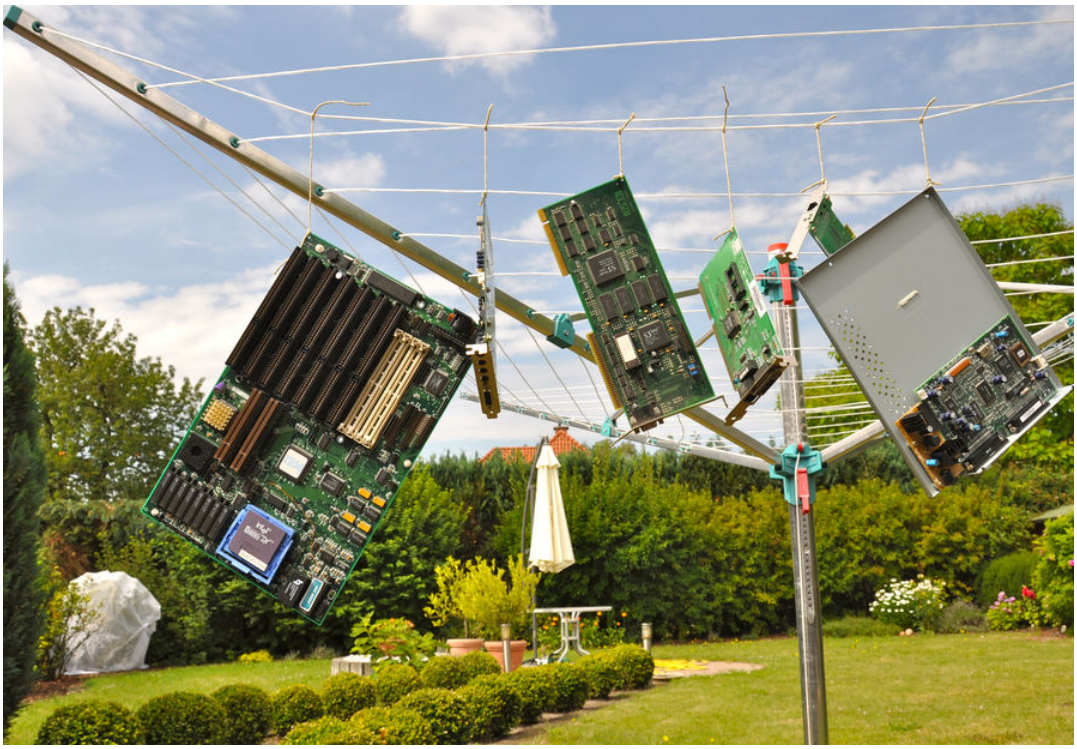
Vierte Dimension

2/2011

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 GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

Bergstraße 10 D–18057 Rostock
Tel.: +49 381 496800-0 Fax: +49 381 496800-29

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–Schöpe

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 32 01 24
68273 Mannheim
Tel: ++49(0)6239 9201-85, Fax: -86
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,

herzlich willkommen zur Sommer–Ausgabe der Vierten Dimension. Der Eine oder Andere wird es bemerkt haben, ein neues Bild auf dieser Seite. Keine Angst, Ulli wird der VD weiterhin als Chefredakteur erhalten bleiben.

Auf der Jahrestagung im April in Goslar wurde beschlossen, die Arbeit des Chefredakteurs auf mehrere Schultern zu verteilen, wobei im Rotierverfahren immer eine andere Person für eine Ausgabe die Arbeit des Chefredakteurs übernimmt. Das werden wir bis zur Tagung im kommenden Jahr so machen, und wenn sich das Schema bewährt, dann wird es beibehalten. Mehr Beteiligte bedeutet mehr Ideen, und ein breites Spektrum an Themen aus dem Verein.

Apropos Tagung 2012, diese wird in den Niederlanden stattfinden, zusammen mit den Forth–Enthusiasten dort. Ort und Termin werden rechtzeitig bekannt gegeben. Koordiniert wird die kommende Tagung von Martin Bitter und mir. Anregungen zur Ausgestaltung und Terminfindung der Tagung sind immer willkommen.

Um das amForth dreht sich vieles im Forth–Universum in diesem Jahr, das erkennt man an den Themen dieser Ausgabe:

Erich Wälde schreibt über den Arduino–Forth–Workshop auf den LinuxTagen in Chemnitz und erklärt, wie verkettete Listen im amForth funktionieren, in einer neuen Folge der Serie ‘Adventures in Forth’.

Michael Kalus re–generiert amForth–Quelltext aus einem Assembler–Text mittels des DELISTER–Programms und erweckt nebenbei ein historisches RSC–Forth–Board mit 65F11–CPU zum Leben.

Matthias Trute schreibt über die Recognizer in amForth, spezialisierte Forth–Wörter zum Parsen von applikationsspezifischen Daten über die Forth–Eingabezeile. Ein weiterer Artikel von Matthias bringt uns Interrupt–Behandlung in Forth näher, wiederum am Beispiel von amForth.

Fred Behringer beschreibt in seiner Serie über Dateisystem–Formate wie sich ein FAT–Dateisystem in den Speicher legen lässt, zum Zwecke der detaillierten Inspektion.

Filippo Sala benutzt Datenstrukturen in Forth, um die Bestandteile des PE–Dateiformats unter modernen Windows–Betriebssystemen zu analysieren.

Hannes Teich gibt eine Vorschau auf seine Arbeiten zur Musik–Generierung mit gForth. Hierzu gibt es einen ausführlichen Artikel in einer späteren Ausgabe.

Und dazwischen stellt ‘Yours truly’ den MS–DOS–Emulator ‘DOSbox’ vor, der neben Spielen auch viele 16bit–Forth–Systeme in die Neuzeit retten kann.

Vieles dreht sich um amForth, aber alle diese Themen sind auch für andere Forth–Systeme umsetzbar, so dass jeder Forthler in diesen Artikeln Anregungen für Arbeit und Heim–Projekte finden kann.

Die Arbeit an der Herbst–Ausgabe 2011 ist schon gestartet. Wie immer werden interessante Artikel, aber auch Nachrichten aus der Forth–Welt gesucht. Das Redaktions–Team der Vierten Dimension wird unter der E–Mail–Adresse ‘`vd@forth-ev.de`’ erreicht.

Viele Grüße und viel Spaß beim Lesen dieser Ausgabe, Carsten

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`

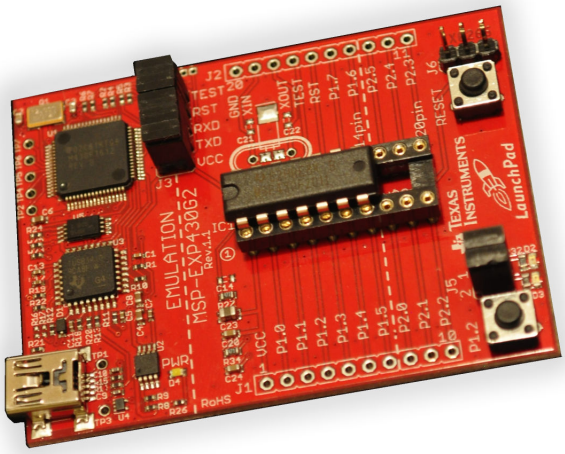
Die Forth–Gesellschaft e. V. wird durch ihr Direktorium vertreten:

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

Das TI–Launchpad und MSP430G2553

Aus einer E–Mail–Kommunikation zwischen Michael Kalus und Carsten Strotmann:

- M: . . . TI–Launchpad MSP430[1]
- C: Ich sehe gerade auf comp.lang.forth, dass Du sehr aktiv bist in dieser Richtung. Ich würde mich über einen kurzen Leserbrief mit Informationen zu Deinen TI–Launchpad–Erfahrungen freuen.
- M: Hm, eigene Erfahrungen damit habe ich ja noch nicht. Dirk Brühl in Pennsylvanien benutzt TI–MSP430–Chips, und meinte, die ersten Stück mit 16k bekommen zu haben, die im 20PDIP–Gehäuse, passend für das TI–Launchpad. Und hat mich aufgestachelt, ein gforth–ec dafür anzufangen. Bernd Paysan, derzeit im Land des Lächelns, war auch dafür. Also ist der erste Schritt in diese Richtung getan, es gibt nun einen Forth–Assembler für den MSP430–Instruktionssatz, den fa430, in einer Alpha–Version sozusagen. Immerhin, einen Test in gforth hat der fa430 gerade eben bestanden, die Opcodes stimmen nun. TI hat mit 512 Byte wenig RAM spendiert für seinen MSP430G2553, die 16K flash sind hingegen schon mal ok. Mal sehen, was draus wird.



Aus einem Chat neulich zwischen Michael Kalus und Dirk Brühl:

- M: Wieviel Mühe machte es denn eigentlich, das TI–Launchpad mit einem Blink–LED, also dem „hello world“ der MCUs, ans Laufen zu bringen?
- D: Das ist kein Problem, man braucht nur den einfachen Anweisungen zu folgen.
- M: Gibt es sie nun wirklich die 20PDIP–MSP430G2553?
- D: Ja, ich habe zwei kostenlose Muster hier.
- M: Und passen die einfach so ins TI–Launchpad?
- D: Ja!
- M: Ist es ein ‘Klacks’, denen per IAR das camelforth[2] einzuflashen?
  D: Nein! Denn, CamelForth ist auf dem 64–Pin–MSP430F161 mit 48kB FLASH und 10kB RAM lauffähig, heißt es. Entwicklungstools dazu gibt es ab US$ 75. Das mit dem MSP430G2553 — 16 kB FLASH und 512 Byte RAM — auf dem Launchpad zum Laufen zu bringen, ist die Herausforderung. Damit gäbe es dann fuer diese interessante Kombination — TI’s Launchpad (US$ 4.30), ergänzt um den MSP430G2553 (US$ 2.80) — ein Forth!
- M: Hierzulande gibt es irgendwie diese TI–Launchpads mit MSP430G2553 dazu nicht.
- D: Gibt es hier auch nicht! Ich habe den Launchpad bei DigiKey bestellt, sobald es ihn dort gab, und die MSP430G2553 habe ich bei TI als kostenlose Muster bestellt, sobald ich davon erfuhr. Da die MSP430 hauptsächlich von Fürstenfeldbruck aus betreut werden, so stellt es sich mir jedenfalls dar, müsste das auch von Deutschland aus möglich sein. Hab gerade mal nachgeschaut (13.06.2011), der MSP430G2553IN20 ist bei europäischen Distributoren nicht erhältlich — offensichtlich derzeit nirgendwo erhältlich. Es gibt da noch den MSP430G2513IN20, aber dazu gibt es nur auf der TI–Webseite eine Liste von Distributoren, die müsste man anrufen. Meine Erfahrung ist, dass die in der Regel nichts haben.
- M: Und nun?
- D: Obwohl der MSP430G2553 im DIL–Gehäuse noch nicht auf dem Markt ist, versuchen wir es mal — ein US/BRD–Kooperationsprojekt sozusagen — zunächst CamelForth, und vielleicht auch das gforth–ec darauf zu bringen.
  In diesem Sinne, Gruß, Dirk.
- M: Da drücke ich die Daumen! Gruß, Michael

MCUs mit FRAM von TI — die idealen Forth–MCUs?

Die TI 130nm Advanced FRAM Technology verspricht Aufsehen Erregendes:

> Embedded–Smart–IC–Speicher wurden soeben noch eleganter, schneller und leistungsfähiger durch die FRAM (Ferroelectric Random Access Memory) Technologie. Speicher, der nicht–flüchtig ist, blitzschnell beschreibbar ist und unerschöpflich wieder beschreibbar, und herausragende Datensicherheit bietet. FRAM–based smart ICs kann man daher leicht kundenspezifisch zuschneiden, produzieren und sehr einfach wieder programmieren, schnell und zuverlässiger als andere IC–Speicher–Typen.

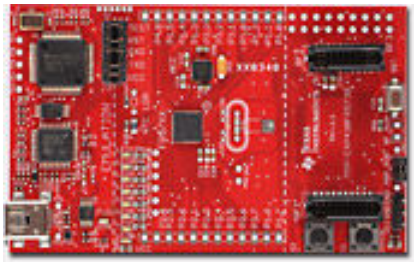
Für Forth heißt das, der FRAM–Bereich ist durchgehend adressiert und verhält sich auch wie ein RAM, obwohl er nicht–flüchtig ist. Müsste also für ein Forth ideal sein.

Und wer einen Freund hat in USA, hat sie schon, die neue FRAM–Experimentierplatine von Texas Instruments MSP–EXP430FR5739. Für 29 US$ kommt sie per Fedex über Nacht als Luftfracht ins Haus. Das geht nur wenn man in USA bestellt, dann aber ohne zusätzliche

[1] `http://www.ti.com/launchpad`: ein Development Kit für die MSP430 Mikroprozessoren (Value Line) für US$ 4.30

[2] `http://www.camelforth.com/page.php?8`

Frachtkosten auch an deutsche Anschriften. Das Experimentierplatinchen kommt mit einem Chip mit 16kB FRAM, 1kB SRAM, 12 Kanal ADC, Comparator_D, 5x16bit Timer, eUSCI (UART, IrDA, SPI, I2C), 32 GPIO. programmierbar via USB. Es ist bestückt mit einem 32.768–KHz–Quarz, die Ports P1 und P3 sind vollständig herausgeführt, sowie einige von P2 auf 2x12–Pin–Leisten. 8xLED, 1xNTC sind auch vorhanden, sowie 2x Taster — also alles, was das Herz begehrt.

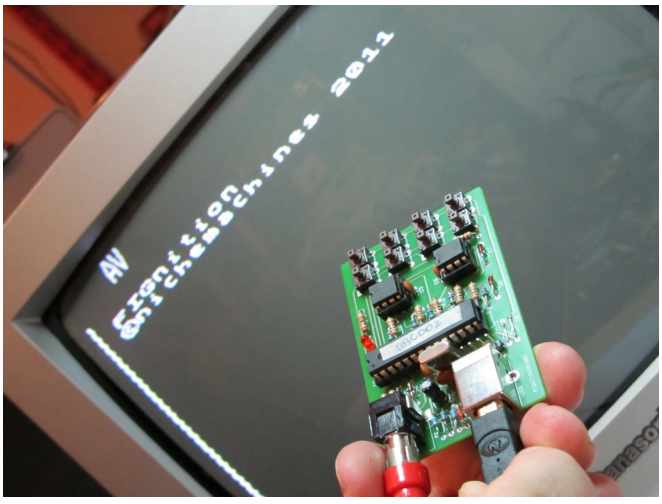


Und wer sich die Sache selber ansehen möchte, kontaktiert am besten Michael Schweiger <mschweiger@ti.com>, Section Head MSP430 Product Engineering, bei Texas Instruments Deutschland GmbH, Haggertystr. 1, D–85356 Freising. Denn dort wurde es gemacht. db/mk

`www.ti.com/fram`

FIGnition DIY Rechner mit Forth

FIGnition nennt sich ein neue kleiner DIY (Do it yourself) Rechner auf Basis eines Atmel AVR chips. Gebaut aus 3 Mikroprozessoren und nur 46 Komponenten ist der FIGnition Rechner für 20 £. Der Rechner hat 8Kb RAM; 384Kb Speicher im Flash, 8–Tasten und PAL Video Ausgabe. Als Programmiersprache dient ein abgewandeltes FIG–Forth.



