
  4 licht! 2 licht! ;
: dauerlauf  BEGIN  lauf  key?  UNTIL ;
' dauerlauf is bootmessage
ram
savesystem
```

ADC–Wandler im R8C

Eine Schleife, die vom ADC liest und auf dem LCD ausgibt:

```
: adcmeter
  BEGIN  6 adc@ 0 <# #s #> lcdpage lcdtype
         &200 ms  key? UNTIL ;
```

Multitasker

Und hier gibt's einen ganz einfachen Multitasker, der genau *einen* Hintergrundtask erlaubt:

```
rom

Variable bgtask ram $20 cells allot rom
:noname  bgtask @ 0= ?EXIT
    rp@ bgtask @ sp@ cell+ bgtask ! sp! rp! ;
IS pause
: task r> bgtask $20 cells + !
  bgtask $20 cells + bgtask $10 cells + !
  bgtask $10 cells + bgtask ! ;
:noname echo @ IF
     BEGIN pause key? UNTIL THEN (key) ;
is key

ram
```

Als Beispiel eine adaptierte Version des Lauflichts von oben:

```
rom
: licht!  led!  &100 ms ;
: lauf  1 licht! 2 licht! 4 licht! 8 licht!
  4 licht! 2 licht! ;
: dauerlauf
  task  BEGIN  lauf  AGAIN ;
ram
```

Lauftext

Analog zum Lauflicht kann man auch einen Lauftext erzeugen — und über das Poti die Textgeschwindigkeit einstellen:

```
rom

Create text
," GNU Forth EC R8C -- Mikroprozessor -- "
Create ledtable 1 c, 2 c, 4 c, 8 c, 4 c, 2 c,
Variable /text

: lauftext  task
  BEGIN  text count /text @ over mod /string
         16 min dup >r lcdpage lcdtype
         r@ 16 < IF  text 1+ 16 r@ - lcdtype
         THEN
         rdrop 1 /text +!
         /text @ 6 mod ledtable + c@ led!
         6 adc@ 2/ ms
  AGAIN ;

ram
```

Lebenszeichen

Berichte aus der FIG Silicon Valley: *Henry Vinerts*

Dear Fred,

schön, von dir zu hören. Ja, ich erinnere mich an unsere Diskussionen über „verar...“ (Übers.: Ausdruck, den ein Autor unter Missachtung der FG-Gepflogenheiten in einem VD–Heft verwendet hat).

Mein Wörterbuch übersetzt „foppen“ mit „to pull one's leg“, ein englischer oder amerikanischer (?) Slang–Ausdruck. Eine andere Übersetzung desselben Zeitwortes ist „to make fun of somebody“. Die letztgenannte Übersetzung erscheint auch beim Wort „veralbern“.

Ich glaube, ich verstehe nicht ganz, was „zum Besten halten“ bedeutet (Übers.: Kann jemand meine Erklärungsversuche ergänzen und Henry helfen?).

Im Moment kann ich die alten E–Mails nicht finden, in denen wir über dieses Thema sprachen. Ich denke immer noch, dass die beste Übersetzung für „verar...“ der Ausdruck „to mess up“ ist, oder eine sehr ähnliche vulgäre Wendung, die ich unter meinem Namen nicht in Druck gehen lassen möchte.

Zum Thema NXT und LEGO...

Wie ich sehe, gibt es in der deutschen Forth–Gesellschaft Leute, die gern wieder Forth in den LEGO–Markt hineinzwängen möchten. Ist das wirklich die Mühe wert? Noch ehe wir uns versehen, wird der Deckmantel des Überholtseins alles verschwinden lassen, was eifrige Forthler geschaffen haben. Es ist sinnlos, gegen den Strom zu schwimmen.

Letzten Samstag habe ich es wieder mal zum SVFIG–Treffen geschafft und war überrascht, 15 oder 16 Anwesende zu zählen. Fünf oder sechs davon waren neu oder mir unbekannt. Dazu gehörten auch die beiden Zeitgenossen, die eine nicht gerade kurze Darstellung von SEA-Forth präsentierten, vom jüngsten Produkt von Chuck Moores eigener Firma, IntellaSys. Wenn du davon noch nichts gehört haben solltest, müssten sich inzwischen im Web genügend Informationen darüber finden lassen. (Gib in Google einfach die Schlüsselworte ein.) Jeder bekam eine CD mit einem Demo in Swift Forth oder G–Forth. Die Firma setzt sich aus einer guten Zahl von altbekannten Forthlern zusammen und versprüht beträchtlichen Enthusiasmus über zukünftige Projekte, besonders auch weil genügend Geld vorhanden zu sein scheint, um Entwicklungen voranzutreiben.

Ich will nicht vergessen zu erwähnen, dass Ting das Treffen mit seinem traditionellen Zweistundenvortrag über eines seiner Projekte eröffnete. Diesmal waren es die HMPP (Hyper Massively Parallel Processors), die alle auf einem einzigen Chip sitzen und die alle ein und denselben Befehl zur selben Zeit, aber von verschiedenen Quellen her kommend ausführen. Ting hatte eine Arbeitsdemo für 512 parallele Prozessoren vorbereitet und mitgebracht. Es spricht aber nichts dagegen, ein vollausgewachsenes HMPP–System mit 1,3 Millionen Prozessoren zusammenzustellen, mit Ausnahme vielleicht von fehlendem Geld. Und natürlich hat Ting, wie üblich, nicht vergessen, bei seinen Schöpfungen Forth und Chucks Prozessoren zu verwenden.

Die Tagungsliste für das SVFIG–Oktobertreffen (`http://www.forth.org`) enthält einen Vortrag, den jemand über Forth in den Schulen, und speziell über LEGO einschließlich NXT, halten will. Ich werde das Treffen wahrscheinlich besuchen, auch wenn ich dann wieder Tings Zweistundenvorlesungen über technische

Einige Teilnehmer des SVFIG–Forth–Treffens im November 2006 — Henry mit Swapshirt

Dinge über mich ergehen lassen muss, die ich meistens so gut wie gar nicht verstehe.

Wishing you well in all of your endeavors,
Henry Übersetzt von Fred Behringer

Dear Fred,

ich möchte dich nur schnell wissen lassen, dass ich deine Nachricht an Prof. Haydon gestern weitergeleitet habe. Ich ging zum SVFIG–Treffen um 11.00 Uhr und kam gerade zum zweiten Teil eines Vortrages von Prof. Ting über eine von ihm konstruierte Suchmaschine zurecht. Es waren, mich eingerechnet, etwa 9 Zuhörer. Als ich am Nachmittag gerade im Begriff war wegzugehen, kam Glen Haydon. Ich sagte ihm, dass du über den Verbleib seines VD–3/2006–Exemplars Nachforschungen angestellt hast und dass du ihm nachträglich alles Gute zu seinem „runden“ Geburtstag wünschst. Er bat mich, dir zu danken, und wünscht dir seinerseits Frohe Weihnachten.

Er beschäftigt sich aktiv wie immer und munter mit Dingen, die ihn interessieren. Was ihn stört ist nur, dass er nicht an zwei Orten gleichzeitig sein kann. Er hatte am Vormittag ein Radio–Amateur–Treffen besucht und konnte uns berichten, dass Morse–Code für eine aktive Teilhabe am Amateur–Radio–Leben nicht mehr unbedingt nötig sei.

Ich verließ das Treffen noch bevor die Nachmittag–Sitzung begann, besuchte ein Museum am Ort und sah mir eine Ausstellung von Lego–Zügen an, die das Museum für die Weihnachtszeit aufgebaut hat. Ich versuche, den Kontakt zu den Lego–Gruppen zu halten, um herauszubekommen, was sich, wenn überhaupt etwas, in Richtung der NXT–Baukästen tut.

Ja, noch was: Ich sagte Prof. Haydon, er möge das VD–Heft 4/2006 nicht vor Januar erwarten.

Ich danke dir für dein Interesse und deine freundliche Unterstützung.

Henry Übersetzt von Fred Behringer

Leserbriefe und Meldungen (Fortsetzung von Seite 8)

Kannst du nicht mal eben. . . AVR–Butterfly

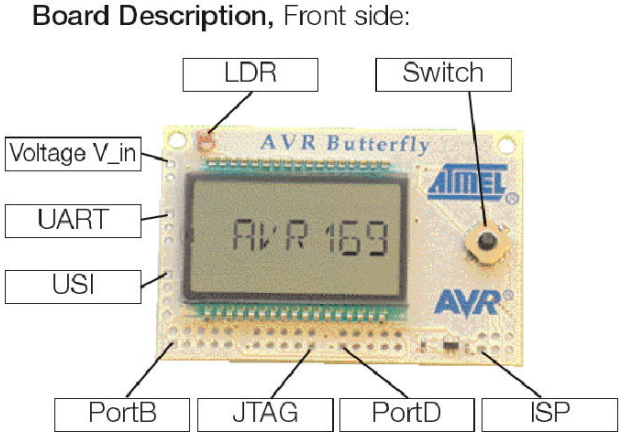
Kürzlich wurde deutlich, dass es ganz nützlich sein könnte, für die weiteren Überlegungen zur Erneuerung meiner Heizung im Haus Daten darüber zu haben, welche Wärme wann eigentlich wohin geht. Ein Daten–Logger sollte her, der einige Temperaturen aufzeichnet. Ein billiges handliches Kärtchen mit viel nichtflüchtigem Speicher, Display und der Möglichkeit, die aufgenommenen Rohdaten an einen PC zu geben, sollte her.

The AVR Butterfly Includes: ATmega169 AVR microcontroller, 100 segment LCD Display, 4Mbit Dataflash, Light sensor (LDR), Temperature Sensor (NTC), Speaker for Sound Generation, Access to pheripherals through header connectors, RS–232 Level Converter. Also eigentlich alles da. Die Möglichkeit, die bereits mitgelieferte Firmware passend *aufzubohren*, um die freien vier ADC Kanäle mit Temperatursensoren (NTC) zu belegen, war verlockend. Ein bisschen C–Code fügte das Erforderliche ein; es ging wirklich fast wie *mal eben.* — A. Krüger teilt Ihnen gerne mit, wie. Nun loggert die Karte seit einigen Tagen still vor sich hin in meinem Keller. Ab und zu wird das Dataflash ausgelesen. Die Datensätze lassen sich als Textdatei in MS–Excel importieren, dort aufbereiten und als Kurvenzüge sichtbar machen. So wächst nun ein Datenbestand heran, der mir hoffentlich mehr Aufschluss darüber gibt, was da in der Heizungsanlage eigentlich läuft. Außerdem weiß ich nun, dass unser Kühlschrank in der Küche nachts heimlich sein Licht innen anmacht — zum Justieren der Temperatur lag der Logger natürlich auch mal darin. Und siehe da: dank seines LDR auf der Karte, die gleich mitgeloggert wird, konnte man das schön sehen. Weitere Informationen zum AVR–Butterfly findet man unter `www.atmel.com`.

Nachtrag

Vermisst habe ich natürlich ein handliches Forth für den AVR–Butterfly. FORTH Inc. soll wohl ein Forth für die Prozessorgruppe bieten. Doch es gibt auch schon ein pures 16Bit–Forth für die AVR–ATmega–Mikro–Controller–Gruppe: Matthias Trute hat es gerade (2006-12-17) als *amforth 0.9* herausgegeben, es unter GNU General Public License (GPL) gestellt und nun auch in Aussicht gestellt, den *new controller: atmega169 (atmel butterfly)* zu unterstützen. Bisher galt für sein Forth jedoch *the main development uses ATmega8/32* — Lieber Matthias, ich hoffe, es kommt bald. Oder kriegen wir nach dem R8C nun auch das gforth für den Schmetterling?

M. Kalus



Das AVR–Butterfly–Board

weitere Leserbriefe und Meldungen auf Seite 24

Gedankenlesen

Michael Kalus

Einige kennen diese Spielerei möglicherweise schon. Aber ich war neulich erstmal wieder verblüfft über den Effekt. Und musste noch mal genauer hinsehen, wie das eigentlich funktioniert. Da wird man aufgefordert, sich eine zweistellige Zahl zu denken, deren Quersumme davon abzuziehen und das Ergebnis als Index in eine Symboltabelle zu benutzen. Und nun intensiv dieses Symbol anzuschauen, damit der Computer es erspüren könne! Und siehe da, es klappt tatsächlich. Ja, warum klappt das??

Deswegen:

```
: ?? ( -- )
  100 10 DO cr
  i .
  i i 10 /mod + - .
  LOOP ;
```

Es ist eigentlich sehr einfach. Hier der Beweis:

```
Gedankenlesen in Forth

Denke an eine zweistellige Zahl, z.B. die 54.
Ziehe von dieser Zahl ihre 2 Ziffern ab (54-5-4=45)
Suche das Symbol, das zu diesem Ergebnis passt.
Konzentriere dich auf dieses Symbol und druecke eine Taste...

 0 P   1 g   2 f   3 ‘   4 V   5 w   6 [   7 _   8 c   9 c
10 W  11 }  12 E  13 <  14 e  15 k  16 h  17 ^  18 c  19 x
20 k  21 R  22 ]  23 l  24 o  25 f  26 d  27 c  28 V  29 z
30 Q  31 c  32 t  33 _  34 c  35 C  36 c  37 O  38 b  39 P
40 {  41 r  42 J  43 W  44 u  45 c  46 u  47 j  48 y  49 V
50 [  51 x  52 v  53 H  54 c  55 j  56 a  57 g  58 W  59 K
60 i  61 z  62 a  63 c  64 m  65 b  66 ‘  67 x  68 |  69 t
70 @  71 <  72 c  73 n  74 <  75 i  76 O  77 f  78 j  79 l
80 W  81 c  82 \  83 ^  84 }  85 @  86 \  87 J  88 k  89 h
90 R  91 ^  92 s  93 V  94 f  95 j  96 p  97 R  98 a  99 N

Es ist:  c

Noch mal? --> Leertaste!
```

Viel Vergnügen beim Nachrätseln, oder damit andere zu verblüffen.

Links

http://www.isnichwahr.de/redirect15446.html
http://www.extremefunnyhumor.com/magic_mindreader.htm

Listing

```
1   \ Gedankenlesen
2
3   \ Prinzip:
4   \ Die Zahlen 9, 18, 27, 36, 45, 54, 63, 72, 81
5   \ bekommen das gleiche Symbol zugeordnet.
6   \ Alle übrigen erhalten ein Symbol random zugeordnet,
7   \ es muss nur druckbar sein.
8
9   Decimal
10
11  \ include random.fs
12  ( gforth-0.6.2 Verzeichnis; Mac OSX Version)
13
14  Variable seed
15
16  $10450405 Constant generator
17
18  : rnd  ( -- n )
19    seed @ generator um* drop 1+ dup seed ! ;
20
21  : random ( n -- 0..n-1 )
22    rnd um* nip ;  \ \include
23
24  : init ( -- )
25    5 seed ! 10 0 DO 100 random drop LOOP ;
26
27    \ choose a token, but no digit.
28  : token ( -- x )  126 58 - random 58 + ;
29
30    \ test number of tokens
31  : tt ( -- )  $FF 0 DO I emit I . LOOP ;
32
33  variable symbol
34
35  : symbol!  ( -- )  token symbol ! ;
36
37  : sy. ( -- ) symbol @ emit ;
38
39  : symbol. ( I -- )
40    DUP 09 = IF sy. drop exit THEN
41    DUP 18 = IF sy. drop exit THEN
42    DUP 27 = IF sy. drop exit THEN
43    DUP 36 = IF sy. drop exit THEN
44    DUP 45 = IF sy. drop exit THEN
45    DUP 54 = IF sy. drop exit THEN
46    DUP 63 = IF sy. drop exit THEN
47    DUP 72 = IF sy. drop exit THEN
48    DUP 81 = IF sy. drop exit THEN
49    token emit drop ;
50
51  : tabelle. ( -- )
52    cr 100 0 DO
53    I 2 .r space
54    I symbol. 2 spaces
55    I 1+ 10 /mod drop 0= IF cr THEN
56    LOOP ;
57
58  \ : page ( -- ) <your own code> ; \ gforth has this comand
59  \   page clears the display.
60
61  : text.de ( -- )
62    cr ." Gedankenlesen in Forth"
63    cr
64    cr ." Denke an eine zweistellige Zahl, z.B. die 54."
65    cr ." Ziehe von dieser Zahl ihre 2 Ziffern ab "
66       ." (54-5-4=45)"
67    cr ." Suche das Symbol, das zu diesem Ergebnis passt."
68    cr ." Konzentriere dich auf dieses Symbol "
69       ." und druecke eine Taste..." ;
70
71  : ?again.de ( -- f )
72    cr ." Noch mal? --> Leertaste! " key $20 = ;
```