`http://sites.google.com/site/libby8dev/fignition`

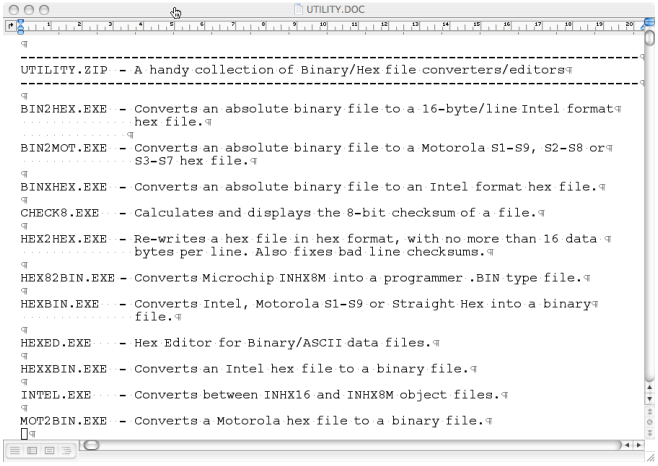
HEX–File–Converter

Hi.

Neugierig unterwegs bei den Großen des Forth fand ich bei mpeforth die kostenlose „Try–before–you–buy“ Demo Software. Darin die utility.zip von ca. 119K, eine nette Sammlung von Hex File Convertern.

`http://www.mpeforth.com/arena.htm`

Grüße, Michael



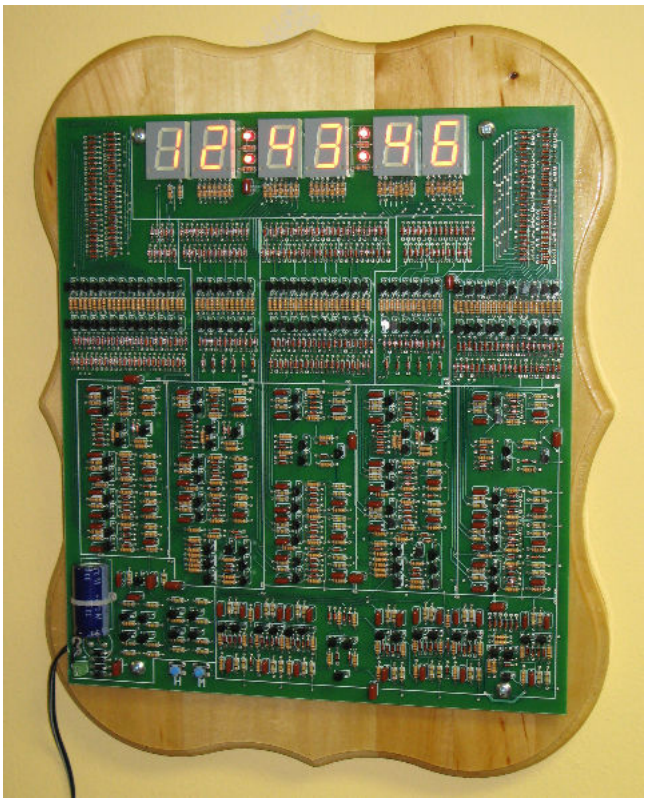
Retro

Hi.

Falls Du die Mikroprozessoren und Forth mal leid bist, ist das hier vielleicht das Richtige:

`http://www.transistorclock.com/`

Dirk



Greetings from Germany to the United States of North America

The undersigned would like to express to Silicon Valley Fort Interest Group their heartfelt thanks for the honorable gift of the two SVFIG T–shirts. Both of us, Carsten Strotmann and Fred Behringer, may consider ourselves as official representatives of the "Forth Gesellschaft" (German FIG), and we are delighted that by wearing the T–shirts we may visibly demonstrate to everyone

the closeness between the German Forth Interest Group and the Forth friends in Silicon Valley. The shirts have already been introduced and dedicated at our recent annual conference, which took place in Goslar, practically in the middle of Germany, from April 14 to 17, 2011.

Sincerely, (C.S. & F.B.) Transl.: HenryVinerts

Grüße aus Deutschland in die Vereinigten Staaten von Nordamerika

Die Unterzeichneten würden gern der Silicon Valley Forth Interest Group ihren herzlichen Dank für die ehrenvolle Überlassung der beiden T–Shirts aussprechen. Wir beide, Carsten Strotmann und Fred Behringer, dürfen uns als Abgesandte der Forth–Gesellschaft (German FIG) betrachten. Wir freuen uns, dass wir mit dem Tragen der T–Shirts die Verbundenheit der Deutschsprachigen Forth–Interessen–Gemeinschaft mit den Forth–Freunden aus dem Silicon Valley für jeden sichtbar zum Ausdruck bringen können. Eingeweiht wurden die T–Shirts zu unserem gerade wieder ausgeklungenen Jahrestreffen, das diesmal vom 14. bis zum 17. April in Goslar, ziemlich in der Mitte von Deutschland, stattfand.

Mit den besten Grüßen, (C.S. & F.B.)

spit.file

Liebe Forth–Freunde,

Dirk, noch immer in USA, fragte neulich, ob es eigentlich auch das Gegenstück zum SLURP–FILE des gforth gäbe? Doch das gab es nicht. Eine Lücke, die unmöglich so bleiben kann, oder? Also:

```
: spit-file ( buffer-adr u name-adr n -- )
  r/w create-file throw dup ( fid) >r
      write-file  throw r>
      close-file  throw ;

: ... ." mach, was du willst" ;

s" my.in" slurp-file ... s" my.out" spit-file
```

mk :-)

Neuigkeiten bei amforth

Die Versionen 4.4 und 4.5 sind kurz nach der Version 4.3 erschienen. Damit wurden die Neuerungen, die dort eingezogen sind, stabilisiert und einige weitere eingebracht. Details weiter hinten in diesem Heft. Damit wurden einige z.T. schon ältere Codezweige abgeschlossen, so dass auch die interne Versionsverwaltung des Autors wieder übersichtlicher wurde.

Das Wichtigste zuerst: Game Of Life. Dessen Portierung hat ein paar Details offengelegt, wie man Forth–Code unter den speziellen Bedingungen des separaten Dictionary–Speichers der Harvard Architektur (also getrennter Programm- und Datenspeicher) zum Laufen bekommt.

Desweiteren wurde die Interruptbearbeitung verbessert. Bislang gab es einige Einschränkungen, was den Einsatz von Forth–Worten als Interrupt Service Routine anging. Die sind jetzt weg. Auch hier der Verweis auf einige Seiten weiter hinten.

Bei der Dokumentation gab es Fortschritte, die Referenz"karte" ist deutlich kleiner geworden, wenngleich es immer noch 7 Seiten sind...

Ansonsten gab es die üblichen Code–Umstrukturierungen (COLD wurde WARM und es gibt ein neues COLD) und es sind noch mehr Dateien geworden, da doppelte Codebereiche zusammengefasst wurden.

Auf vielfachen Wunsch: amforth kann jetzt auch case insensitiv im Dictionary suchen. Das ist (und bleibt) aber in der Standardversion abgeschaltet.

In Version 4.5 sind intern alle Counted–Strings durch die neuen, Forth–200x kompatiblen Addr/Len–Strings ersetzt worden. Dies betrifft inbesondere (BL) WORD und FIND, die durch PARSE–NAME und FIND–NAME abgelöst wurden. Für die Nutzer stehen die bekannten Counted–Strings jedoch weiterhin zur Verfügung.

Matthias Trute

Forthbildung in Chemnitz

Erich Wälde und Carsten Strotmann

Mitte März waren Erich Wälde und Carsten Strotmann für die Forthgesellschaft auf den Chemnitzer Linuxtagen[1]. Wir hatten einen Stand für die Forth–Gesellschaft und einen angekündigten ‚Forth auf dem Arduino'–Workshop[2]. Die Teilnehmer für den Workshops mussten sich vorher anmelden, am Tag des Workshop war dieser dann schon ausgebucht (es gab 20 offizielle Plätze und 22 Teilnehmer), und es gab eine Warteliste. Das Thema scheint Interesse zu wecken.

Der Workshop

Zutaten

- 10 Arduino–Duemilanove–Boards[3]
- 10 Danger–Shields[4]
- 25 nachgedruckte Exemplare der Vierten Dimension, AVR Sonderheft
- ausgedruckte Flyer (wie vom Linuxtag)
- USB–Stick mit einer tar–Datei
- Software
  - amforth–Baum (releases/4.2), am4up, amforth–upload.py
  - arduino–Projekt und Demo–Programm
  - avra–Sourcen
- Dokumentation
  - fosdem slides + proceedings[5]
  - atmega328p–Datenblatt
  - amforth–Referenzkarte
  - amforth–Flyer

Erich hat die tar–Datei auf dem Workshop an die Teilnehmer verteilt. Die Teilnehmer rangierten von "noch nie was von Forth gehört" (einer hat sich danach geoutet) bis hin zu Leuten, die das längst in den Fingern hatten und die Sachen genau wissen wollten. Auch der Entwickler von uBasic für AVR[6] war da.

Jeder Teilnehmer bekam ein Arduino–Duemilanove–(2009–)–Board, welches Erich noch am Morgen vor dem Workshop schnell mit amForth bespielt hat. Dazu gab es für jeden Teilnehmer ein 'Danger–Shield'–Erweiterungsboard (siehe Abbildung). Das 'Danger–Shield' enthält neben LEDs noch drei Analog–Schieberegler, eine 7–Segment–Anzeige, Temperatur–, Licht– und Erschütterungs–Sensor, drei Druckknöpfe sowie einen Buzzer.

Erich hatte die 10 'Danger–Shields' als Bausätze bestellt und sauber zusammengelötet. Um die 'Danger–Shields' nach dem Löten zu testen, hatte Erich ein Test–Programm in amForth geschrieben. Dieses nahm er als Vorlage im Workshop, um zu erklären, wie der Ardiuno unter Forth programmiert wird und wie die Sensoren und Funktionen des 'Danger–Shields' angesprochen werden.

Die Kursteilnehmer (in Zweier–Gruppen) haben via USB–Serielle–Schnittstelle mit dem Arduino–Board kommuniziert und das Test–Programm ausprobiert. Die Zeit des Workshop (2 1/2 Stunden) reichte aus für eine Kurzeinführung in Forth, aber nicht um einen kompletten Forth–Kurs in der Zeit unterzubringen. Es bleibt zu hoffen, dass die Teilnehmer jedoch Spaß an Forth und dem Arduino gefunden haben und sich weitere Informationen besorgen.

Die Teilnehmer konnten das Arduino–Board aus dem Workshop für 29 Euro (inkl. USB–Kabel) mit nach Hause nehmen, 6 Teilnehmer machten von diesem Angebot Gebrauch.

Stand

Der Stand der Forth–Gesellschaft war am Samstag sehr gut besucht, auch wenn (gefühlt) jeder zweite Besucher am Stand sagte "Ach, Forth! Das hab ich mal vor 25 Jahren gemacht. Das gibt's immer noch?" :) Wir haben auch einige ehemalige Mitglieder der Forth–Gesellschaft getroffen und hoffentlich motiviert, sich wieder einmal mit Forth zu beschäftigen. Laut Info–Stand gab es auf den Chemnitzer–Linuxtagen ca. 1200 Besucher. Die Resonanz war subjektiv größer als auf dem Linuxtag in Berlin, der locker auf die zehnfache Menge an Teilnehmern kommt. In Chemnitz waren mehr ‚aktive,' d.h. Personen, die ‚machen,' anstatt nur zu ‚schauen und reden.'

Ein paar Teilnehmer des Workshops im Sublab Leipzig (09/2010) haben unseren Stand am Freitagmorgen besucht. Sowas tut gut. Am Sonntag war das Interesse am Stand leicht geringer, da die meisten Teilnehmer für beide Tage angereist sind und schon am Samstag unseren Stand besucht hatten.

[1] http://www.chemnitzer.linux-tage.de/2011

[2] http://chemnitzer.linux-tage.de/2011/vortraege/709

[3] http://www.arduino.cc/en/Main/Hardware

[4] http://www.sparkfun.com/products/10115

[5] fosdem slides + proceedings:
http://amforth.sourceforge.net/pr/Fosdem2011-slides-amforth.pdf
http://amforth.sourceforge.net/pr/Fosdem2011-proceedings-amforth.pdf

[6] http://www.mikrocontroller.net/articles/AVR_BASIC

Delister
Aus dem Listing des Assemblers die assemblierbare Quelle zurückgewinnen.

Michael Kalus

Bei der Arbeit mit dem amforth an meiner privaten Heizungs- und Solaranlagensteuerung entstand die Situation, dass durch Änderungen im Forthkern (core words) die Versionsverwaltung sehr unübersichtlich geworden war. Weil amforth aus hunderten von kleinen Einzeldateien besteht, die in die Anwendung geladen werden (include files), ist es praktisch unmöglich, mit einem Werkzeug wie DIFFMERGE oder Ähnlichem die Unterschiede der einen zu der anderen Version zu finden. Solche Vergleiche kann man bequem nur zwischen zwei Dateien stattfinden lassen, aber nicht zwischen hunderten eingeschlossener Dateien. Also muss die komplette Quelle der Anwendung in nur eine Datei versammelt werden. Die kann dann mit der nächsten Version davon verglichen werden.