```
73
74
75   : text.en ( --)
76     cr ." MindReader in Forth"
77     cr
78     cr ." Think of a number with 2 digits (ex: 54)"
79     cr ." Subtract from this number its 2 digits "
80        ." (54-5-4=45)"
81     cr ." Find the symbol that corresponds "
82        ." to this number"
83     cr ." Concentrate on the symbol "
84        ." and press a key..." ;
85
86   : ?again.en ( -- f )
87     cr ." Again? --> press space bar! " key $20 = ;
88
89
90     \ german version
91   : Gedankenlesen ( -- )
92     init
93     BEGIN symbol!
94     page text.de cr Tabelle. key drop
95     cr ." Es ist:  " sy. cr
96     ?again.de 0= UNTIL ;
97
98     \ english version
99   : MindReader  ( -- )
100    init
101    BEGIN symbol!
102    page text.en cr Tabelle. key drop
103    cr ." It is:  " sy. cr
104    ?again.en 0= UNTIL ;
105
106    \ Des Rätsels Lösung
107  : ?? ( -- )
108    100 10 DO cr
109    i .
110    i i 10 /mod + - .
111    LOOP ;
112
113  \ fin
```

Leserbriefe und Meldungen (Fortsetzung von Seite 22)

MultiCore–Mikro–Controller von Parallax

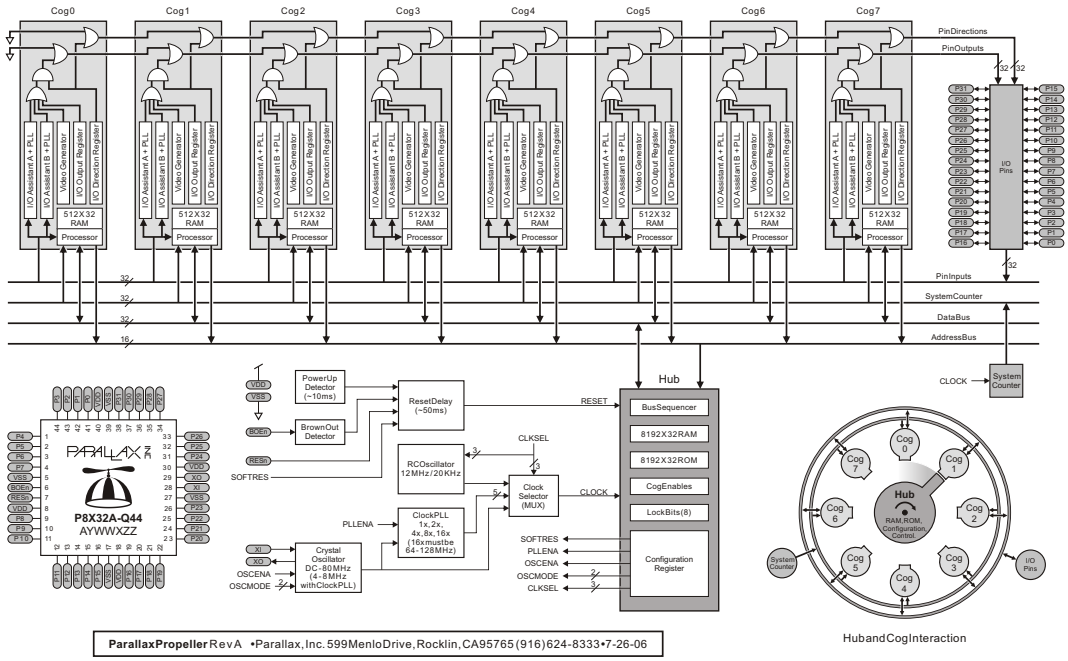
Parallax Inc., bekannt durch die PIC–basierte BASIC–Stamp (Briefmarke), hat einen Mikro–Controller entwickelt, der *Propeller* genannt wird. Er enthält acht (8!) 32Bit–Prozessorkerne, sog. *COGs*. Diese Kerne haben jeweils 2 KB eigenen RAM–Speicher und können auf einen gemeinsamen 32KB/32KB großen RAM/ROM–Bereich zugreifen. Getaktet wird der Propeller mit bis zu 80Mhz. 32 I/O–Pins mit je 40mA können verwendet werden. Einzelchips kosten ca. 17,– €, Entwicklungsboards gibt es ab 80,– € zu haben.

Der Propeller wird in Assembler oder in der von Parallax entwickelten Programmiersprache *Spin* programmiert. Spin stellt vordefinierte „Objekte“ für die Unterstützung von Video, Maus, Tastatur, Funk, LCD–Anzeigen, Schrittmotoren und Sensoren zur Verfügung, die in einer Spin–Applikation geeignet integriert werden. Da das Programmieren eines echten Multiprozessor–Systems wie dem Propeller durchaus eine Herausforderung darstellt, empfiehlt Parallax den Einsatz nur bei schon bestehender Erfahrung mit Mikro-Controllern... Wär das nicht eine spannende Zielplattform für ein Multiprozessor–Forth? Wie ging das noch auf dem Transputer...

Weitere Informationen finden sich unter
`www.parallax.com/propeller`

uho

weitere Leserbriefe und Meldungen auf Seite 26



Gehaltvolles

zusammengestellt und übertragen von *Fred Behringer*

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 57, August 2006

Timers in Forth deel 1 — Ron Minke

Übersetzung des Vorspanns: Die Verwendung eines Timers ist in Forth nicht speziell festgelegt. Im ANSI–Forth–Standard kommt das Wort MS vor, und zwar im Facility Extension Word Set. Mit diesem Wort erzeugt man Verzögerungen von einigen Millisekunden. MS wartet einfach so lange, bis die vorgegebene Zeit verstrichen ist. Es gibt keine Möglichkeit, zwischendurch noch etwas anderes zu tun. Was müssen wir uns ausdenken, wenn wir zwischendurch doch etwas anderes ausführen wollen und keine Lust haben, die Wartezeit sinnlos verstreichen zu lassen?

Der Autor (seit Januar 2006 Redakteur des Vijgeblaadjes) bespricht das Problem für Prozessoren der Reihe 8051. Er geht (wie bei der 8051–Serie üblich) davon aus, dass die ROM– und RAM–Bereiche sich teilweise überlappen. Genauer gesagt, sollen sich die obersten 32 Kilobyte überlappen. Ein weiterer Artikel–Teil soll folgen.

SEAforth–24, een nieuwe Forth processor — Willem Ouwerkerk

Charles Moore wartet mit einem neuen Chip auf (wegen dieser Meldung siehe auch VD 3/2006). 24 Prozessoren auf einem einzigen Chip, mit 18–Bit–Wortbreite, Stack und ROM und RAM. Alle 24 Prozessoren können untereinander Daten und Programm–Code auswechseln. 11 schnelle E/A–Ports für die Kopplung. 8 aktive Prozessoren verbrauchen etwa 150 mW. Ein passendes Forth hat er sich dazu auch ausgedacht: VentureForth. 32 Opcodes. Jede Nanosekunde wird ein Forth–Wort ausgeführt. Der Berichterstatter zählt im Übrigen 15 Haupteigenschaften auf. Für was sind die Chips vorgesehen? 144–Pin–BGA–Sockel, 10 US–Dollar pro Stück (bei 1000–Stück–Kauf). Simulator T18. `http://www.intellasys.net/` . (Der Rezensent: Ist der überwältigende Aufwand, den INMOS seinerzeit mit dem Transputer getrieben hat, berücksichtigt worden?)

Gespot op het web — Frans van der Markt

Übersetzung der ganzen zehn Zeilen des aus dem Netz Gefischten: Unsere Schwesterorganisation Forth–Gesellschaft e.V. hat nahezu das gesamte Archiv der Zeitschrift Vierte Dimension ins Internet gestellt. Wenn Sie der deutschen Sprache einigermaßen mächtig sind, ist das ein Schatz an Informationen über Forth. Das Archiv läuft von 1984 an bis heute und umfasst somit einen beträchtlichen Teil der Entwicklung von Forth. Zugleich existiert auch eine Liste aller publizierten Artikel. Man kann sich die Hefte in Form von PDF–Dateien herunterladen von der Site:

`http://www.forth-ev.de/filemgmt/viewcat.php?cid=2`

Viel Spaß beim Lesen! (Ende der Übersetzung)

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 58, Oktober 2006

Timers in Forth deel 2 — Ron Minke

Übersetzung des Vorspanns: In diesem zweiten Teil wird die Timer-Struktur von der Forth-Seite her betrachtet. Im ersten Teil hatten wir erwähnt, dass wir dieselben Adressen für die Byte-Plätze für den Timer selbst und für die Bit-Plätze für die zugehörigen Steuerbits verwenden wollen. Beim Definieren eines neuen Timers müssen wir die 8052-Timer-Interrupt-Routine erweitern. Neben der Anpassung von Forth kommen wir nicht umhin, auch ein Stückchen Maschinencode für die Behandlung des Low-Level-Timers vorzusehen. (Ende des Vorspanns).

Dieses ist der zweite Teil des Timer-Artikels von Ron Minke für Prozessoren der Reihe 8051. Der Artikel ist in fünf Abschnitte unterteilt:

1. Bits und Bytes verwenden
2. Was tut dieser (hier angegebene) Code?
3. Der Timer ist abgelaufen, was nun?
4. Die Timer-Struktur
5. Der (Forth-)Quelltext

Breinbreker — Frans van der Markt

Unsere holländischen Forth-Nachbarn haben Sudoku für das Vijgeblaadje entdeckt. Der Autor nennt es Alphadoku. Statt der üblichen Ziffern von 1 bis 9 verwendet Alphadoku die Buchstaben des Wortes MAISFORTH als Zeichensatz. MAISFORTH stammt natürlich vom Autor, Frans van der Markt. Hier das Rätsel:

```
. . F  I S .  . . A
. . I  . R .  . . S
. . A  . . H  . . .

. . M  R F T  . . .
```

```
. . H . . . . . .
T . S . . . . R .

. . . . . . A T I
A . . M . . S . .
. H . O . . M . R
```

Ich (der Rezensent) habe mich beim Lösen schneller zurechtgefunden, nachdem ich M A I S F O R T H durch 5 1 4 8 2 6 7 9 3 ersetzt hatte. Man ist dann beim schnellen Durchmustern mit dem Auge sicherer, kein Zeichen (keine Ziffer) ausgelassen zu haben. Die erste Zeile der Lösungsmatrix lautet: RTFISOHMA .

Der jetzige Vorstand — Mitteilung der Redaktion

Vorsitzender:	Willem Ouwerkerk
Sekretär:	Frans van der Markt
Schatzmeister:	Ben Koehorst
Mitglied:	Albert van der Horst
Mitglied/HCC-Koordinator:	Dick Willemse
Mitglied/Redakteur:	Ron Minke

Das Vijgeblaadje erscheint um den Ersten herum eines jeden geraden Monats.

Leserbriefe und Meldungen (Fortsetzung von Seite 24)

Die Anwendung — Automatisierung

Eine Vision, die mir schon lange vorschwebt, wäre, Roboter in der Industrie mit FORTH zu steuern.

Nun ist es ja so, dass alle Roboterhersteller schon ihre eigene Programmiersprache etabliert haben, weil sie FORTH schlicht und ergreifend nicht kannten.

Einen Standard wie z.B. „IEC–61131“ für speicherprogrammierbare Steuerungen –SPS– (der übrigens auch viele Jahre brauchte, um sich durchzusetzen und von Siemens immer noch mit einem eigenen Dialekt versehen wird) gibt es für Roboter nicht und die IEC–61131 setzt sich gar nicht oder nur zögerlich auf Robotern durch.

Manch ein FORTH–Programmierer fragt sich jetzt sicher, was bedeutet „IEC–61131“. Den Standard hier darzulegen, wäre natürlich fehl am Platz, aber es bedeutet grob gesagt 5 Programmiermethoden für die Automatisierungstechnik mit:

FUP, Funktionsplan	(Signalverabeitung — grafisch)
KOP, Kontaktplan	(Relaisersatz — grafisch)
AS, Ablaufsprache	(Ablaufkonstrukte — grafisch)
AWL, Anweisungsliste	(Textsprache in UPN auf niedrigem Niveau)
ST, Strukturierter Text	(Kreuzung aus Pascal, BASIC und ???).

Die Idee in diesem Zusammenhang wäre jetzt, ein Open–Source–Project zu starten, welches anfangs einen Parser für FORTH schafft, der IEC–61131–Statements in FORTH umsetzen kann und damit eine programmierbare Logik (PLC), bzw. speicherprogrammierbare Steuerungen (SPS) zum Leben erweckt. Man könnte auch sagen, einen FORTH–Realtime–Kernel für unterschiedlichste Hardware und Plattformen zu erzeugen. Dieser ist dann natürlich auch für Roboter geeignet und könnte mit der „Ummantelung IEC–61131“ Akzeptanz und Einsatz finden.

Einen ersten Ansatz für dieses Thema habe ich in Anton Ertls Paket „Gray“ gefunden. Gray bringt schon eine Menge mit, was erforderlich wäre. In Antons Beispielen findet sich eine kleine Pascal–Quelle, die mittels Gray auf FORTH umgesetzt wird.

All dies beinhaltet jedoch einen nicht zu unterschätzenden Aufwand, denn Steuerungen müssen echtzeitfähig sein, Multitasking realisieren und eine Möglichkeit zum komfortablen Debuggen bieten. Peripherieanbindungen über Feldbusse, wie z.B. CANopen, ProFiBus, Sercos, Powerlink, etc., sind die Normalität und über 99,9 % Zuverlässigkeit sollte nicht diskutiert werden. Prozesssicherheit und Zuverlässigkeit bringen unter Umständen die Notwendigkeit einer Zertifizierung mit sich, was in FORTH, entsprechend einem Vorschlag von Uli Hoffmann, jedoch bestens lösbar wäre.

Darüber hinaus könnten aus der freien Software–Welt viele gute Dinge, wie z.B. Treiber in Linux, mit übernommen werden, FORTH kann an Stellenwert gewinnen und seine Potenzen für die Automatisierung ausspielen, denn dessen Nutzung bleibt ja nach wie vor offen und evtl. wächst dann sogar die FORTH–Gemeinde.

Ausbaufähig für die Zukunft bliebe ein solches Open–Source–Project auf jeden Fall, denn es bleiben immer noch die grafischen Teile der IEC–61131, welche dann auch an den, ich nenne es mal FORTH–Realtime–Kernel, angebunden werden müssten.

Ich möchte diese Überlegungen zur Diskussion stellen, denn ich nehme mal an, dass eine Sprache, welche nicht, oder nur wenig genutzt wird, sei diese noch so gut, über kurz oder lang zum Scheitern veruteilt ist. Eine sinnvolle Anwendung kann Referenzen bringen und natürlich auch wieder Input für FORTH selbst geben. Viele Dinge, wie z.B. die Objektorientierung etc., wurden schon diskutiert und realisiert, sollten diese aber in einem solchen gemeinsamen Open–Source–Project notwendig sein, dann werden diese realisiert und finden ihren Stellenwert. Nicht zuletzt könnte davon eine Fokussierung ausgehen, die die nun doch schon fast nicht mehr zu überschauende Vielfalt an FORTH–Systemen und Dialekten bündelt.

Last but not least, richtige Knickarm–Roboter könnten schweißen, kleben, palettieren etc. ;-)

Dezember 2006 / Gerd Franzkowiak

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 59, Dezember 2006

F51 de volgende stap — Willem Ouwerkerk

Übersetzung des Vorspanns: „Marc Hartjes und ich beschäftigen sich seit gut einem Jahr mit einem Nachfolger für das altvertraute ATS–Board. Warum? Nun, der daraufsitzende 80C535 wird nicht mehr hergestellt und wird wohl in kürzester Zeit im Preis steigen. Wir haben darüber nachgedacht, was ein solcher Nachfolger alles können sollte: Wir würden gern noch mehr I/O–Anschlüsse sehen, einen schnelleren Chip mit Zukunft, einen Festplattenanschluss, einen Bootlader, der das Draufladen von neuer Software erleichtert. Auch würden wir gern eine Lowpower–Version verwirklicht sehen."

Es folgt im Artikel ein Blockschaltbild und eine Aufzählung der vorgesehenen Bestandteile der F51–Platine mit Erläuterungen: AT89S8253, 24 MHz, ATMEGA169 mit RTC, SPI–Bus–Verbindung, 128k–RAM, 8xADC, 48xI/O, SD–Karte, I2C–EEPROM.

In de aanbieding — Albert van der Horst

Im Laufe der Zeit hat sich eine ganze Reihe von vereinseigenen forth–bezogenen Teilen angesammelt. Die Forth–Freunde, die diese Dinge bisher bei sich beherbergten, räumen auf. Es soll nichts weggeworfen werden. Also wurde eine Versteigerung gestartet. Die unten stehenden Preisangaben stellen das Mindestangebot dar.

3 tragbare HP–Computer (10,– Euro), 1 programmierbares HP–Voltmeter (5,– Euro), 2 NOVIX–Einplatinen–Computer (5,– Euro), 1 HP–8-Floppy-Drive (3,– Euro), 1 Erstdruck von SStarting Forth"mit Widmungen von Charles Moore und Leo Brody (100,– Euro), verschiedene Handbücher, darunter Fysforth (1,– Euro).

IR–afstandsbedieningen en coderingen — Ernst Kouwe

Übersetzung des Vorspanns: „Zahlenschlösser kannte man vom Fahrrad her und von den Aktenköfferchen. Tausendmal probieren, und auf waren sie. Heutzutage gibt es Fernsehgeräte, bei denen der Erwachsenenkanal auch hinter einem Zahlenschloss sitzt, aber dann so eins, bei dem man zehntausendmal probieren muss. Wenn die werte Gattin den Code eingestellt hat, *der Kinder wegen*, und wenn sie es nicht für sinnvoll hält, den teuren Gemahl einzuweihen, entsteht ein unheimlicher Drang, die persönliche Freiheit wiederzugewinnen. Ganz natürlich, aus Prinzip...

Was ist zu tun? Wir wollen einen Automaten bauen, der alle Codes durchgeht, einen nach dem anderen, während man selbst ruhig und munter an der Lötstation sitzt — und die Angebetete nach den aufregenden Tagen in die Stadt gefahren ist. Wir müssen den Code der Fernbedienung irgendwie auslesen, entziffern und nachbilden."

Im Hauptteil des Artikels erklärt der Autor, wie er das Problem, den Code der Fernbedienung zu knacken, angegangen ist, nur mal so, ganz allgemein, ohne Forth.