Das AVRA-Listing heizung.lst

Der Assembler AVRA ist schon so nett und versammelt alles Assemblierte in einer einzigen Datei. Dateien, die nicht includiert worden sind, werden auch nicht gelistet, von IF .. ELSE .. ENDIF werden auch nur die Teile eingeschlossen, die für die Version benutzt werden, usw. Also genau das, was man braucht, ist im Listing schon drin — siehe Beispiel–1. Aber das Listing dient vor allem dazu, dass ein prüfender Leser erkennen kann, an welchen Adressen der Assembler welchen Code erzeugt hat. Das Listing kann daher in dieser Form nicht erneut durch den Assembler geschickt werden. Es muss befreit werden von den Adress- und Codeangaben. Und auch die .include–Directiven dürfen nicht noch mal ausgeführt werden, es ist ja schon alles in der Datei versammelt, was dazu gehört. Auch die Macros dürfen kein zweites Mal expandiert werden. Dann ist da noch eine kleine Hässlichkeit des Listings zu beachten. Die Instruktionen im Code–Segment cseg werden etwas anders gelistet als das, was an Daten ins EEPROM–Segment geschrieben worden ist. Im meinem Beispiel entsteht aus dem amforth schließlich ein Listing von 17102 Zeilen, heizung.lst genannt. Und da möchte man nicht mehr von Hand ran gehen und es in eine 1–Datei–Quelle zurückverwandeln, obwohl für das menschliche Auge leicht zu erkennen ist, was zu dem Zweck wegmüsste.

parsen in gforth

So ist es also an der Zeit, diesen Text maschinell zu parsen. Im Prinzip ganz einfach. Man öffne die Datei heizung.lst, lese eine Zeile, prüfe deren Aufbau und entscheide, was damit geschehen soll, und schreibe das Ergebnis hin. Und so ist es auch ein one–pass delister geworden. Die Regeln werden vom checktype durchgeführt, das als Eingang die Adresse und Länge der zu prüfenden Zeichenkette erhält. Je nachdem wird die Zeile einfach verworfen, gekürzt, oder in einen Kommentar verwandelt. Zur Hilfe kam mir, dass im gforth ein passender parser schon da ist. Das Forthwort search sucht nach einem Ausdruck in einer Zeichenkette. Damit waren schon viele typische Zeilen zu erkennen und damit behandelbar. Lediglich das parse–bl musste neu erfunden werden, um eine Kommentarzeile am ; zu erkennen, das auf Leerzeichen des Zeilenanfangs folgt. Hilfreich war es, dass gforth konsequent alle Zeichenketten über (adr u —) auf dem Stack handhabt. Da konnte ich mich ganz gut reinfinden und somit diese parse areas handhaben.

file handling in gforth

Zentral ist es dabei, den folgenden Teil verstanden zu haben, den man aber glücklicherweise nicht erfinden, sondern nur abzuschreiben braucht.

```
0 Value fid-in
: open-input ( addr u -- )
    r/o open-file throw to fid-in ; mk

: close-input ( -- )
    fid-in close-file throw ;

: getline ( -- adr u f )
    linebuffer maxline fid-in
    read-line throw ( -- u f )
    linebuffer -rot ;
```

Hat man im Handbuch des gforth erst einmal diese Stelle gefunden, kommt man zurecht, finde ich. Auch dabei ist das Prinzip eingehalten worden, alle Daten per Adresse und Längenangabe zu handhaben. Womit parsen in Text–Dateien schließlich recht einfach wird.

Zum Gebrauch des Delister

An einer Stelle ist noch etwas Handarbeit nötig. AVRA schreibt Sätze in das Listing, die nicht in das Schema der ganzen Auswertung passen, eine Überschrift und vier Schlusssätze. Da diese Sätze ganz am Anfang und am Ende des Listings stehen, kommentiere ich die einfach von Hand aus, und fertig. Dieses heizung.lst genannte Listing wird dann vom delister in die Datei heizung.re.asm umgeschrieben. Nun kann dieses frisch gewonnene heizung.re.asm assembliert werden. Und AVRA gab das

[0] Gforth Manual; 5.17.2 General files; Seite 112.

ersehnte "Assembly complete with no errors". Den schlüssigen Beweis liefert dann der Vergleich der Images mit DIFFMERGE, ein Programm zum Vergleich von Dateien. Die von AVRA erzeugten Images, das originale heizung.hex und unser heizungs.re.hex, waren identisch. Möge es nützlich sein.

Links

http://dl.dropbox.com/u/1170761/delister.fs
http://www.jwdt.com/~paysan/gforth.html
http://avra.sourceforge.net/
http://www.atmel.com/

Beispiel–1: Auszug aus dem vom Assembler erzeugten Listing.

```
1   ...
2              .include "words/create.asm"
3             ; ( -- ) Dictionary
4             ; R( -- )
5             ; create a complete dictionary header.
6              VE_CREATE:
7                  .dw $ff06
8   C:000a0e ff06
9                  .db "create"
10  C:000A0F 637265617465
11                 .dw VE_HEAD
12  C:000a12 0a04
13                 .set VE_HEAD = VE_CREATE
14             XT_CREATE:
15                 .dw DO_COLON
16  C:000a13 1c0a
17             PFA_CREATE:
18                 .dw XT_DOCREATE
19  C:000a14 0131
20                 .dw XT_COMPILE
21  C:000a15 016c
22                 .dw PFA_DOCONSTANT
23  C:000a16 1c69
24                 .dw XT_EXIT
25  C:000a17 1c37
26  ...
```

Dem gegenüber das originale create.asm

```
1
2   ; ( -- ) Dictionary
3   ; R( -- )
4   ; create a complete dictionary header.
5   VE_CREATE:
6       .dw $ff06
7       .db "create"
8       .dw VE_HEAD
9       .set VE_HEAD = VE_CREATE
10  XT_CREATE:
11      .dw DO_COLON
12  PFA_CREATE:
13      .dw XT_DOCREATE
14      .dw XT_COMPILE
15      .dw PFA_DOCONSTANT
16      .dw XT_EXIT
```

delist.fs forth source code

```
1   \ heizung.lst Datei in Assembler-Quelle zurueckfuehren.
2
3   : .. bye ;
4   : mk ; \ comment
5
6
7   \ misc
8   : .mark   ( -- ) ." ; +++ " ;
9   : .header ( adr u -- ) cr cr .mark type ."  re-listing" cr ;
10  : .footer ( -- ) cr .mark ." end of re-listing" ;
11
12  \ allocate a buffer for character handling
13  256 Constant maxline
14  Create linebuffer maxline 2 + allot
15
16  \ file handling
17  0 Value fid-in
18  : open-input ( addr u -- ) r/o open-file throw to fid-in ; mk
19  : close-input ( -- ) fid-in close-file throw ;
20  : getline ( -- adr u f )
21      linebuffer maxline fid-in read-line throw ( -- u f )
22      linebuffer -rot ;
23
```

```
24
25  \ check content of lines.
26  : parse-bl { adr1 u1 -- adr2 u2 } \ Erstes nicht-bl-Zeichen.
27    u1 0 do
28          adr1 c@  bl <> if leave then
29          adr1 1+ to adr1 u1 1- to u1
30      loop adr1 u1 ;
31  : ;line? ( adr u -- f ) \ Ist es eine Kommentarzeile?
32    parse-bl drop c@  [char] ; = ;
33
34  : .db? ( adr u -- f )  s"   .db" search ( -- adr u f )  >r 2drop r> ;
35  : .dw? ( adr u -- f )  s"   .dw" search ( -- adr u f )  >r 2drop r> ;
36
37  : macro? ( adr u -- f )  drop  11 + c@ [char] + = ;
38  : macro  ( adr1 u1 -- adr2 u2 )  13 - swap 13 + swap ." ; macro: " ;
39
40  : C:00? ( adr u -- f ) s" C:00" search >r 2drop r> ;
41  : stripC:00 ( adr1 u1 -- adr2 u2 )   13 - swap 13 + swap ;
42
43  : .inc? ( adr u -- f ) s" .include" search >r 2drop r> ;
44  : .if?  ( adr u -- f ) s" .if" search >r 2drop r> ;
45  : .else? ( adr u -- f ) s" .else" search >r 2drop r> ;
46  : .endif? ( adr u -- f ) s" .endif" search >r 2drop r> ;
47
48  : -type  ( adr u ) dup 5 > if 5 - swap 5 + swap then type ;
49  : skipline ( -- )  getline 2drop drop ;
50
51  variable segflag  \ Ab dem EEPROM-Segment nichts mehr skippen.
52  : seg?  ( -- ) segflag @ ;
53  : .eseg? ( adr u -- f )  s"    .eseg" search >r 2drop r> ;
54
55  \ delist file
56  : checktype ( adr u -- )
57      2dup .inc?   if 2drop           exit then
58      2dup .if?    if 2drop           exit then
59      2dup .else?  if 2drop           exit then
60      2dup .endif? if 2drop           exit then
61      2dup macro?  if macro type     exit then
62      2dup C:00?   if stripC:00 -type exit then
63      2dup ;line?  if -type           exit then
64      2dup .db?    if -type  seg? if skipline then   exit then
65      2dup .dw?    if -type  seg? if skipline then   exit then
66      2dup .eseg?  if -type  false segflag !  exit then
67      -type ;
68
69  : delist ( adr u -- )
70    2dup open-input .header  true segflag !
71      begin getline while ( -- adr u )
72          checktype
73      cr repeat
74    2drop close-input .footer ;
75
76  : go  s" heizung.lst" delist bye ;
77
78  ( finis)
```

RSC–FORTH V1.7 — vintage computing

Michael Kalus

Neulich stolperte ich auf dem Dachboden über einen verstaubten Umzugskarton mit der Aufschrift „Keller.“ Weil grad etwas Zeit war öffnete ich den Karton und fand darin einige Platinen, darunter ein fein säuberlich verpacktes RSC–Forth Board. Ob es noch laufen würde? Aber da kein Anlass bestand, packte ich das Teil wieder ein, zurück in den Karton.

Zufall

Wie der Zufall es so wollte, traf ich auf Carsten Strotmann, der gerade vom Vintage Computer Festival in München zurückgekommen war. Er berichtet von dem Spaß dort, die alten Rechner ans Laufen zu bringen. Davon angeregt, wollte ich es dann doch wissen, ging wieder auf den Dachboden und ans Werk. Die RSC–Platine sah noch sehr gut erhalten aus, keinen Kratzer hatte sie. Auch das separate Netzteil von damals, Marke Eigenbau, sah gut aus. Es war schnell aufgebaut und angeschlossen, die rote LED auf der Platine zeigte an, dass die Spannung da war, aber auf der seriellen Schnittstelle tat sich nichts.

Hilfe

Hilfe kam von einem Freund mit Speicheroszilloskop. Da war klar zu erkennen, dass alles da war: Die ROM–Adressen wurden abgefragt nach dem reset, der Quarz lief, und auf der seriellen Leitung waren tatsächlich Pegelwechsel zu erkennen. Eingefangen, zeigten sie eine Bit–Dauer, die auf Baud–Rate 2400 hinwies, und bei näherer Betrachtung wurde klar, dass 7–N–2 eingestellt werden musste am Terminal. Und siehe da, das ersehnte OK war da. Nach reset kam brav die Meldung RSC–FORTH V1.7 und VLIST lieferte die altgewohnte Wordlist, ASSEMBLER VLIST ebenso. Also alles da und intakt. Nun wurde erstmal der Forth–Kern ausgelesen. Zum einen als

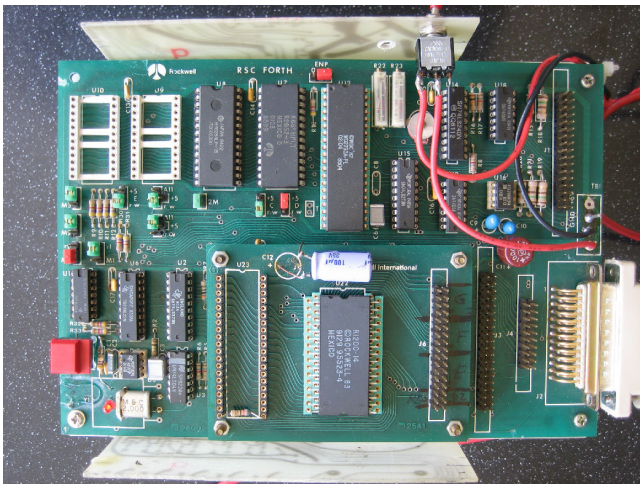
```
HEX F400 0C00 DUMP
```

und zum anderen als

```
HEX F400 FFFF ADMP
```

Letzteres ist das Intel–Hex–Format, das alle Prommer lesen können. Freundlicherweise ist das ADMP ja schon im Entwicklungs–ROM des Forth mit drin.

Fotos wurden gemacht, das VLIST gefilmt. Was man so alles kann heutzutage mit der Digitalkamera. Auch die Ausgaben des RSC–Forth in eine Datei zu sichern, alles auf USB–Stick an den Apple–Laptop zu transferieren — all das, was man damals sich wünschte, es ist heute Realität. Die Textfiles lassen sich bequem austauschen, Bilder betrachten, und ins Internet stellen, per mail übermitteln. Und es wird auch klappen, auf dem Mac ein gforth zu starten, dort hinein das RSC–Forth zu laden, zu decompilieren und zu simulieren. Eigentlich eine unglaubliche Leistung, die man kaum ermessen kann, wenn man mit Rechnern groß geworden ist, auf denen Windows 7 betrieben wird, Mac OSX und Linux und dergleichen, diese fetten Betriebssysteme die (fast) alles können. Diese Micros, wie der 6502 und Forth darin, waren die Anfänge davon. Man hat es geahnt, was damit so alles kommen könnte, von manchem geträumt, und auf etliches dann mit hingearbeitet.



Links

http://forth-ev.de/filemgmt/singlefile.php?lid=372

Nostalgiepaket

Nun, nostalgisch Angehauchte finden das so Geborgene nun im Internet, in meiner drop–box — auch so ein Ding, das vor noch nicht allzu langer Zeit nur Science Fiction gewesen ist. In dem Archiv `RSC_Forth_V1.7.zip`, das dort nun liegt befinden sich folgende Dateien:

```
6502_FIG_Forth.ASM ( Assembler Quelle des FIG Forth zum Vergleich )
6502_Forth_Assembler.fs ( Forth Quelle des Assemblers )
6502_RSC_Forth.ASM ( Assembler Quelle des RSC Forth )
R1200-14_Rockwell.jpg ( Bild des Prozessors )
R65FRIP1.7.jpg ( Bild des ROM )
RSC_FORTH_V17.bin ( Binäres Image des ROM )
RSC_FORTH_V17_VLIST.txt ( Textausdruck des FORTH VLIST )
RSC_FORTH_V17_VLIST_ASSEMBLER.txt ( Textausdruck des ASSEMBLER VLIST )
RSC_FORTH_V17_admp_0x2000_0x3FFF.txt ( Intel Hex Format )
RSC_FORTH_V17_admp_0xF400_0xFFFF.txt ( Intel Hex Format )
RSC_FORTH_V17_dump_0x2000_0x3FFF.txt ( Forthtypisches hex dump )
RSC_FORTH_V17_dump_0xF400_0xFFFF.txt ( Forthtypisches hex dump )
RSC_Forth_V17_readme.txt ( Erklärungen )
```

Viel Vergnügen damit.