Im *Epilog* erklärt der Autor dann, was ihm eigentlich am Herzen gelegen hatte: Er wollte die Fernbedienung des Beamers in seiner Schule für den Fall aller Fälle nachbauen, für den Fall nämlich, dass wieder mal so ein vergesslicher Lehrerkollege die Fernbedienung in der Jackentasche mit nach Hause nahm und den ganzen Betrieb aufhielt. Die Geschichte mit seiner Frau, den Kindern und der Wahrung seiner persönlichen Freiheit, so gesteht er, war erstunken und erlogen. Ganz kleinlaut gibt er außerdem zu, dass er Monate brauchte, um in der Bedienungsanleitung zu blättern und herauszufinden, dass die werksseitige Einstellung, die natürlich niemand je verändern wird, 1234 ist...

HELP: Next? — Ron Minke

Der Rezensent: In den folgenden drei Zeilen übersetze ich den unter dieser Überschrift veröffentlichten Aufruf von Ron Minke, dem Redakteur des Vijgeblaadjes.

„Persönliche Gründe zwingen euren Redakteur des Vijgeblaadjes leider, sein Amt niederzulegen. Wer will der Forth–Gruppe mit der Vijgeblaadje–Organisation weiterhelfen? Meldet euch beim Vorstand auf der nächsten Zusammenkunft!"

Der Rezensent:

Die Zusammenkunft, auf die hier Bezug genommen wird, war am 9. Dezember 2006. Ein historischer Tag: Es hat sich nämlich, wie ich aus meiner hoffentlich noch recht lange weiterbestehenden engen E–Mail–Verbindung mit Ron erfahre, keiner gemeldet. Ich empfinde das als sehr schade. Gerade aus dem diesmaligen Feigenblättchen erfahren wir ja wieder, dass unsere Nachbarn im Westen in Bezug auf Forth erstaunlich aktiv sind. Aktiv sein aber, und das kann ich sehr gut verstehen, und die Ergebnisse der Aktivitäten niederzuschreiben und in immer wiederkehrender verlässlicher Form den Lesern darzubieten, sind zwei Paar Stiefeln. Dabei hilft auch die heutzutage üblich gewordene Form der elektronischen Darstellung recht wenig: Von nichts kommt nichts. Ich bin der Meinung, Ron habe seine Sache gut gemacht, und ich kann nur hoffen, dass unsere niederländischen Forth–Freunde einen Weg finden, das ansehnliche Vijgeblad aus den früheren Jahren in Form des Vijgeblaadjes (der kleineren Ausgabe aus den letzten Jahren) weiterzuführen. Mir hat die Zusammenarbeit mit Willem Ouwerkerk, mit Albert Nijhof und seit geraumer Zeit mit Ron Minke immer große Freude bereitet. Das Direktorium der FG und das Redaktions–Team der Vierten Dimension werden, dessen bin ich sicher, die Bestrebungen unserer Forth–Nachbarn mit Wohlwollen begleiten.

Volksforth–Update

Carsten Strotmann

Neue Version für Commodore–Rechner

Im August wurde eine überarbeitete Version der Commodore C64/C16/Plus4–Version veröffentlicht. In dieser Version sind nun die Kommentare im Quellcode in englischer Sprache, die Anfangsinformationen (README–Datei) gibt es nun in deutscher und in englischer Ausführung. Von der Atari–8Bit–Version (siehe unten) wurden einige neue Quelltexte mit auf die Disketten gepackt.

Die neue Version habe ich dann in vielen verschiedenen Commodore–Foren bekannt gemacht und die Downloadzahlen sprangen Ende August in die Höhe. Das VolksForth–Projekt war in der Rangfolge aller Sourceforge Projekte (mehrere 10tsd) auf einen Rang unter 500 vorgestoßen.

Forth–Kurs

Mitte September gab es dann einen Forth–Einsteiger–Wochenendkurs in Darmstadt mit 7 Teilnehmern. Neben VolksForth war auch gForth, BigForth und Open–Firmware ein Thema. Drei Teilnehmer des Kurses haben sich entschlossen, einen Swap–Plüsch–Drachen zu *adoptieren*, heißt, sie sind Mitglied der FG geworden. Nachdem der Forthkurs allen Beteiligten sehr viel Spaß gemacht hat, werde ich versuchen, für das Jahr 2007 wieder einen Kurs zu organisieren.

Mensch–Computer

Im September bin ich mit William (Bill) Mensch vom Western–Design–Center in Kontakt getreten. WDC verfügt über die Rechte der 6502–Rechnerarchitektur. WDC vertreibt auch einen Konzept–Rechner rund um den 65816 (6502 in der 16Bit–Ausführung), den *Mensch*–Computer (benannt nach obigem Bill Mensch).

Das Systemhandbuch des Mensch–Computers erwähnt ein Forth–System für den Rechner, aber weitere Informationen waren nicht zu finden. Ich fragte WDC nach einem Testsystem für eine Anpassung des VolksForth und hatte Ende September ein Testsystem. Eine erste Version des VolksForths für den Mensch–Computer ist für Ende Januar vorgesehen. Informationen zum Mensch–Computer gibt es auf der Webseite des Western–Design–Centers, `http://www.westerndesigncenter.com`.

Classic Computing

Auf der Classic Computing in Nordhorn, einer Ausstellung für über 20 Jahre alte Rechner, wurde an zwei Tagen (30. September / 1. Oktober) über VolksForth und die Forth–Gesellschaft informiert.

ABBUC–Softwarewettbewerb

Der Atari Bit Byter User Club (ABBUC) schreibt jährlich einen Software–Wettbewerb für neue Programme aus. Nachdem VolksForth den Sommer über für den 8Bit–Atari (800XL, 130XE, 6502 CPU) angepasst wurde, habe ich diese VolksForth–Version für den Softwarewettbewerb eingereicht.

Anfang Oktober wurde diese Version auf einer doppelseitigen 5 1/4"–Diskette an die ca. 400 Mitglieder des ABBUCs verschickt. Oliver Rapp hat für den Disketten–Aufkleber ein neues Logo entworfen, welches nun auch auf der VolksForth–Webseite zu sehen ist (`http://volksforth.sf.net`).

Auf der Jahreshauptversammlung des ABBUCs Ende Oktober wurden die eingegangenen Stimmen der Mitglieder ausgezählt und das Abstimmungsergebnis bekanntgegeben. Ich hatte mit einer Platzierung im hinteren Mittelfeld gerechnet, da in den vergangenen Jahren wenig spektakuläre Wettbewerbsbeiträge, wie Programmiersprachen oder Anwendungen, immer hinter den Spiele–Programmen lagen. Umso erstaunter war ich, als es hieß, VolksForth hat den ersten Platz erreicht und das Preisgeld von 500 Euro gewonnen. Ein schöner Erfolg, dank des Teams, welches über die letzten 25 Jahre am VolksForth gearbeitet hat. Das Preisgeld wird dem Projekt zufließen, z. B. um Ausgaben für neue Hardware zu begleichen.

Ausblick auf 2007

Für das kommende Jahr sind eine Reihe von neuen VolksForth–Entwicklungen geplant:

- im Januar die erste Version für den Mensch–Computer (6502–Version), danach ggf. eine Anpassung an den 65816.
- Die CP/M–Version bekommt auch eine Wiederveröffentlichung. Ich plane auch eine Anpassung an die Z80–*Laptop*–Maschinen Cambridge–Computer–Z88 und Amstrad NC100. Neben der CP/M–Version soll es auch eine AMS–DOS–Version für die Schneider/Amstrad–CPC–Reihe geben.

  Der Apple II–Version fehlt noch die Pro–DOS–IO–Schnittstelle. Diese sollte in ersten Halbjahr 2007 mit der ersten offiziellen Apple–II–Version erscheinen.

Auf der jährlichen Tagung der Forth–Gesellschaft werde ich euch über die weiteren Fortschritte des Projektes berichten.

Forth von der Pike auf — Teil 7 und 8

Ron Minke

Die hier mit freundlicher Genehmigung der HCC–Forth–gebruikersgroep wiederzugebende achtteilige Artikelserie erschien in den Jahren 2004 und 2005 in der Zeitschrift *Vijgeblaadje* unserer niederländischen Forth–Freunde. Übersetzung: Fred Behringer.

Hier kommen die Folgen sieben und acht der Wiedergabe des Versuches, ein AVR–Forth–System mit der Voraussetzung from scratch zu erstellen. (Ed.: Um die VD–Leser nicht allzu lange warten zu lassen, wurden die Teile 7 und 8, die vorläufig letzten beiden des holländischen Originals, in der vorliegenden deutschen Übersetzung zusammengefasst.)

In der vorigen Folge haben wir gesehen, dass bei Verwendung eines AVR–Prozessors die klassische, indirekt gefädelte Methode für unser Forth–System die einzige in Frage kommende Möglichkeit ist. In der vorliegenden Folge überlegen wir uns, wie das eine oder andere zu implementieren ist. Wir gehen von einer Standard–Ausführung aus. Die besteht aus einem AVR–Prozessor, einem Adress–Latch und einem RAM–Speicher. Das übliche EPROM fehlt; das sitzt bereits als FLASH–Speicher im Prozessor selbst. Wir gehen davon aus, dass im Prozessorchip ein serieller Anschluss in Form eines UARTs zur Verfügung steht. Natürlich müssen wir auch einen AVR–Prozessortyp wählen, der überhaupt externes RAM ansteuern kann (das können sie längst nicht alle). Das im Prozessor eingebaute RAM verwenden wir auch (dadurch wird ein kleiner Teil des externen RAMs außer Kraft gesetzt). In unserem Beispiel wählen wir einen ATmega162–Prozessor in Verbindung mit einem RAM von 32 Kilobyte (Schaltbild siehe Abbildung 1).

Stackgröße

Zuerst müssen wir ein paar Annahmen machen. Wie groß sollen wir den Datenstack wählen, und wie groß den Returnstack? Wenn wir sie alle beide im internen RAM des Prozessors unterbringen können, dann haben wir auf jeden Fall die allerschnellste Konfiguration. Der Zugriff auf das interne Prozessor–RAM kostet nur 1 Maschinenzyklus, während ein Zugriff auf das externe RAM 2 Zyklen verlangt. Hier bietet sich der erste Zeitgewinn an. Für ein Stardard–Forth für Experimentierzwecke reichen uns

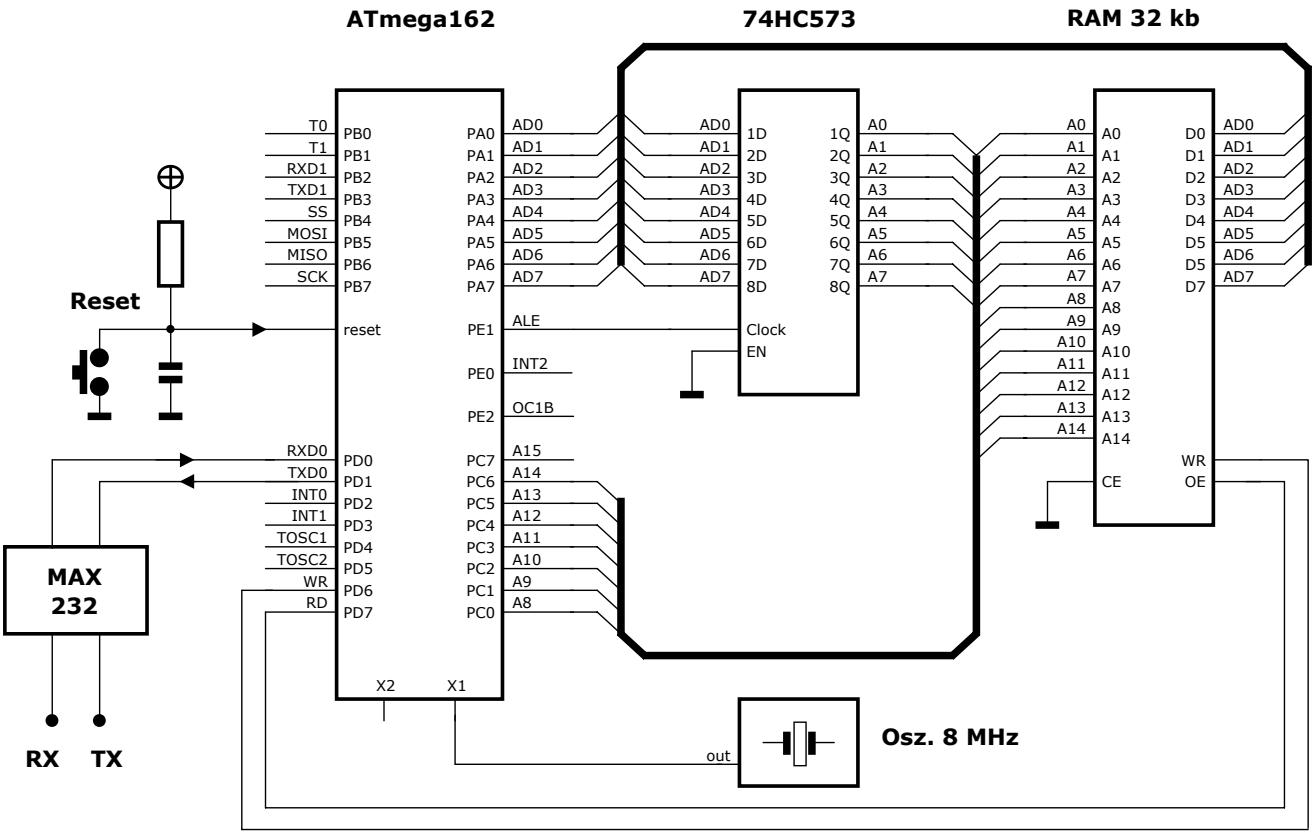


Abbildung 1: Schaltbild des Versuchsaufbaus

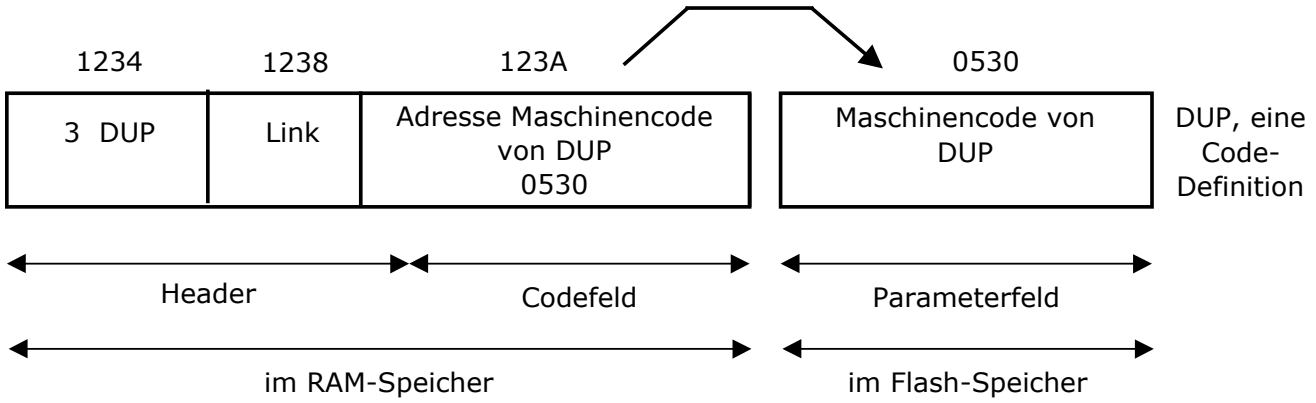


Abbildung 2: Das Wort DUP, teils im RAM, teils im Flash

fürs Erste ein Datenstack von 32 Worten und ein Returnstack von ebenfalls 32 Worten. Der ATmega162 hat ein internes RAM von 1024 Bytes, also genügend Platz, um tatsächlich beide Stacks intern aufzunehmen.

Entscheidung 10: Der Datenstack und der Returnstack sind je 32 Worte lang und befinden sich im internen RAM des Prozessors.

Über den Forth–Code

Dieses experimentelle Forth ist nach dem FIG–Modell (aus dem Jahre 1982!) aufgebaut. Das deshalb, weil das Modell an zahlreichen Literaturstellen beschrieben wurde. Ein erschöpfendes Buch ist *The Forth Encyclopedia* von Mitch Derick und Linda Baker. Das FIG–Modell geht jedoch von einem Forth aus, das vollständig im RAM läuft. An vielen Stellen im System werden da Pointer angepasst und Code–Worte hinzugefügt. In Folge 6 unserer Serie haben wir gesehen, dass das in der AVR–Umgebung nun gerade nicht geht. Mit einigem Erfindungsgeist lässt sich hierfür aber eine Lösung finden. Wenn wir uns eine Methode ausdenken, bei der alle Low–Level–Codeworte ins Flash wandern und alle High–Level–Worte ins RAM, dann können wir uns dem Forth in normaler High–Level–Art nähern. Das Selbermachen von Code–Worten ist dann jedoch nicht möglich. Es geht also darum, einen wohldurchdachten Kernel zu entwickeln, der bereits alles enthält, was wir haben wollen!

Der Low–Level–Kernel

Woraus besteht der Kernel–Code des Forth–Systems nun eigentlich? Richtig: Aus CODE. Wir müssen uns gut klarmachen, dass ausschließlich Code (Maschinenbefehle) ausgeführt werden kann. Dass die Forth–Codeworte auch einen Namen haben, ist nebensächlich, aber angenehm. Und wenn wir jetzt alle Maschinenbefehlsteile auf einen Haufen ins Flash werfen, und alle Namen ins RAM bugsieren? Dann entsprechen wir auf jeden Fall dem AVR–Prozessor–Modell. Aber mit Code–Worten ohne Namen haben wir noch kein Forth–System. Auf die eine oder andere Art müssen wir doch die Namen verarbeiten können. Die Auflösung: Wir stellen eine vorgefertigte Liste auf, die nur die Namen der Code–Worte enthält, setzen die in den Flash–Speicher und kopieren beim Hochfahren des gesamten Systems sämtliche Codewort–Namen ins RAM. Die Forth–Worte können dann in gewohnter Weise dadurch angesprochen werden, dass man ihre Namen im RAM–Speicher aufsuchen lässt und anschließend den zugehörigen Maschinencode im Flash–Speicher zur Ausführung bringt. Der zugehörige Maschinencode wird über einen Pointer, die CFA, erreicht. Zur Verdeutlichung die Darstellung des Wortes `DUP` (siehe Abbildung 2).

Der Header und das Codefeld befinden sich im RAM–Speicher, wobei das Codefeld auf ein Stückchen Maschinencode auf Adresse 0530 im Flash–Speicher zeigt. Wir sehen hier auch, dass neben dem Namen des Forth–Wortes auch die Linkfeld–Adresse des vorhergehenden Wortes und natürlich die CFA im RAM–Speicher liegen müssen. Der Maschinencode des Parameterfeldes liegt im Flash–Speicher. Damit haben wir ein Standard–Forth erzeugt, mit der Besonderheit, dass die Bytes nicht allesamt hintereinander im RAM–Speicher liegen, sondern sich auf zwei Bereiche verteilen, die ihren Platz in zwei verschiedenen, voneinander getrennten Speicherteilen haben. Das geht problemlos, solange nur alle Zeiger auf den richtigen Platz verweisen. Es sollte klar sein, dass die Suchfunktion (ein Low–Level–Codewort im Flash), die die Forth–Worte in der Wortliste aufsucht, ihre Suchaktion vollständig im RAM–Speicher (dem Datenaufbewahrungsort) ausführt. Bevor das alles richtig ineinanderpasst, ist noch einiges an Knobelarbeit zu verrichten.

Der High–Level–Kernel

Neben den Low–Level–Codeworten besteht der Forth–Kernel auch aus High–Level–Worten. Zunächst einmal müssen wir uns vergegenwärtigen, dass ein High–Level–Wort ein echtes Forth–Wort ist, das also aus Daten besteht. Für Maschinencode gibt es hier nichts zu tun. Die Bytes, die da stehen, sind ganz und gar reine Daten. Es braucht hier kein Maschinencode ausgeführt zu werden. Die Forth–Maschine ist ja eine virtuelle Maschine. Die eigentliche Arbeit wird von einem ganz kleinen Stück speziellem Maschinencode geleistet, von `NEXT`. Wir können

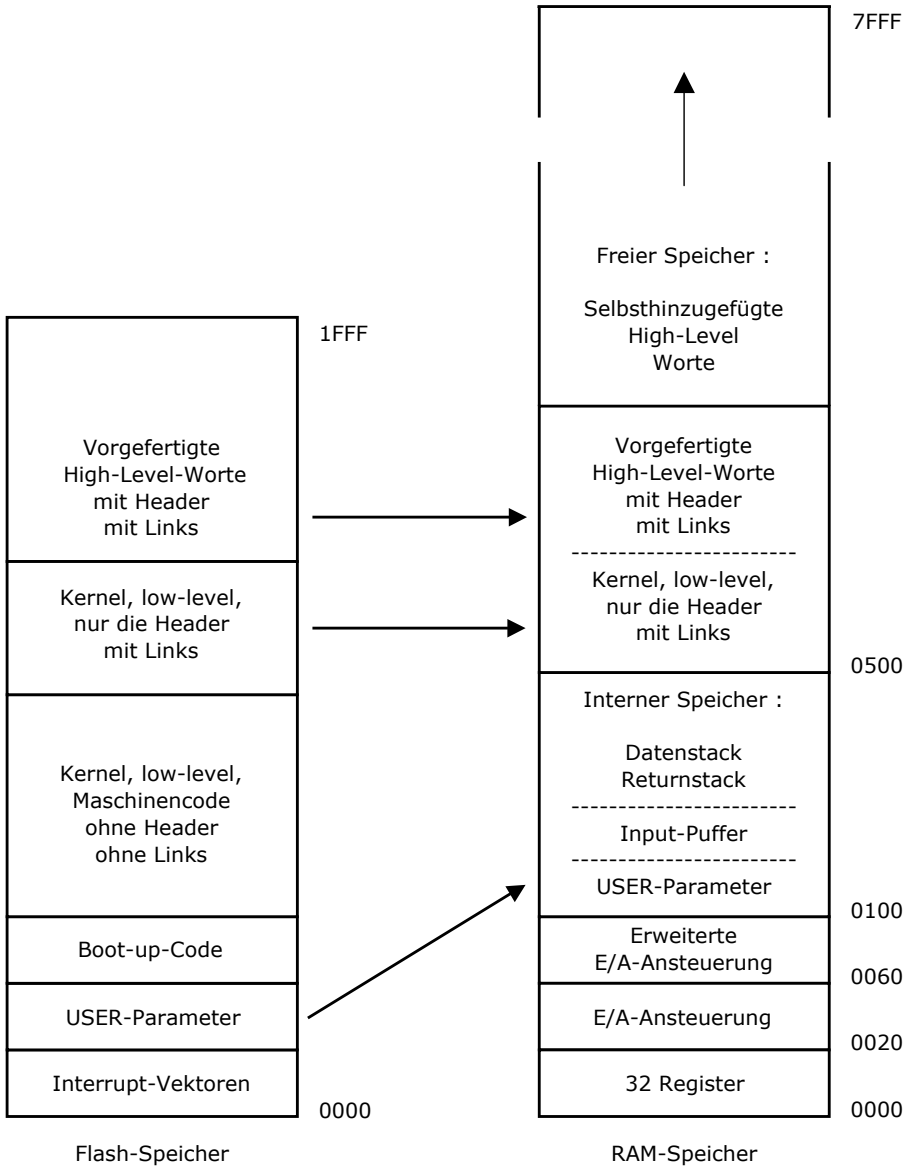


Abbildung 3: Speicheraufteilung nach der Kopieraktion

in den Kernel sehr wohl eine Anzahl von High–Level–Worten aufnehmen, die dann aber als vorgefertigte Daten vorliegen müssen, welche beim Hochfahren erst ins RAM zu kopieren sind. Wir müssen also in den vorgefertigten Worten alle Kopplungen (Links) und Verweise auf andere High–Level–Worte von Anfang an sorgfältig setzen. Das Ganze bekommt erst dann seinen vollen Wert, wenn es ins RAM kopiert ist. Auf dem ursprünglichen Platz im Flash lässt sich nichts damit anfangen. Hier liegt eine auserlesene Aufgabe für einen Forth–META–Compiler vor (ein META–Forth ist ein Forth, mit welchem man ein anderes Forth aufbauen kann). Eine Version, die mit einem eigenständigen AVR–Assembler (beispielsweise den von Atmel, dem Hersteller des Atmega162) erzeugt wurde, ist natürlich auch möglich, aber das kostet viel mehr Mühe. Allerdings weiß man bei Verwendung eines Assemblers bis auf den letzten Maschinenbefehl, wie das aufzubauende Forth–System aussieht.

Abbildung 3 gibt eine Vorstellung davon, wie die Dinge nach der Kopieraktion beim Hochfahren im Speicher eingeteilt sind.

(Übers.: Hier beginnt die Übersetzung des ursprünglich achten Teils des holländischen Originals.)

In Abbildung 3 haben wir gesehen, wie wir mit einer Kopieraktion das vorgefertigte Forth vom Flash ins RAM verfrachten können. Das liefert uns ein funktionierendes Forth–System. Mit solch einem System(chen) kann man bereits viele Dinge auf einem selbstentwickelten Hardware–Experimentieraufbau ausprobieren. Unangenehm am Quelltext einer solchen Forth–Version ist der Umstand, dass man schon beim Entwurf darauf achten muss, dass die vorgefertigten Worte im RAM am richtigen Platz zu liegen kommen. Eigentlich wäre hier ein gehörig großer RAM–Bereich eine schöne Sache: Es

wäre dann Platz genug da, um den vorgefertigten Anteil an High–Level–Worten ins RAM zu kopieren. Später angefügte Definitionen fänden dann ihren Platz oberhalb des so kopierten Teils. Aber was ist, wenn ein hinreichend großer RAM–Bereich gar nicht vorhanden ist? Können wir unser Forth–System dann immer noch verwenden? Das widerspricht doch eigentlich unserem Ausgangspunkt, nämlich dem Entwurf eines Forth–Systems ganz aus dem Nichts.