VLIST

Zur Erinnerung hier mal das VLIST-Ergebnis. Der Forthkern passte in F400 bis FFFF also in 3071 Bytes, und das Entwicklungs ROM belegte die Adressen 0x2000 bis 0x4000, also 8192 bytes — cool, oder?

```
COLD
RSC-FORTH V1.7
VLIST
  40B TASK          3852 ADMP          3813 ;DUMP         37DB FORMAT
 368A FMTRK         3680 BANKEXECUTE   3670 BANKEEC!      3663 BANKC@
 3658 BANKC!        364D EEC!          362A CASE:         360A MEMTOP
 35FA SCDR          35EC SCSR          35DE SCCR          35D0 MCR
 35C3 IER           35B6 IFR           35A9 PG            359D PF
 3591 PE            3585 PD            3579 PC            356D PB
 3561 PA            3555 NMIVEC        3545 IRQVEC        3535 INTVEC
 3525 INTFLG        34FC C,CON         34B9 .S            34AC MON
 3468 VLIST         33F9 INDEX         33AD LIST          33A4 ?
 339E .             3398 .R            3391 D.            338A D.R
 3382 #S            337B #             3375 SIGN          336C #>
 3365 <#            335E SPACES        334B WHILE         3327 ELSE
 330E IF            32F5 REPEAT        32DC AGAIN         32CC END
 32B6 UNTIL         329E +LOOP         3286 LOOP          3271 DO
 3264 THEN          3247 ENDIF         3233 BEGIN         3192 FORGET
 3156 AUTOSTART     311D ?KERNEL       30C9 HWORD         3093 H/C
 307B '             3075 SEEK          306C INIT          3063 DWRITE
 3058 DREAD         304E SELECT        3043 DISK          3030 R/W
 3024 B/SCR         3016 B/BUF         2FFA -BCD          2FD6 -->
 2FA6 LOAD          2F4D MESSAGE       2F1C >LINE         2F08 .LINE
 2EE4 (LINE)        2EA1 DUMP          2E76 FLUSH         2E16 BLOCK
 2DCC BUFFER        2DA7 EMPTY-BUFFERS  2D7F UPDATE        2D4E +BUF
 2D45 M/MOD         2D3B */            2D34 */MOD         2D2A MOD
 2D22 /             2D1C /MOD          2D13 *             2D0D M/
 2D06 M*            2CFF MAX           2CF7 MIN           2CEF DABS
 2CE6 ABS           2CDE D+-           2CD6 +-            2CCF S->D
 2CC6 COLD          2C59 ABORT         2C2A QUIT          2C18 (
 2C06 DEFINITIONS   2BEE ASSEMBLER     2BD6 FORTH         2BA4 VOCABULARY
 2B8A IMMEDIATE     2B3A INTERPRET     2B0F ?STACK        2AF2 DLITERAL
 2ABF LITERAL       2AA1 [COMPILE]     29ED CREATE        29C4 ID.
 2987 ERROR         2973 (ABORT)       2945 -FIND         28ED NUMBER
 28E2 (NUMBER)      2890 WORD          2887 HOLD          287E BLANKS
 2873 ERASE         2869 FILL          2842
```

Recognizer – Interpreter dynamisch verändern

Matthias Trute

Der Artikel beschreibt ein Konzept, um applikationsspezifische Daten mit einem modifizierbaren Interpreter zu verarbeiten. Dabei werden spezialisierte Parsing Words aus der jeweiligen Applikation über eine Standard-API dynamisch einbezogen.

Anlass und Idee

Amforth ist ein Forth für (sehr) kleine Systeme. Nichtsdestotrotz gibt es eine Gleitkommazahlbibliothek. Um sie nutzen zu können, sollte man die Gleitkommazahlen auch am Kommandoprompt eingeben können. Soweit so einfach. Nur, wie bringt man dem bestehenden Interpreter bei, Gleitkommazahlen zu verarbeiten? Der Unterschied zwischen „1000“ und „1E3“ ist eine Herausforderung nicht nur für den Parser. Es werden auch verschiedene Daten auf dem Stack abgelegt: 1 Zelle für den Integer und 2 Zellen für das Float.

Die auf den ersten Blick einfachste Methode ist, das passende *parsing word* aus der Gleitkommabibliothek in den Code des Interpreters zu integrieren, ein neues System zu generieren und auf den Controller zu übertragen.

Das erfordert Anpassungen am Quellcode von amforth, was, wie sich herausgestellt hat, eine große Hürde sein kann.

Besser wäre eine dynamische Erweiterung des laufenden Interpreters, die nach einigen wenigen Codezeilen die neuen Eingabeformate *native* nutzbar macht.

Der erste Schritt ist die Erkenntnis, dass die neuen Formate dem Interpreter zunächst unbekannt sind. Er wird sie nicht im Dictionary finden und als (Integer-) Zahl sind sie auch nicht erkennbar.

Amforth bietet schon lange die Not-Found-Exception, die der Interpreter generiert, sobald ein Wort nicht verarbeitet werden kann.

Es ist jedoch nicht trivial, die Reaktion auf die Exception so zu gestalten, dass der Interpreter mit Sonderaktionen reagiert und danach weitermacht, als wäre nichts gewesen. Methodisch ist es zudem wenig elegant, einen Fehlermechanismus für einen normalen Ablauf zu nutzen.

Ulrich Hoffmann hat auf der EuroForth 2008 ein System vorgestellt, das eine Vielzahl redefinierbarer Stellen im Interpreter vorsieht. Diese kann man umbiegen und so auf alle Eventualitäten reagieren. Auf den ersten Blick sehr reizvoll. Wenn man aber an die Details geht, kommt die Ernüchterung. So richtig *einfach* ist das auch nicht.

Es ist eine Lösung gefragt, mit der man einen Interpreter erweitern kann, so dass er Worte in einer neuen Syntax verarbeiten kann. Da Mikrocontroller notorisch knapp an allen Ressourcen sind,heißt das zudem, dass das Kernsystem nicht größer werden darf. Wer die Flexibilität nicht braucht, soll nicht bestraft werden. Unterschiedliche Codeabschnitte mit bedingter Assemblierung sind ebenfalls unzweckmäßig, da der Testaufwand steigt.

Das nachfolgend vorgestellte Konzept wird in amforth ab Version 4.3 umgesetzt und heißt Recognizer[1]. Die grundlegende Idee hierfür basiert auf einem Posting in der Usenetgroup `comp.lang.forth`, in dem Anton Ertl *Number Parsing Hooks* vorstellt: `https://groups.google.com/group/comp.lang.forth/msg/f70a9ea205b5b75a`

> Essentially the program has to provide a word (let's call it a recognizer) that takes a string (for the „word“ that was not found), and returns a flag that indicates whether the string was recognized or not. In addition, the word may do things to the stack below the string/flag (e.g., push literal numbers) or compile things (e.g., literal numbers). So, such a recognizer would have the stack effect:
> (i*x c-addr u – j*x f)
> [Yes, let's not design counted strings into this interface]
> In addition, the interface should support stacking of recognizers, so that one library can provide support for recognizing time syntax, while another library provides support for complex numbers. The program should be able to specify in which order the recognizers should be processed (for cases where the recognized syntax overlaps).

Seine Ideen habe ich auf die bestehenden Strukturen im Interpreter verallgemeinert[2] und den Interpreter zu einem generischen Werkzeug umgebaut, das wesentliche Teile der Arbeit den Recognizern überlässt.

Ein Recognizer analysiert das vorliegende Wort und versucht, es zu „verstehen“. Ist das erfolgreich, wird das Wort komplett verarbeitet und dem Interpreter signalisiert, dass dieser mit dem nächsten Wort fortfahren kann. Anderenfalls wird der nächste Recognizer aus der Liste aufgerufen.

Die Idee ist ähnlich zur SEARCH ORDER. Auch dort wird eine Reihe von word lists bis zum ersten Treffer abgearbeitet.

Zum Standardfunktionsumfang zählen drei Recognizer: `rec-find`, `rec-intnum` und `rec-notfound`. Was die ersten beiden machen, sollte unmittelbar einleuchten. Letzterer wird das Word im Eingabedatenstrom nicht verstehen, aber einen geordneten Ausstieg ermöglichen, indem er die Exception NOT-FOUND (-13) wirft.

[1] Ja, amforth kann nicht nur Assembler, der Eindruck mag entstanden sein.

[2] Die Diskussion ging damals nur um Zahlenpräfixe

Umsetzung

Die Umsetzung ist erstaunlich einfach. Der bereits existierende Interpretercode wird in seine Elemente zerlegt und in eine Iteration mit Abbruchbedingung über einer Liste umgebaut.

Ein „normaler“ Interpreter wird etwa wie folgt aussehen[3]:

```
: interpret ( -- )
  BEGIN
    BL WORD DUP COUNT DUP C@
  WHILE ( c-addr )
    FIND ?DUP
    IF OVER STATE @ 0<> =
      IF COMPILE, ELSE
      EXECUTE THEN
    ELSE
      COUNT NUMBER? ?DUP IF 0< IF ?literal
      ELSE
        notfound
      THEN
    THEN
  REPEAT
  DROP ;
```

Hier ist die Reihenfolge der Elemente FIND und NUMBER unveränderbar festgelegt. Demgegenüber steht der folgende Code.

```
: interpret
  BEGIN
    \ Nur Worte mit mind. 1 Zeichen auswerten.
    BL WORD DUP C@ 0> IF
      \ EE_RECOGNIZER verweisst auf ein Array
      \ im EEPROM: Erstes Feld ist die Länge des
      \ Arrays, danach folgen die einzelnen Execution
      \ Tokens der Recognizerworte
      EE_RECOGNIZERS DUP @E 0 ?DO
        \ Auf dem Datenstack darf nichts ausser
        \ dem Zugang zum Wort stehen, alles andere
        \ muss weg, damit der Recognizer freie Bahn
        \ hat
        OVER >R CELL+ DUP >R
        \ Execution Token des Recognizers lesen
        \ und ausführen
        @E EXECUTE
        \ Rückkehrcode prüfen und entweder
        \ weiter mit dem nächsten Wort ...
        IF R> R> DROP DROP LEAVE THEN
        \ ... oder mit dem nächsten
        \ Recognizer.
        R> R> SWAP
      LOOP
      \ Housekeeping
      ?STACK
    REPEAT
    DROP ;
```

Dieser Interpreter zerlegt den Input in einzelne Worte[4] und arbeitet anschließend eine Liste von Aktionen ab. Was die Aktionen machen, ist für den Interpreter nicht wichtig, er muss nur wissen: Hats geklappt oder nicht.

Das heißt bei Worten aus dem Dictionary (FIND), dass das Execution Token ausgeführt bzw. compiliert wird. Analog muss der Integer-Recognizer den Wert auf dem Stack hinterlassen bzw ins Dictionary compilieren. In jedem Fall wird ein Flag zurückgegeben, das den Erfolg oder Misserfolg kommuniziert.

Beispielhaft der FIND Recognizer.

```
: rec-find
    \ Suche in allen Wordlisten
    FIND
    ?DUP IF
      \ gefunden, jetzt verarbeiten
      \ immediate?
      0> IF EXECUTE
      ELSE
        STATE @
        IF COMMA, ELSE EXECUTE THEN
      THEN
      \ Signal für den Interpreter:
      \ habs gemacht
      -1
    ELSE
      \ Aufräumen und Signal:
      \ Habs nicht gemacht
      DROP 0
    THEN ;
```

Einfacher ist der Not-Found-Recognizer. Da er eine Exception generiert, muss er keinen eigentlichen Returncode bereitstellen[5]. Das funktioniert, da der Interpreter keine Exceptions auffängt, sondern dies der darüberliegenden Schicht überlässt (QUIT).

```
: rec-notfound
    COUNT TYPE
    -13 THROW
;
```

Im Terminal sieht das dann so aus

```
> ver cr 1 2 + . cr huhu
amforth 4.4 ATmega16
3
huhu ?? -13 23
>
```

Der Text `?? -13 23` stammt vom Exception Catcher in `quit`, der die beiden Fragezeichen, die Exceptionnummer und die Position im durch `source` bezeichneten Buffer ausgibt, an dem das Problem erkannt wurde (bei -13 ist das die Stelle nach dem betreffenden Wort).

Die eingangs erwähnte Gleitkommabibliothek enthält ein Wort `>float`, mit dem sich eine Zeichenkette in ein Float konvertieren lässt und obendrauf ein Flag über den Erfolg der Aktion liefert. Damit wird der Recognizer `rec-float` sehr einfach:

```
> 123e4 fs.
123e4 ?? -13 6
> : rec-float count >float
 ok if state @ if postpone fliteral then -1 else 0
 ok then ;
 ok
```

[3] Quelle: Hoffmann, Euroforth2008

[4] Aus pragmatischen Gründen basiert die Implementierung noch auf Counted Strings und `word`, das soll sich jedoch noch ändern.

[5] Eigentlich liefert der Recognizer immer -1 aka TRUE zurück.

```
> ' rec-float place-rec
ok
> 123e4 fs.
1.2299999E6 ok
>
```

Verwaltung

Wie werden die Recognizer verwaltet? Da die Idee nicht auf kleine Systeme beschränkt bleiben muss, ist auch die Implementierung als Array im EEPROM alles andere als zwingend.

Hierfür habe ich zwei Worte vorgesehen: `set-recognizer` und `get-recognizer`. Beide stehen in Analogie zu `set-order` und `get-order`. `Get-recognizer` hinterlässt genau wie `get-order` eine Liste samt Anzahl der Elemente auf dem Datenstack. Diese Liste wird mittels `set-recognizer` gespeichert und unmittelbar aktiviert. Eine portable Version des obigen INTERPRET-Codes wird natürlich hierauf aufsetzen.

Die konzeptionelle Ähnlichkeit zum Search Order Wordset lässt weitere Worte in Analogie zu diesem sinnvoll erscheinen: `order-` und `only`-Analoga fallen zuerst ein. Welche konkret nützlich sind, wird hoffentlich eine Diskussion ergeben.

Ein derartiges Wort ist sicher das bereits benutzte `place-rec`. Es baut den übergebenen Recognizer unmittelbar *vor* dem letzten derzeit aktiven ein. Damit bleibt die Recognizerliste inklusive dem abschließenden not-found intakt.

```
: place-rec ( xt -- )
    get-recognizer
    \ Anzahl sichern
    dup >r
    \ Alle außer dem Letzten weg
    1-  n>r
    \ den neuen einbauen
    swap
    \ und alles wieder aufbauen
    nr> drop r> 1+
    set-recognizer
;
```

Ein weiterer Punkt ist der `rec-notfound`-Recognizer selbst. Man kann darüber diskutieren, ob er nicht als Standard-Aktion *immer* an das Ende der Liste angefügt wird. Die gegenwärtige Lösung lässt hingegen alle Freiheit.

Spielereien

Die Idee des Recognizers erlaubt Anwendungen, die zunächst erst einmal die Portabilität des Sourcecodes beeinträchtigen.

```
wordlist constant foo
... definiere wort(e) in foo
wordlist constant bar
... definiere wort(e) in bar
```

Ein Recognizer ist darauf spezialisiert, anhand des Schemas `<wordlist-id>::<wortname>` die Zuordnung der Worte in den entsprechenden Wordlists zu finden und zu verarbeiten.

- `foo::wort` Suche in der Wortliste identifiziert durch `foo` nach dem Wort `wort` und verarbeite es
- `bar::wort` Suche das Wort `wort` in der Wortliste `bar` und verarbeite es.

Syntaktische Feinheiten, wie `$foo->wort,` wobei `foo` eine Variable ist, die auf die zu nutzende Wordlist verweist, werden bei Forthianern sicher Schaudern hervorrufen, aber OO-Anhänger, die von anderen Sprachen kommen, begeistern...

Ein anderer Einsatzfall ist das Verarbeiten von vielen Zahlen bei wenig Code (Tabellen). Vertauscht man die beiden Recognizer für FIND und INTNUM, geht das spürbar schneller, da die nutzlose Suche im Dictionary entfällt.

Ein dritter denkbarer Einsatzfall könnten die zahlreichen neuen Worte aus dem Memory-Access-Standard sein, die regelbasiert gebildet, aber wohl nur selten im vollen Umfang implementiert werden. Hier ist eine Wort-Factory hilfreich, die als Recognizer eingebunden werden könnte.

Und wer weiß, vielleicht lernt auch gforth irgendwann, dass die IP Adresse des `www.forth-ev.de` Servers (85.214.243.249, 3 Punkte in einer Zahl) nicht den numerischen Wert 85214243249. (double cell), als vielmehr 1440150521 (passt in eine 32bit Zelle) hat...

Finale

Das vorgestellte Konzept ist in Gänze im (aktuellen) amforth implementiert und funktioniert. Die Codegröße hat sich durch den Umbau nicht verändert, einzig der Platzbedarf im EEPROM für die Recognizerliste ist hinzugekommen.

Mein Dank geht an alle, die mit mir in den letzten Monaten über das Thema im Chat #forth-ev diskutiert haben, Feedback gaben oder einfach nur zugehört haben. Nicht alle werden so wirklich verstanden haben, was ich mit meinen Fragen bezweckt habe, jetzt werden sie es hoffentlich wissen.

Adventures in amForth 7: Eine einfache Liste

Erich Wälde

Auf meinen Messstationen wollte ich die Information über die angeschlossenen Sensoren so verwalten, dass ich jederzeit neue Sensoren dazuhängen kann, ohne den Kontroller neu zu programmieren. Ich wollte eine einfach verkettete Liste, auch wenn ein Vektor mit zehn Einträgen wahrscheinlich weit gereicht hätte. Ich habe mich für eine sehr simple, einfach verkettete Liste entschieden. Der gezeigte Quelltext bezieht sich auf amForth 4.2.

Paarweise Zellen

Der Eintrag in einer einfach verketteten Liste besteht nur aus zwei Feldern:

1. die Adresse des nächsten Eintrags (`next`)
2. der zu speichernde Wert (`value`)

Das Ende der Liste markiere ich, indem das erste Feld den Wert null erhält. Der Kopf (Anfang) der Liste (`head`) wird in einer Variablen aufbewahrt, am besten in einer nicht–flüchtigen Variablen vom Typ `value`.

```
0 value LISTE   \ define head
```

Um einen Eintrag am Anfang der Liste einzufügen, definiere ich das Wort `list.add`, welches auch *unshift* heißen könnte.

```
: list.add ( value head -- new.head )
  dp >r          \ save new.head
  ,              \ store head as "next"
  ,              \ store "value"
  r>             \ new.head on stack
;
```

An der nächsten freien Adresse im Flash-Speicher (`dp`) werden der Zeiger auf den nächsten Eintrag und der zu speichernde Wert eingetragen. Der Wert kann natürlich ebenfalls eine Adresse sein, z.B. die Adresse einer Funktion oder eines weiteren Datenfelds. Die beiden Felder sind von der Größe `cell`, also 2 Byte.

Die Adresse des Eintrags verbleibt auf dem Stack und kann dann in `head` eingetragen werden.

Vorsicht: Die Worte `dp` und `here` heißen in den amForth–Versionen vor 4.0 `here` und `heap`.

Im ersten Eintrag der Liste soll die Zahl `$0042` gespeichert werden:

```
> 0 value LISTE   LISTE hex .
0  ok
> $0042 LISTE list.add  to LISTE
 ok
> LISTE .
11B9  ok
>
```

Das bei jedem Eintrag benötigte Quelltext–Fragment lässt sich auch schöner schreiben. Noch schöner wäre allerdings ein Wort, welches zum Namen eines Listenkopfs das zugehörige Wort definiert.

```
: >LISTE        ( value -- )
  LISTE list.add to LISTE
;
```

Um den Inhalt der Liste auszugeben, definieren wir drei Worte:

```
: list.next     ( addr -- addr' )
  i@
;
: list.value    ( addr -- value )
  1+ i@
;
: list.show     ( 'head -- )
  dup 0= if   drop exit   then
  begin
    dup              4 u0.r space \ flash addr
    dup  list.next   4 u0.r space
    dup  list.value  4 u0.r cr
  list.next dup 0= until
  drop
;

> LISTE list.show
11B9 0000 0042  ok
> $08FF >LISTE
 ok
> LISTE list.show
11BB 11B9 08FF
11B9 0000 0042
 ok
>
```

Diese Liste ist sehr einfach. Man kann nur am Anfang neue Werte hinzufügen und die ganze Liste ausgeben. Die Werte werden in umgekehrter Reihenfolge des Speicherns ausgegeben, was u.U. stört. Immerhin muss die Liste nicht am Stück im Speicher liegen. Man kann bequem noch ein paar Worte definieren, und danach weitere Einträge für die Liste.

Mit der Liste will man irgendwann etwas Konkretes anfangen, sonst bräuchte man sie nicht. Also schreibe ich eine Funktion, die einen Zeiger auf eine weitere Funktion und den Kopf einer Liste als Argumente bekommt und dann die übergebene Funktion auf alle gespeicherten Werte anwendet.

```
: list.walk     ( xt head -- )
  dup 0= if drop drop  exit  then
  begin          \ -- xt item
    over over    \ -- xt item  xt 'list.item
    list.value   \ -- xt item  xt value
    swap         \ -- xt item  value xt
    execute      \ -- xt item
    list.next    \ -- xt next
  dup 0= until
  drop drop
;
```

Die übergebene Funktion verbraucht den obersten Wert auf dem Stack (`value`), produziert aber keine Rückgabewerte.

```
> : doit ( value -- )  6 u0.r cr ;
 ok
> ' doit LISTE list.walk
0008FF
000042
 ok
```

Ausgerüstet mit diesen Worten, erzeuge ich eine Liste, in der die Werte weitere Adressen sind. Sie verweisen auf Datenstrukturen, in denen alles Wissenswerte über einen bestimmten Sensor zu finden ist, etwa wie man einen aktuellen Wert beschaffen kann, oder wie der beschaffte Wert weiterverarbeitet wird. Eine regelmäßig aufgerufene Funktion (`data.collect`) arbeitet dann die Liste ab. Die Werte werden im RAM abgelegt und nicht auf dem Stapel behalten. Kommen jetzt neue Sensoren an den Kontroller, dann erzeuge ich für jeden eine neue Datenstruktur und verlinke sie zusätzlich in der Liste der Sensoren. Mit dem nächsten Aufruf von `data.collect` werden auch die neuen Sensoren gelesen. Weitere Funktionen `data.ls` und `data.reset` versenden die gesammelten Daten bzw. löschen die entsprechenden Speicherbereiche.

Für eine ordentliche Liste fehlen weitere Funktionen, die einen bestimmten Eintrag aufsuchen, löschen, oder einen zusätzlichen Eintrag dazwischen verlinken. Für meine Anwendung habe ich das allerdings nicht gebraucht.

Referenzen

1. `http://amforth.sourceforge.net/`

Listings

```
\ 2011-01-14  EW  list.fs
\ amforth 4.0 or later
\
\ a list is a series of locations with
\ 2 cells each:
\ 1. a pointer to the "next" entry
\    next == 0 is end of list
\ 2. a value
\ the actual head of the list is kept
\ elsewhere, e.g. in a value.
\ 0 value List

\ words:
\   list.add    ( value head -- new.head )
\   list.next   ( addr -- addr' )
\   list.value  ( addr -- value )
\   list.show   ( head -- )
\   list.walk   ( xt head -- )

: list.add      ( value head -- new.head )
  dp >r         \ save new.head
  ,             \ store head as "next"
  ,             \ store "value"
  r>            \ new.head on stack
;

: list.next     ( addr -- addr' )
  i@            \ addr of next entry
;

: list.value    ( addr -- value )
  1+ i@         \ value
;

: list.show     ( head -- )
  dup 0= if drop exit then
  begin
    dup             4 u0.r space
    dup list.next   4 u0.r space
    dup list.value  4 u0.r cr
  list.next dup 0= until
  drop
;

: list.walk     ( xt head -- )
  dup 0= if drop drop  exit  then
  begin         \ -- xt item
    over over   \ -- xt item  xt item
    list.value  \ -- xt item  xt value
    swap        \ -- xt item  value xt
    execute     \ -- xt item
    list.next   \ -- xt next
  dup 0= until
  drop drop
;
```

Historische Dokumente und Tutorials

Forth ist zwar ungefähr so alt wie das Internet, aber älter als das World Wide Web (WWW). Daher liegen viele Informationen über Forth aus den 80er und Anfang der 90er Jahren in den damals gebräuchlichen Archiv–Formaten (arc, lzh, ...) auf FTP–Servern (die leider heute oft schon verschwunden sind). Auf modernen Systemen ist es oftmals schwierig, diese Dateien aus den Archiven zu entpacken und dann zu lesen.

Im Wiki der Forth–Gesellschaft–Webseite gibt es seit Mai historische Dokumente und Tutorials zu Forth aus diesen historischen Quellen.

Mit dem Übertragen dieser Dateien in das Wiki[a] der Forth–Gesellschaft werden diese Dokumente einfach zugänglich, und können über die bekannten Suchmaschinen gefunden werden.

Das Wiki ist eine Gemeinschaftsarbeit. Wenn Ihr historische Dokumente besitzt, welche frei veröffentlicht sind und auch heute noch von Interesse, tragt diese bitte in das Wiki ein oder sendet die Dokumente per E–Mail an `carsten@forth-ev.de`.

Carsten Strotmann

[a] `http://forth-ev.de/wiki/doku.php`

Bootmanager und FAT–Reparatur: Neunter Fort(h)schritt (Diskimage in RAMD–Datei)

Fred Behringer

3.5–HD–Disketten sollen 'am Stück' hexadezimal ins RAM geschrieben werden, und zwar (zur zwischenzeitlich möglichen Weiterverarbeitung) gleich in eine Datei innerhalb einer RAM–Disk. Die Adressen werden linear (32–Bit–breit) beansprucht, alle Vorgänge laufen aber im puren Real–Modus ab. Ein Abstecher in den Protected–Mode (wie noch seinerzeit in [FB98] als Allheilmittel gegen die 64–K–Segmentierung vorgesehen) ist nicht nötig.

Ziel

Der vorliegende Teil 9 meiner Artikelserie stützt sich stark auf Teil 8 (siehe [FB09]). Es ging und geht um das sektorweise Zwischenspeichern des Inhalts einer 3.5–HD–Diskette in Hexform ins 'Extended–RAM' — mit der Absicht, im RAM Änderungen vorzunehmen, um dann das modifizierte Disketten–Image auf dieselbe oder auf eine andere Diskette zurückzuschreiben. Natürlich könnte man das auch mit einem Hex–Edit–Programm von der Sorte machen, wie es sie auch für DOS schon immer gegeben hat ([CP88], [CW99], [MK92]). Leicht irritiert ist man eben nur, wenn man überraschend z.B. die Meldung 'Keine DOS–Diskette' bekommt und wenn der 'professionelle' Hex–Editor dann aussteigt. (In Teil 4 der Artikelserie (siehe [FB09]) wurde ein praktischer Anlass diskutiert.) Was tun? Man besinnt sich auf Forth und weiß: Forth kann alles. Ganz schnell und wunderbar interaktiv. Und der Lerneffekt ist groß! Wo man sich in einer puren Assembler–Umgebung nicht ans Werk wagen würde, traut man sich das in Forth ohne Weiteres zu. Natürlich in Mischung aus High–Level– und Low–Level–Forth.

Disketten 'beamen'

Das Wiederzurückschreiben auf Diskette soll im Vorliegenden abermals auf einen späteren Artikel verschoben werden. Für diesmal liegt mir die Möglichkeit am Herzen, das (nach dem Vorgehen in Teil 8 hergestellte) Disketten–Image als Datei (z.B. nach Amerika ;-) zu versenden.