Das Unmögliche wurde doch möglich

Es ist natürlich eine Extraherausforderung, ein Forth auf einem nackten Prozessor ohne externes RAM zum Laufen zu bringen. Im ersten Moment denkt man: Das klappt nie, das funktioniert nicht. Wenn Sie jedoch das hier Geschriebene lesen, werden Sie begreifen, dass es sehr wohl gelungen ist. Aber wie können wir denn den vorgefertigten Block an High–Level–Worten auf einem nur recht kleinen Bereich an internem RAM unterbringen? Die Antwort lautet: Das tun wir nicht. Wir lassen den vorgefertigten Block einfach da liegen, wo er liegt, nämlich im Flash! ABER ????? Forth benötigt doch einen gewissen RAM–Bereich, um seine Verwaltungsaufgaben erledigen zu können?? Wo es seine Wortliste absetzen kann?? Das ist absolut richtig, jedoch die Art und Weise wie, können wir unseren eigenen Vorstellungen anpassen.

Die Wortliste intern oder extern?

Basis unseres Forth–Systems ist die Wortliste. Wenn wir mit einer Colon–Definition ein neues Wort erzeugen, müssen, wie wir wissen, die Verweise auf das, was dieses neue Wort tun soll, im RAM Platz finden. Und zwar ganz oben im schon bestehenden Teil. Und wenn wir nun dafür sorgen könnten, dass sich in der Wortliste, die im RAM liegt, nur ein einziges Wort befindet und dass alle anderen vorgefertigten Worte im Flash abgelegt sind? Wir müssen dann dafür sorgen, dass der Verweis auf die internen Worte richtig arbeitet. Und dass wir die internen Worte mit der Suchfunktion `FIND` finden können. Hier drängt sich wieder der Unterschied zu einem System auf, das mit einem 8052–Prozessor arbeitet: ROM (beim AVR der Flash–Speicher) und RAM können wir dort im Gesamtspeicher aufeinanderlegen. Für die Suchfunktion besteht dann kein Unterschied, ROM– und RAM–Bereich sind dann praktisch dasselbe. Schön und gut, aber wir haben einen AVR–Prozessor, und der kann das nun einmal nicht. Was er aber sehr wohl kann, ist die Behandlung der beiden Speicherarten auf eine andere Weise. Im Befehlssatz des AVRs gibt es für das Einholen von Daten zwei verschiedene Maschinenbefehle (siehe Befehlssatz in der AVR–Dokumentation auf der Atmel–Website).

Das Anpassen von `FIND`

Wir müssen uns für die Suchfunktion `FIND` etwas überlegen, das deutlich macht, wo `FIND` suchen soll. Man denkt zunächst einmal an ein Software–Flag, das anzeigt, ob die Suche intern, im Flash des Prozessors, vonstattengehen soll, oder extern, im RAM. Alle neu zu machenden Definitionen koppeln wir mit dem Flagstand extern aneinander, und bei allen vorgefertigten Definitionen steht das Flag auf intern. Die Suchfunktion `FIND` muss aus zwei Teilen bestehen; der erste Teil sucht ausschließlich im RAM, der zweite Teil ausschließlich im Flash. Als Trennung zwischen den beiden Teilen könnten wir ein Null–Link gebrauchen, das auf das Ende der Wortliste verweist. Sobald das `FIND` zum ersten Mal auf das Null–Link trifft, muss es auf interne Suche umschalten. Beim zweiten Mal ist das Ende der Wortliste schon erreicht. Auf diese Weise muss eine Forth–Definition gefunden werden können. Nach einigen Software–Experimenten schien das tatsächlich zu funktionieren. Wir begegnen dabei jedoch einem anderen Problem. Das Auffinden eines Wortes ist nur die eine Hälfte des Problems, dessen Ausführung ist die andere Hälfte. Auf die eine oder andere Weise muss man an der Liste von PFAs erkennen können, woraus eine Definition aufgebaut ist, wo sich das Wort befindet. Intern oder extern? Der Schlüssel dazu liegt beim Adresswert.

Es musste noch experimentiert werden

Zunächst dachten wir daran, die internen Worte irgendwie zu markieren, beispielsweise durch ein Hochsetzen von Adressbit 15. Bei den externen Worten sollte das Bit zurückgesetzt bleiben. Alle Code–Worte, die mit dem direkten Einholen von Daten aus dem Speicher zu tun haben, müssen dann dieses Bit untersuchen und die richtige Arbeitsweise wählen. Der Gedanke war einfach, die Ausführung nicht ganz so einfach. Beim Ausarbeiten dieser Idee ergab sich jedoch eine andere Lösung: Eine Trennung in Adressbereiche.

Wenn wir uns den Speicherplan (siehe Abbildung 4) genauer betrachten, bemerken wir zwei Dinge:

Internes RAM beim MEGA162: Von 0100 bis 0500
Flash: Die Header im Kernel beginnen bei (ungefähr) 0500.

Was können wir damit anfangen? Nun ja, hier liegt die Basis einer Implementation der Innerhalb/außerhalb–Idee.

Die Implementation

Wenn wir nun unterscheiden würden zwischen Bereichen unterhalb der Adresse 0500und Bereichen oberhalb 0500? Unterhalb der Adresse 0500 entscheiden wir uns für den Zugang zum internen RAM (mit dem Befehl `Ld`) und oberhalb der Adresse 0500 verwenden wir den speziellen Befehl `Lpm`, um uns den Zugang zum Flash zu sichern. Erst müssen wir jedoch herausfinden, welche Forth–Worte nun eigentlich mit dem Speicher direkt zu tun haben. Worte, die Daten auf dem Stack bearbeiten, fallen nicht hierunter. Die betreffenden `CODE`–Worte sind: `LIT BRANCH NEXT EXECUTE @ C@ CMOVE (FIND) COUNT` und die internen Codeteile der Worte `CONSTANT` und `USER`. Alle diese Worte holen aus

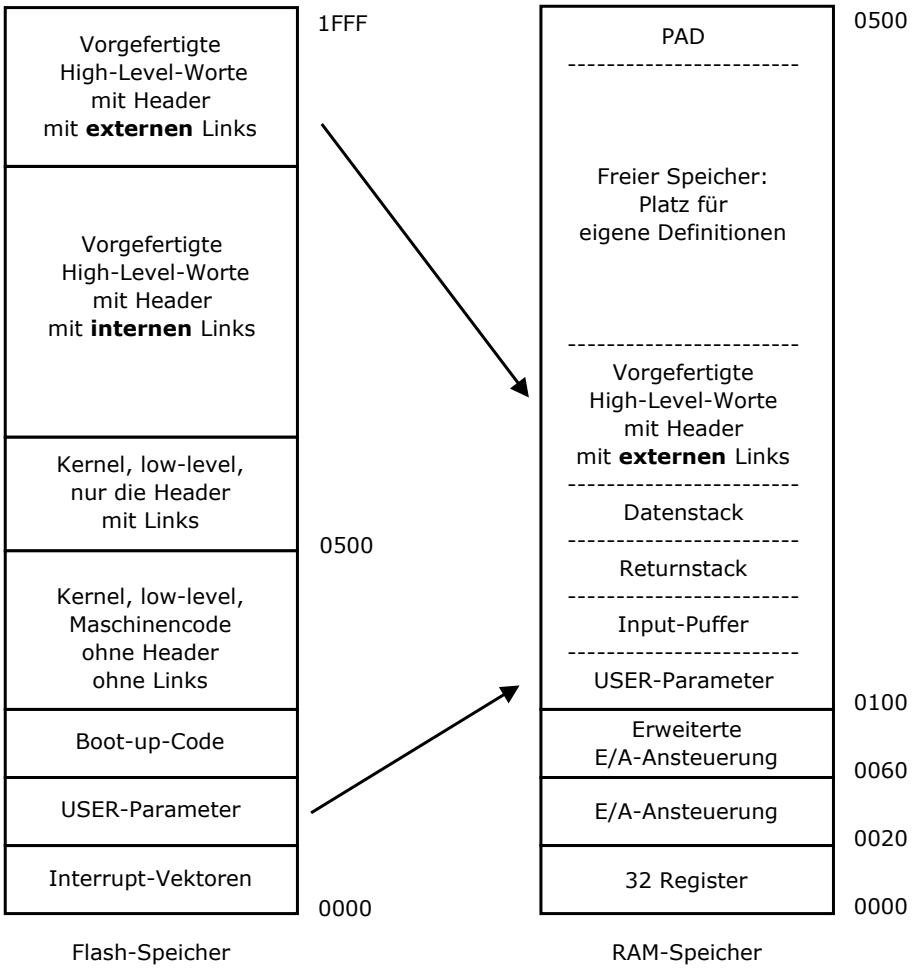


Abbildung 4: Speicherplan

der Wortliste etwas herein und stellen dort dann etwas damit an. In allen diesen Worten wurde ein Test eingebaut, der prüft, ob die angebotene Adresse kleiner als 0500 ist. Wenn ja, dann ist es eine externe Adresse im internen RAM. Wenn nein, dann ist es eine Adresse aus dem vorgefertigten Teil, der nur die Header und Links enthält. Das Ergebnis des Adresstests zeigt an, mit welchem Befehl die entsprechenden Daten hereingeholt werden sollen. Und siehe da: ES FUNKTIONIERT. Das experimentelle Forth läuft auf einem nackten Prozessor! Der einzige Nachteil besteht darin, dass da wenig Platz übrig bleibt, um eigene Definitionen hinzuzufügen. Aber für das eine oder andere Experiment mit einem (selbstgemachten) Hardware–Aufbau ist es prima geeignet. Haben wir keinen Bedarf an der sogenannten Vocabulary–Struktur, dann entfernen wir den dafür vorgesehenen Code und haben ein Simple–Forth übrig, das etwas mehr an Speicher zur Verfügung hat.

Ein letzter Stolperstein

Beim Herumexperimentieren kommt man natürlich auch mal in die Situation, dass das System hängen bleibt. Im unserem Fall kann das dann vorkommen, wenn man zu viele Definitionen hinzugefügt hat. Warum nun das schon wieder? Die Antwort ist in der Struktur unseres Forth–Systems (das auf der FIG–Version 1982 beruht) zu suchen. Das System enthält einen Notizblock für interne Zwecke, das `PAD`, das in einem festen Abstand oberhalb des zuletzt definierten Wortes liegt. Wenn man nun zu viele eigene Definitionen erzeugt hat, ist kein Platz mehr da für den `PAD`–Teil. Oberhalb der Adresse 0500 gibt es einfach keinen Speicher mehr! Forth kann seinen Verwaltungsaufgaben nicht mehr nachkommen ... und bleibt hängen. Eine einfache Lösung besteht darin, dass man bei den `USER`–Konstanten, die das gesamte Forth–System beschreiben, extra Platz für das `PAD` freimacht und an eine niedrigere Adresse die Warnung Wortliste voll legt.

Schlusswort

In dieser Artikelserie haben wir versucht, ein Bild davon zu skizzieren, wie man ein kleines Forth–System von nichts her aufbauen kann, welche Punkte dabei zu beachten sind, welche Wahlmöglichkeiten man hat und welche Entscheidungen man bei diesem Vorhaben treffen muss. Als Ergebnis liegt ein bestens arbeitendes experimentelles Forth auf einem Atmel–AVR–Prozessor (mit oder ohne RAM) vor. Wollen Sie mithelfen, es weiter zu verbessern? Ideen sind sehr willkommen!