Voraussetzungen

Ich gehe von einem PC–Kompatiblen mit einer CPU von mindestens dem Typ 80486 und mit DOS als Betriebssystem aus. Für meine Experimente verwende ich ein System mit einem AMD–K6–2/500 als CPU und mit 392192 KB an RAM. Meine Arbeiten habe ich mit DOS 6.2 — aber auch unter dem DOS–Prompt von Windows 95 (DOS 7.0) — durchgeführt. Von den zu verarbeitenden Disketten verlange ich, dass sie mit 80d Spuren zu je 2 Seiten mit 18d Sektoren/Spur und 512d Bytes/Sektor formatiert sind. Ansonsten wird kein Einhalten bestimmter Disketten–Parameter verlangt. Insbesondere braucht es sich durchaus nicht um DOS–Disketten zu handeln — ja, der Bootsektor (mit den Disk–Parametern) darf sogar weitestgehend leer sein. Das Forth–Programm (Turbo–Forth und meine Entwicklungen bis Teil 8) laufen von der Festplatte, die also eine DOS–Partition enthalten muss, von welcher zu booten ist. (Turbo–Forth braucht keine großen Ressourcen. Man könnte sich also auch überlegen, den Computer von Diskette zu booten.) Das BIOS muss den erweiterten Interrupt 13h bedienen können. In der `config.sys` müssen sich `himem.sys` und `ramdrive.sys` befinden. (Der RAM–Disk–Treiber kann auch anders heißen, muss aber natürlich mit dem verwendeten DOS verträglich sein. Ich habe auch Experimente mit FreeDOS und anderen DOS–Systemen gemacht. Geht prima!)

Sämtlichen Programmteilen aus Teil 8 und aus dem hier Gesagten liegt die Zahlenbasis `hex` zugrunde.

RAM–Disk?

Ich arbeite vorwiegend mit einer RAM–Disk von 30 MB. Für das Vorliegende wäre 1,44 MB ausreichend. Es muss halt eine Datei mit dem Hex–Dump einer vollen 3.5–HD–Diskette hineinpassen.

Welches Forth?

Ich gehe von Turbo–Forth in der 16–Bit–Version aus ([MP87]). Diejenigen Forth–Worte aus Teil 8, die (neben Turbo–Forth) im Vorliegenden gebraucht werden, habe ich in einem 'Glossar' (siehe gleich) zusammengestellt. Es sind wenige Dinge, die nicht auch unter ZF formuliert werden könnten. Schwierigkeiten würde bei ZF eigentlich nur das komfortable Anzeige–Wort `dump-32` aus [FB98] und (mit leichter Verbesserung) aus Teil 8 machen. (Ich schreibe alle Forth–Worte klein, was in Turbo–Forth ohne Weiteres erlaubt ist.) Der Einfachheit halber lasse ich ZF im Vorliegenden aus dem Spiel.

Grundsätzliches Vorgehen

1. Man lege in der RAM–Disk eine Datei von mindestens der Größe 1,44 MB an.
2. Am besten geeignet ist eine Text–Datei. Andere Formen tun es auch.
3. Wo finde ich unter DOS eine so große Datei? Im Repository der VD. Von dort hole ich mir regelmäßig die aktuellste Version der PDF–Datei mit dem gerade in Bearbeitung befindlichen VD–Heft für die Vorschläge meiner Artikel–Korrekturen ab. Die PDF–Datei vom 27.5.2011 hat eine Länge von 2232137 Byte. Da passt das Disketten–Image einer 3.5–HD–Diskette prima hinein.
4. Mit Hilfe der Angaben in [MA89] könnte ich mir die benötigten Parameter der RAM–Disk aus deren

Bootblock heraussuchen. Ich könnte das auch mit PC-TOOLS [CP88] machen, aber eben auch mit meinem `dump-32` aus Teil 8.

5. Könnte! Ich kann es aber auch rein überschlagsmäßig angehen. Und das habe ich getan: Die RAM–Disk beginnt hexadezimal bei `100000.` (1048576. dezimal). `100000.` ('Punktzahl'!) ist genau die Hex–Adresse des ersten Bytes des Extended–Memorys. Expanded–Memory sei im Vorliegenden ignoriert. Das in die RAM–Disk zu legende Disketten–Image hat eine Länge von 168000h (= 1474560) Bytes. Ich muss dafür sorgen, dass der Bootblock der RAM–Disk einerseits und der Vorspann der (für das Disketten–Image als Container missbrauchten) PDF–Datei andererseits für den beabsichtigten Kopiervorgang erhalten bleiben. Wenn ich also statt mit 100000h mit 150000h anfange, liege ich ganz bestimmt auf der sicheren Seite. Das habe ich nachgeprüft. Geht bestens!
6. Ich bin also mit `150000. getdiskimage` in die zweckmissbrauchte PDF–Datei auf der RAM–Disk hineingegangen und war und bin sicher, dass ich damit das (eventuell modifizierte) Disketten–Image beliebig auf Festplatte, Zip–Diskette oder USB–Stick (der Panasonic–USB–Treiber — siehe gleich — läuft bei mir unter DOS und Windows 95 bestens) abspeichern und später wieder hereinholen kann — ganz einfach dadurch, dass ich die (als Container missbrauchte) PDF–Datei hin und her kopiere.
7. In [MA89] wird darauf hingewiesen, dass man in den Bytes 1eh und 1fh des Bootblocks der RAM–Disk (dort nur 128 KB) das Ende der RAM–Disk (in KB) findet und sich durch Hochsetzen des dort gefundenen Wertes einen von der generellen Speicherverwaltung respektierten RAM–Bereich 'sichern' kann. Ich habe das hier nicht nötig: Durch Abspeichern in eine Datei, die als 'Container' für das Disketten–Image betrachtet werden kann, ist der von mir dafür beanspruchte RAM–Bereich genügend 'gesichert'. Und mit den Überlegungen aus Teil 8 brauche ich mir über solche 'Absicherungen' sowieso keine Gedanken zu machen. Ja, ich kann noch etwas weiter gehen: Dadurch, dass das Disketten–Image in einer Datei (auf der RAM–Disk) 'eingekapselt' ist, kann selbst dann nichts passieren, wenn ich außer den Turbo–Forth–Teilen noch andere Dinge, wie z.B. PCTOOLS, mit in die RAM–Disk packe. Das habe ich zum Ausprobieren denn auch getan.

USB–Stick unter DOS

Ich habe es eben unter Punkt 6 angedeutet: Ein als lokales Laufwerk (als 'Festplatte' mit FAT16) formatierter USB–Stick von 1 GB läuft bei mir unter DOS 6.2 bestens. „Geht prinzipiell nicht," hatte ich vor meinen Recherchen sagen hören. Von wegen! Ich brauchte dazu in die config.sys lediglich die beiden Zeilen

```
device=USBASPI.SYS /v
device=Di1000dd.SYS
```

einzubinden. Inzwischen habe ich schon fast wieder vergessen, von welchem FTP–Mirror es mir nach einigem Bemühen gelang, das eben angegebene Panasonic–Treiber–Paar herunterzuladen.

Mit DUSE

von Cypress hat das nichts zu tun. DUSE ist ein Kapitel für sich. Mit DUSE bin ich nicht zurechtgekommen. Da nützte auch der als PDF–Datei pompös angelegte DOS–Driver–User's–Guide von 94,1 KB nichts. Vielleicht war da meine AMD–K6–2–CPU zu primitiv? — Und warum gibt es niemanden, der versucht, den Quelltext von DUSE zu rekonstruieren? Vielleicht sogar in Forth? Das wäre doch eine interessante Aufgabe (für einen begeisterten Amateur)!

microcore.iso in der RAM–Disk

Auf dem eben erwähnten 1–GB–Stick fand ich zufällig noch (aus einem früheren Experiment) die ISO–Datei des Mikro–Linux–Live–Systems 'microcore' (hat mit dem in der Forth–Gesellschaft bekannten MicroCore nichts zu tun). Diese ISO–Datei hat eine Länge von 6,81 MB. Das ist mehr als genug für `getdiskimage` aus Teil 8. Auch das habe ich ausprobiert. 6,81 MB sind 7145472 Bytes, was 6d0800h Bytes entspricht. Mit `400000. getdiskimage` konnte ich absolut sicher sein, dass die RAM–Disk funktionsfähig bleibt und dass das in der auf der RAM–Disk deponierten ISO–Datei enthaltene Disk–Image bei beliebigem Umkopieren (der ISO–Datei) nicht angetastet wird. Meine RAM–Disk ist, wie schon erwähnt, 30 MB groß. Die ISO–Datei liegt am Anfang der RAM–Disk. Das in die ISO–Datei, ich will sie `x.iso` nennen, gelegte Disketten–Image ist mit meinem `dump-32` aus Teil 8 gut zu erkennen: Gleich am Anfang des Disketten–Images kommt die Bezeichnung FREEDOS vor (ich hatte das Image einer FreeDOS–Diskette erzeugt). Ich konnte mir also zur vertrauensbildenden Überprüfung folgendes Vorgehen erlauben (die Türme von Hanoi lassen grüßen):

1. `x.iso` in die RAM–Disk kopieren zu `y.iso`,
2. in `x.iso` mit `400000. getdiskimage` das Image der Diskette legen,
3. das derart modifizierte `x.iso` auch in die RAM–Disk kopieren zu `z.iso`,
4. `y.iso` kopieren nach `x.iso` — per `dump-32` auf image–frei überprüfen,
5. `z.iso` kopieren nach `x.iso` — per `dump-32` auf 'Image drin' überprüfen.

Nachgeprüft werden sollte, dass bei dem Hin- und Herkopieren am Disk–Image (innerhalb der ISO–Datei) nichts verlorengegangen war. Man könnte es mit einem Vergleichs–Programm genauer machen. Fürs Erste dürfte das aber genügen.

Freischalten der Adressleitung a20

Mit `himem.sys` in der `config.sys` wird die Adressleitung 20 normalerweise freigeschaltet. Hat man aus irgendeinem Grunde keine `himem.sys` in der `config.sys`, dann kann man zur Freigabe des Zugriffs auf Adressen

oberhalb `ffff:ffff` das Wort `free-a20` aus Teil 8 verwenden.

Nur Real–Modus benötigt

In [MA89] steht: „Für den Zugriff auf den Extended Memory müssen Sie den Prozessor in den 'Protected Mode' schalten."

Wirklich? Was mich betrifft, ich komme ohne eine solche Maßnahme aus! Die Adressen lassen sich (jedenfalls beim 486 und höher) auch im Real–Modus gleich 32–bit–breit ansprechen. Das Adress–Präfix 67h vor den Maschinenbefehlen macht es möglich. Alles Weitere siehe Teil 8. Hier zur schnellen Überprüfung (123456. = Punktzahl!):

```
00 123456. c!-32  77 123456. c!-32
123456. c@-32 , [ret]  77 OK
```

Glossar einiger Worte aus Teil 8

(Definitionen im Vokabular forth) ad. und len. = Punktzahlen (doppeltgenau).

```
free-a20 ( -- )          Adressleitung a20 lösen
   c@-32 ( ad. -- c )    Byte c von Adresse
                         ad. zum Stack holen
   c!-32 ( c ad. -- )    Byte c vom Stack nach
                         Adresse ad. speichern
 dump-32 ( ad. len. -- ) Stop: [SPACE].
                         Weiter: [SPACE].
                         Exit: Eingabetaste
                         Mehr: + . Nochmal zurück: -
                         Vor- oder Zurückschaltung
                         jedesmal 100h Bytes
getdiskimage ( ad. -- )  Hex-Image der 3.5-HD-Diskette
                         nach ad. ins RAM legen
```

DD–Disketten im Arbeitsspeicher

Hätte ich mich auf DD–Disketten beschränkt, so hätte ich mir nicht so viel zu überlegen brauchen: 360 KB passen 'normalerweise' gut in das RAM des (vom ursprünglichen PC so bezeichneten) 'Arbeitsspeichers'. Bei 1,44 MB (Gegenstand des vorliegenden Artikels) sieht die Sache natürlich anders aus.

Zip–Disketten

Warum aber bei HD–Disketten aufhören? Als nächstes Experiment könnten Zip–Disketten vom IOMEGA–kompatiblen Typ anstehen. Ich habe an der von mir verwendeten Experimentier–Anlage 392192 KB RAM–Speicher zur Verfügung. Da passen bis zu drei 100–MB–Zip–Disketten locker hinein. Als RAM–Disk könnte man dann beispielsweise XMSDSK von Franck Uberto [FU98] einsetzen.

Literatur

[CP88] PCTOOLS: PC–Tools–Deluxe R4.21. pctlsdlx.exe. Central Point Software, Inc.(1988).

[CW99] Walter, Christoph: cwdskedt.exe 2.22. Für DOS. Freeware.

[FB98] Behringer, Fred: Real–Mode–32–Bit–Erweiterung für Turbo-Forth. Vierte Dimension 2/1998.

[FB09] Behringer, Fred: Vierter Teil meiner VD–Artikel-Serie. Vierte Dimension 2/2009.

[FU98] Uberto, Franck: XMSDSK, eine RAM–Disk, deren beliebige Länge auch während des Betriebs noch verändert werden kann.

[MA89] Althaus, Martin: Gesprengte Ketten. DOS International 12 (1989).

[MK92] Kalisch, Martin: Disk–Editor 1.2. diskedit.exe, (1992).

[MP87] Petremann, Marc: Turbo–Forth. Liegt in verschiedenen Versionen z.B. auf dem amerikanischen FTP–Server taygeta.com.

MiniForth

MiniForth ist ein minimales, in C geschriebenes Forth–System. Neben bekannten PC–Architekturen kann MiniForth auch für 8–Bit–Architekturen wie dem ATMEL AVR kompiliert werden.

Gedacht ist MiniForth als Forth–Scripting–Erweiterung für C–Programme. Auf einem 32Bit–Linux–System mit clang[a] kompiliert MiniForth in ein 20K großes Binärprogramm.

`http://www.controlq.com/blog/wordpress/?p=352`

BareMetal OS

BareMetal OS ist ein minimalistisches Betriebssystem für die AMD/Intel x86_64 CPU Architektur. Dieses Betriebssystem ist eine Lösung wenn die standard 64bit PC Technologie benutzt werden soll, aber die ‚grossen' Betriebssysteme wie Linux oder Windows zu viel unnützen Ballast mitbringen. BareMetal OS ist ein multi–core, mono–tasking Betriebssystem, d.h. es kann pro Prozessorkern je eine Programmtask ausführen.

[a] `http://clang.llvm.org/`

BareMetal OS ist klein und übersichtlich und verfügt über ca. 70 Funktionen zur Ansteuerung der Hardware (Festplatte FAT–16, Tastatur, Bildschirm, Ethernet). An einem TCP/IP Stack wird gerade gearbeitet. Es ist vollständig in x86_64 Assembler geschrieben. BareMetal OS ist OpenSource unter einer BSD Lizenz.

`http://www.returninfinity.com/baremetal.html`

Reva Forth 2011-01

Reva, ein kleines Forth–System (Derivat von RetroForth) für Windows, Linux und MacOS X, ist in der Version 2011–01 erschienen. Die neue Version enhält Geschwindigkeitsoptimierungen, Fehler wurden bereinigt und neue Beispiele und Bibliotheken (x86–fpu, json, Datenbanken, PCRE und SQLite) wurden hinzugefügt oder aktualisiert. Reva ist unter `http://dev.ronware.org/p/reva` als ZIP–Datei zum Download erhältlich. Reva ist Public–Domain–Software.

DosBox — Laufzeitverlängerung für Forth unter DOS

Carsten Strotmann

Auf modernen Betriebssystemen wird es immer schwieriger, alte Forth Programme aus der MS–DOS–Zeit zu betreiben. Die Kompatibilität der DOS–Umgebung von Microsoft–Windows–Betriebssystemen wird von Version zu Version schlechter. Neue Rechnersysteme, wie das Google Chrome Notebook, werden mit ARM–CPUs betrieben und können daher keine DOS–Programme, welche ja für 80x86 geschrieben sind, ausführen. Ein für die Ausführung von Spiele–Software geschriebenes OpenSource–Programm namens DosBox kann helfen.

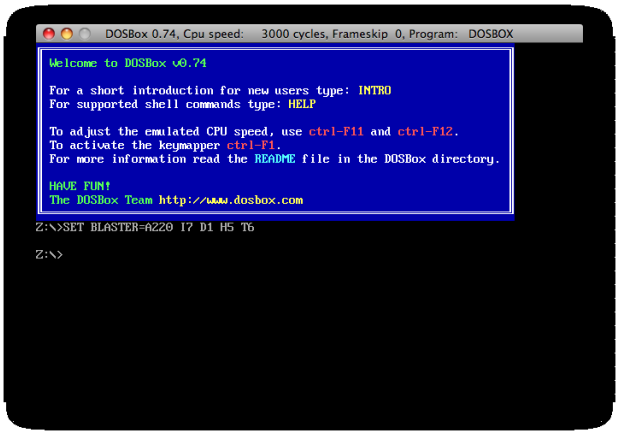
Dos in der Box

Wenn im Folgenden von "DosBox"gesprochen wird, so ist der PC-Emulator, jedoch nicht die in Windows–Betriebssystemen eingebaute DOS–Umgebung gemeint (die umgangssprachlich auch oft 'dosbox' genannt wird).

Das Programm DosBox[1] ist ein x86–Emulator mit eingebauten DOS (MS–DOS–kompatibel). Anders als Virtualisierungslösungen, wie KVM/Qemu, VirtualBox oder VMWare und Hyper–V, ist DosBox ein echter Emulator. Die x86–Instruktionen werden nicht an die CPU im Rechner unter Aufsicht eines Hypervisors weitergegeben, sondern in einer virtuellen Maschine vom Emulator abgearbeitet. Dies braucht mehr CPU–Rechenkapazität, erlaubt es aber, dass DosBox DOS–kompatible Programme auf beliebigen CPUs ausführen kann (ARM, MIPS, PowerPC, und auch x86).

Benutzung

Auf der DoxBox Webseite sind Installationspakete für die gängigen Betriebssysteme (Windows, MacOS X, Linux) und auch für weniger genutzte Systeme (Solaris Sparc, RISC-OS, FreeBSD, OS/2, BeOS ...) zu finden. Für alle anderen Systeme gibt es den Quellcode. DosBox gibt es auch für eine Reihe von mobilen Geräten wie Smartphones oder tragbaren Spielekonsolen.



Nach dem Start der Anwendung findet der Benutzer sich auf dem gewohnten DOS–Prompt wieder, jedoch nicht auf Laufwerk C: sondern auf Laufwerk Z:. Eine Besonderheit von DosBox ist, dass kein separates DOS–Betriebssystem installiert werden muss (FreeDOS, MS-DOS, PC-DOS etc), sondern dass DosBox ein DOS mitbringt. Dieses DOS befindet sich außerhalb des emulierten PCs, so dass ganze 632KB im unteren Speicher des PCs für die Anwendungsprogramme zur Verfügung stehen.

Im Laufwerk Z: des DosBox–PCs befinden sich Hilfsprogramme, mittels der die DOS–Umgebung angepasst werden kann. Diese Hilfsprogramme sind auch emuliert und nicht physisch vorhanden, die angezeigte 'AUTOEXEC.BAT' ist (wenn vorhanden) nicht als echte Datei vorhanden, sondern wird von DosBox aus Informationen in der DosBox-Konfiguration erstellt.

Serielle Schnittstellen über TCP/IP

Echte serielle Schnittstellen sind selten geworden in modernen PCs. Es gibt zwar seriell–zu–USB–Adapter, doch lassen sich diese Adapter mit alter DOS–Software (die ja auch auf die echte UART–Hardware zugreifen will) oft nicht zuverlässig nutzen.

DosBox erlaubt es dem Benutzer, die serielle Schnittstelle umzulenken. So kann über die Konfigurationseinstellung 'directserial' eine serielle Schnittstelle des Hostrechners (auf dem das Programm DosBox läuft) an eine serielle UART Schnittstelle innerhalb der DosBox weitergeleitet werden. Unter Unix/MacOS X kann die DosBox–Schnittstelle an jedes beliebige Zeichen–Ein/Ausgabe–Gerät gebunden werden. Im folgenden Beispiel (MS Windows) wird die serielle Schnittstelle 1 innerhalb der DosBox mit dem Gerät 'com1' unter Windows verbunden:

```
serial1=directserial realport:com1 startbps:1200
parity:e bytesize:8 stopbits:1 irq:4
```

Programme innerhalb der DosBox sehen eine echte Hardware–UART–serielle Schnittstelle, auch wenn auf dem Hostsystem nur eine USB–serielle Schnittstelle vorhanden ist. Die Umleitungseinstellungen werden in der Konfigurations–Datei 'dosbox.conf'[2] vorgenommen.

Sehr nützlich ist die Möglichkeit, eine serielle Schnittstelle über das Netzwerk umzuleiten. DosBox sendet in diesem Modus die Daten über eine Telnet–Verbindung über TCP zu einem entfernten Rechner. Damit lassen

[1] `http://www.dosbox.com/`

[2] siehe `http://www.dosbox.com/wiki/Dosbox.conf`

sich zwei Programme, die beide auf verschiedenen Rechnern innerhalb der DosBox laufen, über ein TCP/IP–Netzwerk verbinden. Die Programme sehen eine Nullmodemverbindung zwischen den beiden Maschinen. Damit lassen sich DOS–Forth–Programme mit Netzwerkfunktionen aufrüsten, ohne dass ein Netzwerk–Stack oder Netzwerk–Karten–Treiber unter DOS benötigt werden.

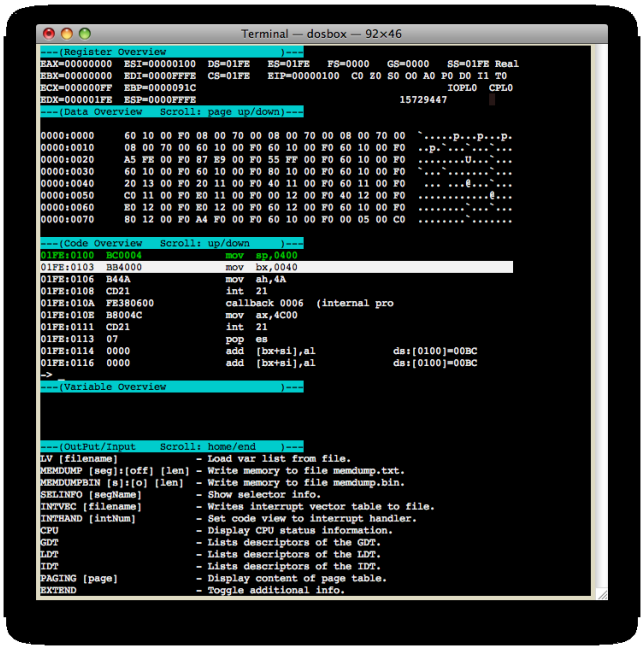
Beispiel einer Nullmodem–über–TCP–Konfiguration:
`serial1=nullmodem server:my.other.doxbox.example.com port:20000`

Neben dem Netzwerk über serielle Schnittstellen unterstützt DosBox auch das alte IPX–Netzwerkprotokoll[3].

DosBox–Debugger

Bestandteil des DosBox–Quellcodes ist auch ein sehr guter Debugger. Mit diesem Debugger kann die gesamte DOS–Umgebung innerhalb des DosBox–Programms untersucht werden. Der Debugger muss beim Kompilieren des DoxBox–Programms angeschaltet werden `./configure -enable-debug`. Da die vorgefertigten DosBox–Pakete im Internet, aber auch in Linux Distributionen, eher Spieler denn Programmierer als Zielgruppe sehen, ist in diesen Installationspaketen der Debugger leider nicht enthalten. Aber zum Glück kann der Sourcecode geladen und selbst kompiliert[4] werden.

Ist der Debugger im Programm enthalten, kann dieser mit einer Tastenkombination [5] oder mit dem Programm 'debug.com <exe-datei>' (aus Laufwerk Z:) aufgerufen werden.



Diese Ausgabe des Debuggers erscheint im Terminal/Consolen–Fenster, DosBox muss daher von einer Kommandozeile gestartet werden (und nicht über ein Programm–Icon). Das Debugger–Fenster ist in fünf Bereiche aufgeteilt:

1. „Register Overview“ zeigt die Prozessor–Register der emulierten CPU an.
2. „Data Overview“ bietet eine Hex– und ASCII–Anzeige des Hauptspeichers. Mit den Tasten 'Bild–auf'/'Bild–ab' kann ueber den Inhalt des Speichers gescrollt werden.
3. „Code Overview“ zeigt einen symbolischen Disassembler. Hier können Programme im Speicher der DosBox im Assemblercode betrachtet werden. Mittels der Cursor–auf/–ab–Tasten kann der Programmtext ausgewählt werden.
4. „Variable Overview“ ermöglicht es, bestimmte Speicherstellen immer im Blick zu halten. Mit dem Befehl 'IV segment:offset name' kann ein Speicherbereich benannt werden. Dieser wird nun im „Variable Overview“ Bereich angezeigt und laufend aktualisiert. Beispiel: 'IV 0:413 memsize'. Über den Befehl 'SV datei' kann die Variablenliste gespeichert werden, mit 'LV datei' wird eine gespeicherte Liste wieder eingeladen.
5. „Input/Output“: In diesem Bereich zeigt der Debugger Status– und Fehlermeldungen an. Auch die Kommando-Liste wird durch den Befehl 'help' hier ausgegeben. Dieser Bereich wird über die Tasten 'Pos1' und 'Ende' gescrollt.

Klappe, die Erste

DosBox bietet verschiedene Möglichkeiten, Ton– und Bildschirm–Ausgabe aus dem Emulator aufzunehmen. CTRL+F5 speichert den aktuellen Bildschirm–Inhalt als Bildschirmphoto im PNG[6] Format. CTRL+ALT+F5 startet eine Videoaufzeichnung. Bis wieder CTRL+ALT+F5 gedrückt wird, speichert DosBox den Bildschirm–Inhalt als AVI–Film auf die Festplatte. Ideal für kleine Filme zu Schulungs– und Dokumentations–Zwecken.

Töne lassen sich aus der DosBox im WAV–, DRO–[7] oder MIDI–Format aufnehmen.

DosBox ist ein hilfreiches Programm für Programmierer, die 'legacy' Forth–Programme unter MS-DOS warten. Alte PC–Hardware wird sehr zuverlässig emuliert, und man muss nicht auf den Komfort neuer Rechner verzichten.

Und nicht vergessen: Spielen kann man in der DosBox auch!

[3] `http://de.wikipedia.org/wiki/Internetwork_Packet_Exchange`

[4] Windows Benutzer finden eine DosBox Version mit Debugger unter `http://kannan.jumbledthoughts.com/index.php/howto-use-dosbox-as-a-quick-dirty-disassembler/`

[5] Tastenkombination einstellbar mit dem Tastaturmapper, CTRL+F1

[6] Portable Network Graphics `http://de.wikipedia.org/wiki/Portable_Network_Graphics`

[7] siehe `http://de.wikipedia.org/wiki/Yamaha_YM3812`

Interrupts in amForth

Matthias Trute

Bei Mikrocontrollern sind Interrupts ein wichtiger Teil der Laufzeitumgebung. Für jeden Interrupt wird eine spezialisierte Routine aufgerufen, die sich kurz und knapp mit ihm beschäftigt. Bei compilierten Sprachen ist es unkompliziert. Wenn die Interrupts jedoch im Interpreter abgewickelt werden sollen, steigt der Aufwand.

Anfänge

Die ersten Versionen von amforth hatten schlicht keine Unterstützung für Interrupts. Der erste Interrupt war derjenige für die serielle Schnittstelle und war direkt in Assembler programmiert. Weitere Interrupts waren nach diesem Muster einzubeziehen. Beispiele sind in dem Artikel von Michael und Adolf in VD2010–01 zu sehen.

Sehr bald kam der Wunsch auf, die Interrupt–Routinen nicht in Assembler, sondern in Forth zu implementieren. Dazu wurde eine Möglichkeit geschaffen, mit der der innere Interpreter (amforth ist eine ITC–Implementierung) mit dem Interruptsystem des Controllers synchronisiert wurde.

Das hat grundsätzlich funktioniert, Timerticks wurden sauber verarbeitet. Sehr bald traten die ersten Probleme zu Tage: Interrupts konnten verloren gehen und einige Interrupttypen ließen den Controller einfrieren, so dass nichts mehr ging.

Es hat einige Zeit gedauert, bis die Ursachen geklärt waren und eine Idee entwickelt wurde, wie man dem beikommen kann. Wesentliche Anregungen kamen von Wojciech und Al, denen hier gedankt sei.

Ablauf

Die Atmegas kennen zwei Betriebszustände: Normalbetrieb und Interruptbetrieb. Wenn der Controller eine Interruptbedingung erkennt, unterbricht er das laufende Programm, schaltet in den Interruptbetrieb und springt zu einer im Programmspeicher hinterlegten Adresse. Die Interrupt–Routine bearbeitet den Interrupt und sorgt zusätzlich dafür, dass sich nichts am Prozessorzustand (Flags, Register) ändert. Den Abschluss der Interruptverarbeitung bildet das Zurückschalten in den Normalbetrieb und der Rücksprung zum unterbrochenen Programm.