Lego Forth NXT

Bernd Paysan

Lego bringt einen Nachfolger für den RCX auf den Markt: Den NXT[1]. Für den RCX gab es bereits ein Forth, das pbForth[2] von Ralf Hempel. Ulli Hoffmann formuliert dazu allerdings vernichtende Kritik:

> Wenn ich an den RCX–Wettbewerb in Haminkeln zurückdenke, fand ich das Forth und das Entwicklungssystem, das wir damals benutzten, primitiv und lästig zu bedienen. Für ein Wettbewerbs–Rapid–Prototyping war das nicht wirklich geeignet und als Vorzeige–Objekt, um andere für Forth zu interessieren, würde es wohl eher abschreckend wirken.

Wobei man fairerweise sagen muss, dass die Hardwareunzulänglichkeiten des RCX–Bricks sehr wohl zu dem unprofessionellen Auftreten beigetragen haben — wenn der Motor keine Tachorückmeldung hat, kann man eben keine Regelung implementieren.

Der Plan

Der NXT ist wesentlich leistungsfähigere Hardware als der RCX. Auf diesem System sollte es möglich sein, ein leistungsfähiges Forth zu implementieren, das komfortabel bedient werden kann, und die Möglichkeiten des Geräts auch ausnützt.

Insbesondere sind auch die oben angedeuteten Schwachpunkte des RCX–Moduls behoben:

- Der Motor hat einen Drehgeber, mit dem eine Steuerung der Fahrzeuge möglich ist
- Die Kommunikation über Bluetooth erlaubt eine sichere Kommunikation zwischen Host und Gerät und auch zwischen den Geräten

Anforderungen

- Ein vernünftiges Forth (z.B. Gforth); mit 256k Flash und 64k RAM auf dem ARM ist genügend Platz und Leistung vorhanden.
- Interaktive Kommunikation mit Terminal–Programmen und Kommunikation zwischen den Robotern ähnlich wie bei ONF.
- Die im Lieferumfang enthaltene Peripherie (Sensoren und Aktoren) müssen ausgereizt werden können — es muss auch eine Bibliothek üblicher Regelalgorithmen vorhanden sein, damit nicht jeder Jugendliche das Rad neu erfinden muss.
- Das Dot–Matrix–Display muss Text und einfache Grafik anzeigen.
- Was machen wir mit dem AVR–Koprozessor? Lassen wir die vorhandene Firmware drinnen (Power– und PWM–Management), oder tun wir ein Umbilical–Forth 'rein, dessen Host im ARM ist?

Vorgehensweise

- Beschaffung hinreichend detaillierter Informationen über den NXT
- Machbarkeit? D.h. kann man die Firmware überhaupt ersetzen?
- Beschaffung von Kits und Bricks für die Teilnehmer
- Zusammenstellen eines Teams
- Beta–Test mit echten Kindern (ewige Spielkinder zählen nicht)
- Dokumentation mit angemessener Berücksichtigung des Vorwissens des Zielpublikums (das Zielpublikum hat noch *kein* Diplom in Elektrotechnik oder Maschinenbau!)

Informationen über den NXT

Auf der NXTreme–Seite[3] gibt es ein paar Informationen zum Aufbau, z.B. Schaltpläne, aber keine weiteren Details. Gottseidank ist der Schaltplan sehr einfach, d.h. man braucht eigentlich nur auf die verwendete CPU zu gucken, um weiterzukommen:

Der verwendete Single–Chip–Contoller ist ein AT91SAM7S256[4]. Da ist ein Boot–Assistent drin (SAM–BA), mit dem man die vorhandene Firmware „ganz einfach“ über USB ersetzen kann (mit einem Windows–Programm natürlich).

Mehr findet man auf der NXT Hacks[5]–Seite. Dort gibt es auch einen Link auf ein Linux–Programm zum Firmware flashen[6], GCC–Crosscompiler und mehr.

[1] http://mindstorms.lego.com/
[2] http://www.hempeldesigngroup.com/lego/pbForth/
[3] http://mindstorms.lego.com/Overview/NXTreme.aspx
[4] http://www.atmel.com/dyn/products/product_card.asp?part_id=3524
[5] http://nxt.natulte.net/trac/
[6] http://nxt.natulte.net/trac/wiki/LibNxt

Forth–Gruppen regional

Moers
Friederich Prinz
Tel.: (0 28 41) – 5 83 98 (p) (Q)
(Bitte den Anrufbeantworter nutzen!)
(Besucher: Bitte anmelden!)
Treffen: 2. und 4. Samstag im Monat 14:00 Uhr,
MALZ, Donaustraße 1
47441 Moers

Mannheim
Thomas Prinz
Tel.: (0 62 71) – 28 30 (p)
Ewald Rieger
Tel.: (0 62 39) – 92 01 85 (p)
Treffen: jeden 1. Mittwoch im Monat
Vereinslokal Segelverein Mannheim e.V. Flugplatz Mannheim–Neuostheim

München
Bernd Paysan
Tel.: (0 89) – 79 85 57
`bernd.paysan@gmx.de`
Treffen: jeden 4. Mittwoch im Monat ab 19:00. Treffpunkt ändert sich im Moment gerade, bitte auf der Web–Site nachsehen oder nachfragen!

Hamburg
Küstenforth
Klaus Schleisiek
Tel.: (0 40) – 37 50 08 03 (g)
`kschleisiek@send.de`
Treffen 1 Mal im Quartal
Ort und Zeit nach Vereinbarung (bitte erfragen)

Mainz
Rolf Lauer möchte im Raum Frankfurt, Mainz, Bad Kreuznach eine lokale Gruppe einrichten.
Mail an rowila@t-online.de

Gruppengründungen, Kontakte

Hier könnte Ihre Adresse oder Ihre Rufnummer stehen — wenn Sie eine Forthgruppe gründen wollen.

µP–Controller Verleih

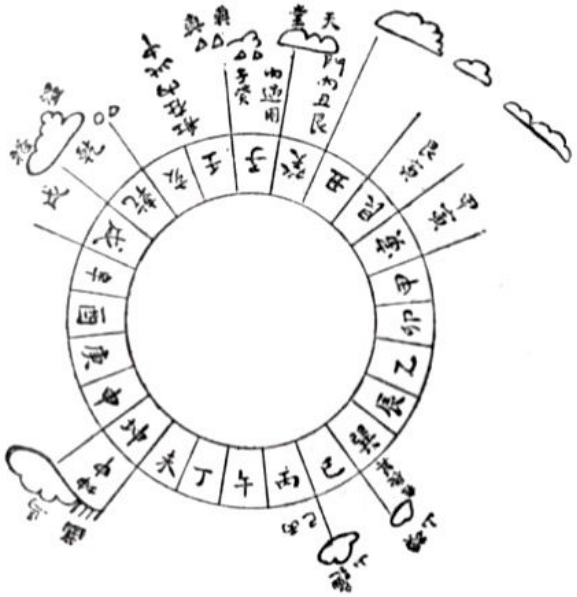
Carsten Strotmann
`microcontrollerverleih@forth-ev.de`
`mcv@forth-ev.de`

Spezielle Fachgebiete

FORTHchips (FRP 1600, RTX, Novix) — **Klaus Schleisiek–Kern**, Tel.: **(0 40) – 37 50 08 03** (g)

KI, Object Oriented Forth, Sicherheitskritische Systeme — **Ulrich Hoffmann**, Tel.: **(0 43 51) – 71 22 17** (p), Fax: **– 71 22 16**

Forth–Vertrieb, volksFORTH, ultraFORTH, RTX / FG / Super8, KK–FORTH — Ingenieurbüro **Klaus Kohl**, Tel.: **(0 70 44)–90 87 89** (p)



Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen hinausgehen? Schreiben Sie einfach der VD — oder rufen Sie an — oder schicken Sie uns eine E–Mail!

Hinweise zu den Angaben nach den Telefonnummern:
Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeitszeiten
g = geschäftlich
Die Adressen des Büros der Forth–Gesellschaft e.V. und der VD finden Sie im Impressum des Heftes.





Krainerhütte Baden Wien

Einladung zur Forth–Tagung 2007

Anton Ertl

Zeit und Ort

26./27.April 2007 – 29.April 2007 im Hotel Krainerhütte im Helenental nahe Baden bei Wien.

Die eigentliche Forth–Tagung beginnt am Freitag (27.4.) um 15h und endet am Sonntag (29.4.) nach dem Mittagessen. Optional kann man auch schon am Donnerstag nachmittag ankommen, für Freitag vormittag ist ein Programm vorgesehen.

Bitte melden Sie sich möglichst bald an, das Hotel reserviert uns ungebuchte Zimmer nur für begrenzte Zeit, danach könnte es mit den Zimmern knapp werden. Als weiteren Bonus gibt es eine Frühbucherermäßigung.

Vorläufiges Programm

26.4.	19h	Abendessen
27.4.	9h	Wanderung (z.B. auf den Hohen Lindkogel)
	12h	Mittagessen unterwegs
	15h	Beginn der Forth–Tagung
	15h	Vorträge, Kaffeepause
	19h	Abendessen
28.4.	9h	Vorträge, Kaffeepause
28.4.	12h	Mittagessen
	14h	Ausflug (z.B. zur Seegrotte Hinterbrühl)
	19h	Abendessen
29.4.	9h	Mitgliederversammlung
	12h	Mittagessen

Umgebung

Die Krainerhütte liegt im romantischen Helenental, ein schönes Gebiet zum Wandern (ein möglicher Ausflug wäre zum Schutzhaus Eisernes Tor auf dem Hohen Lindkogel. In der Nähe befindet sich weiters noch Heiligenkreuz, bekannt für das Kloster Stift Heiligenkreuz; das Jagdschloss Mayerling, wo sich Kronprinz Rudolf das Leben nahm; und die Kurstadt Baden, mit u.a. Casino, Kurpark, und Bad; und natürlich ist Wien mit all seinen Sehenswürdigkeiten auch nicht weit weg.

Anreise

Bei Benutzung von Bahn oder Flugzeug empfiehlt es sich, früh zu buchen, um Billigtarife ausnutzen zu können (sicherheitshalber aber erst die Bestätigung der Anmeldung zur Forth–Tagung abwarten). Mehr Informationen zur Anreise finden Sie auf der Tagungs–Homepage `http://www.complang.tuwien.ac.at/anton/forth-tagung07/`.

Weitere Informationen

... wie zum Beispiel das Anmeldeformular finden Sie auf der Tagungs–Homepage: `http://www.complang.tuwien.ac.at/anton/forth-tagung07/`.