Eine Interruptroutine in (ITC–) Forth erfordert, dass der/ein innerer Interpreter verfügbar ist, der mit Interrupts umgehen kann.

Um die Interrupts von der Controller–Ebene in die Forth–(VM)–Ebene zu transferieren wird ein ansonsten ungenutztes Flag im Controller–Status–Register genutzt. Dieses Flag wird von keinem Assemblerbefehl beeinflusst, außer dem SET–Befehl natürlich.

Damit ist der erste Schritt klar: Sobald der Controller einen Interrupt bearbeitet, wird dieses, T genannte, Flag gesetzt und die Nummer des verursachenden Interrupts gespeichert. Dann wird erst mal mit dem normalen Programm weitergemacht. Irgendwann ist dann der Interpreter wieder an der Reihe und erkennt das T–Flag. Die Forth–VM ist zu diesem Zeitpunkt in einem wohlbekannten Zustand. Das ermöglicht es, nicht mit dem nächsten Wort aus dem Standardprogramm weiterzumachen, sondern ein ganz anderes Wort einzuschieben: Die Interrupt–Routine.

Anhand der gespeicherten Interruptnummer wird aus einer Tabelle das passende Execution Token ausgelesen und dieses aufgerufen. Nach Abschluss dieses Wortes ist automatisch dafür gesorgt, dass das unterbrochene Programm weitergeht.

Da die einzige kritische Ressource der Forth–VM die beiden Stacks sind, muss die Forth–Interruptroutine lediglich vom Stackeffekt her leer sein. Die CPU–Register werden automatisch verwaltet.

Mit diesem Ansatz laufen einfache Aufgaben, wie ein Time–Ticker, zufriedenstellend. Zu Problemen kommt es jedoch, wenn in der Zeitspanne zwischen dem ersten Interrupt und der Aktivierung der zugehörigen ISR ein zweiter Interrupt auftritt. Dann geht der erste schlicht verloren und wird durch den zweiten ersetzt. Eine Lösung hierfür wäre eine Queue, die dann im inneren Interpreter abgearbeitet wird.

Das zweite Problem ist, dass einige Interruptquellen seitens der ISR bereinigt werden müssen. In der Regel erfolgt dies durch Auslesen eines Registers oder Löschen eines Flags. Erfolgt dies nicht, wird der Controller beim Wechsel in den Normalbetrieb den gleichen Interrupt erneut auslösen. Für den Nutzer friert das System ein…

Umsetzung

Interessanterweise lassen sich beide Probleme mit einer einzigen Maßnahme lösen: Der Controller wechselt erst dann vom Interruptbetrieb in den Normalbetrieb, wenn auch die Forth–ISR erledigt ist. Die Codeänderungen sind minimal. Die Auswirkungen jedoch weit reichend.

Zum ersten Problem: Solange der Controller im Interruptbetrieb arbeitet, unterdrückt er alle weiteren Interrupts. Diese werden jedoch nachgereicht, sobald der Normalbetrieb wieder aktiviert wird. Damit gehen keine Interrupts mehr verloren. Das wird seitens des Herstellers garantiert.

Das zweite Problem kann durch die entsprechenden Instruktionen in der Forth–ISR gelöst werden. Da muss der Programmierer wissen, was er machen muss.

Die Lösung ist natürlich nicht nebenwirkungsfrei. Zum einen verbleibt der Controller verhältnismäßig lange im Interruptbetrieb. Das erhöht die Latenz für andere. Desweiteren wird ein Teil des Codes der Primitiva im Normalbetrieb und ein Teil im Interruptbetrieb ausgeführt.

Das ist für die bestehenden Worte kein Problem, es muss aber bei selbst geschriebenen zumindest geprüft werden.

Kritische Routinen wie das EEPROM/Flash–Schreiben sind gegen störende Interrupts geschützt.

Ein weiterer Effekt tritt bei lang laufenden Assemblerroutinen, wie der Division oder dem Busy–Wait Loop in `1ms`, auf. Hier fängt die Forth–ISR erst an, wenn das Wort beendet ist[1]. Glücklicherweise sind die meisten Assemblerworte sehr schnell.

Beispiel

Wie sieht so eine Interuptroutine nun aus?

```
\ Einfacher Timerinterrupt
\ benutzt den Timer0

\ Timer 0 overflow interrupt
\ Die Nummer gilt für den Atmega16
$12 constant TIMER0_OVFAddr

\ Randdaten
\ CPU-Takt / 1024, overflow interrupt
\ alle 256 timer ticks.
\  16MHz -> 64 ticks/sec
\   8MHz -> 32 ticks/sec

variable tick

\ einfach nur hochzählen
: timer-int-isr
   1 tick +!
;

\ Den Timer0 programmieren und
\ tick zurücksetzen
: timer-init
  5 TCCR0 c! \ Siehe Datasheet
  ['] timer-int-isr TIMER0_OVFAddr int!
  0 tick !
;

\ Ticker einschalten
: +timer
  1 TIMSK c!
;

\ Ticker ausschalten
: -timer
  0 TIMSK c!
;
```

Das Wort `timer-isr` ist das Wort, das im Interruptmodus ausgeführt wird. Aus diesem Grund ist es auch sehr kurz. Hier ist auch die Stelle, an der jegliche Interrupt–Gründe bereinigt werden müssen. Für den Timerinterrupt ist dies jedoch nicht notwendig.

Die anderen Worte dienen dazu, das Timermodul im Controller zu initialisieren und den Interrupt zu (de–) aktivieren. Wie wird das genutzt?

```
\ Von der VCE 2011 Benchmark Suite angepasst
\ Führt das Wort XT n-mal aus

: benchme ( xt n -- )
  dup >r  \ Anzahl sichern
  +timer-init +timer
  tick @ swap
  0 do dup execute loop
  drop \ XT entfernen
  tick @
  swap -
  cr r> . ." Iterations." u. ." timer ticks" cr
;
```

Und sonst?

Controller–Kenner werden wissen, dass es eine Art globalen Interrupt–Schalter gibt. Der ist schon für den Kommandoprompt immer eingeschaltet. Es empfiehlt sich nicht, ihn abzuschalten.

Ach ja: Die beste Möglichkeit, Interrupts zu bedienen, ist immer eine kurze Assemblerroutine. Michael und Adolfs Artikel sei zur Nachahmung empfohlen.

Nicht gesucht und doch gefunden: `ttytter`

C.S.

`ttytter` ist ein Twitter–Klient–Programm für die Kommandozeile. `ttytter` ist in Perl geschrieben und sollte auf jedem System mit Perl–Interpreter funktionieren (Windows, MacOS X, Linux, xBSD ...).

`ttytter` wird interaktiv genutzt werden, um Twitter Nachrichten zu lesen und zu senden, oder von Scripten (oder von Forth–Programmen) aus per Kommandozeilen–Modus.

Mit einem Textmodus–Twitter–Programm wie `ttytter` können die Twitter–Meldungen der Forth–Gesellschaft direkt beim Programmieren verfolgt werden. Wenn Ihr Neuigkeiten für den Twitter–Dienst habt, sendet eine E–Mail an `twitter@forth-ev.de`. `ttytter` gibt es als OpenSource (FFSL) unter `http://www.floodgap.com/software/ttytter/`.

[1] Der Loop in `1ms` wird übrigens auf das gesetzte T–Flag überwacht. Wenn das „plötzlich“ gesetzt sein sollte, wird der Loop vorzeitig beendet. Bei der Division gibt es leider kein derartiges Ausstiegsszenario.

Anzeigen von Strukturen

Filippo Sala

Dieser Artikel befasst sich mit dem Anlegen und Anzeigen von Strukturen. Wenn man mit Strukturen arbeitet, benötigt man ein Werkzeug zum Anzeigen der Strukturen. Das Wort **DUMP** *ist hierfür unzureichend. Das Aufrufen der einzelnen Feldnamen enorm zeitaufwändig. Die Redefinition der Strukturworte* **STRUCTUR**, **FIELD** *und* **ENDSTRUCTUR** *löst das Problem optimal und 100% Forth–gemäß! Als praktische Anwendung wird das Programm* **DUMPPEH.4TH** *vorgestellt. Es zeigt die Struktur des Programmkopfes und die importierten Funktionen einer EXE–Datei oder DLL an. Eine solche Datei wird PE–Datei (portable executable[1]) genannt.*

Einiges über Strukturen

Strukturen sind bekanntlich eine Erweiterung des Datentyps Array. Die Struktur–Elemente sind über Namen erreichbar und können unterschiedlich lang sein. Die Struktur–Definition benutzt ausgiebig die Methode `CREATE .... DOES>`. Wenn eine Struktur–Datei geladen wird, wird ein Prototyp der Struktur angelegt. Der CREATE–Teil kreiert neue Worte, und zwar den Struktur–Bezeichner (`BNME`) und alle Feldname (`FNAME`). Zwei Tatsachen sind besonders hervorzuheben:

- Im Body (Wortkörper) von `BNAME` ist die Länge der Struktur gespeichert.
- Im Body von `FNAME` ist der Offset des Feldnames gespeichert.

Die Ausführung von `FNAME` (`DOES>`–Teil) bringt somit die Feldname–Adresse auf den Stack.

```
FNAME ( adr dfa ) @ + ( adr+ofs )
```

Der Parameter adr ist die Anfangsadresse der strukturierten Daten, die sich im Speicher befinden. Der Parameter dfa (data field address) ist die ANSForth–Bezeichnung für den Body. Befinden sich im Speicher keine Daten, so muss die Struktur erst mal mit Hilfe des Struktur–Bezeichners angelegt werden. Mit

```
BNAME ( "sname" dfa) @ char allot
```

wird der Struktur–Name `SNAME` kreiert und Speicherplatz reserviert. Mit

```
SNAME FNAME ( adr dfa ) @ + ( adr+ofs )
```

wird schließlich die Feldname–Adresse auf den Stack gebracht.

Einiges über Windows

Ein Windows–Header besteht aus mehreren Strukturen. Sie sind z. B. in MASM (Microsoft–Assembler) in der Datei `WINDOWS.INC` zu finden. Es sind 4 Strukturen und sie heißen `DOS_HEADER`, `NTFILE_HEADER`, `OPTIONAL_HEADER` und `SECTION_HEADER`. Der `OPTIONAL_HEADER` muss, trotz seines Namens, immer vorhanden sein.

Der `SECTION_HEADER` ist am wichtigsten. Er zeigt die Beschaffenheit der einzelnen Sektionen. Die 2 Sektionen TEXT (oder CODE) und DATA müssen immer vorhanden sein. Die kleinste Blockgröße im Speicher (Granularität) beträgt 4096 Bytes und somit, obwohl der Programmkopf maximal ca. 1000 Bytes belegt, beginnt die Text–Sektion bei Offset 4096. Jede Sektion belegt 4096 Bytes oder ein Vielfaches davon. Die Programme werden an die virtuelle Adresse 400000H (preferred base address) geladen. Mit der Linker–Option `/FIXED` kann man eine andere Basis–Adresse vorschreiben. Wegen der Granularität liegt das erste Code–Byte bei 401000H.

Zu Beginn des Programmkopfes steht der (alte) `DOS-HEADER`. Er ist daher wegen zweier Tatsachen wichtig. Das erste Feld `e_magic` mit dem magic byte 'MZ' zeigt, dass es sich um einen Programmkopf handelt. Das letzte Feld `e_lfanew` liefert den Offset zum `NTFILE-HEADER`.

Programm–Beschreibung

Zuerst werden die Strukturworte definiert. Die Datei mit den Strukturen `PE.INC` wird danach geladen und somit werden die Struktur–Prototypen angelegt. Alle Struktur–Elemente bekommen einen Namen. Anschließend wird der Windows–Programmkopf in den Speicher geladen und werden Worte definiert, die die Zeiger zu den einzelnen Strukturen liefern. Der Offset der einzelnen Strukturen ist entweder bekannt oder er lässt sich berechnen.

Die Strukturworte `STRUCTUR`, `FIELD` und `ENDSTRUCTUR` werden jetzt redefiniert und die Datei `PE.INC` wird erneut geladen. Die zweite Definition bewirkt, dass die Strukturen, statt erneut angelegt, jetzt angezeigt werden.

Das Programm ist ANSForth–kompatibel und wurde mit Win32for und Bigforth getestet.

Anwendungsbeispiel für Win32For

Verzeichnis `DEMOS\WINHEADER\` anlegen mit den Dateien:

- DUMPPEH.4TH — das Programm
- PE.INC — die Strukturen
- DUMPWORD.4TH — das Anwendungswort

Datei DUMPWORD.4TH einlesen. Dabei werden folgende Worte definiert:

```
: includepath_dump ( -- str len )
      s" demos\winheader\dumppeh.4th" ;
: includepath_pe   ( -- str len )
      s" demos\winheader\pe.inc" ;

CREATE path\name 64 chars allot

: DUMPPEH ( "name" -- )
   nostack ( nur für win32for )
```

[1] `http://de.wikipedia.org/wiki/Portable_Executable`

```
    bl word count path\name place
    includepath_dump INCLUDED
;
```

Mit dem Aufruf `dumppeh "name"` wird die Programmkopf–Struktur von “name” angezeigt.
Beispiele:

```
dumppeh win32for.exe
dumppeh wincon.dll
dumppeh c:\windows\notepad.exe
dumppeh .....\bigforth.exe.
dumppeh .....\gforth.exe
dumppeh .....\zf.exe
```

Listing: pe.inc

```
\ ===============
\ PE.INC
\ Die Strukturen
\ ===============

DECIMAL

doshdr ((
\? cr .( HEADER VON ) path\name count type
structur IMAGE_DOS_HEADER
   2 field e_magic       \? .( 'MZ')
   2 field e_cblp        \? .( Laenge des letzten Sektors Modulo 512)
   2 field e_cp          \? .( Dateigroesse in 512 Byte-Seiten)
   2 field e_crlc        \? .( Laenge der Relocation-Tabelle)
   2 field e_cparhdr     \? .( Headergroesse in Paragraphen [je 16 Byte])
   2 field e_minalloc    \? .( Anzahl.min der Paragraphen)
   2 field e_maxalloc    \? .( Anzahl.max der Paragraphen)
   2 field e_ss          \? .( Stack-Segment-Offset)
   2 field e_sp          \? .( Inhalt des SP Registers beim Start)
   2 field e_csum        \? .( Checksumme in Zweierkomplement)
   2 field e_ip          \? .( Inhalt des IP Registers beim Start)
   2 field e_cs          \? .( Code-Segment-Offset ab Programmanfang)
   2 field e_lfarlc      \? .( Relocation-Tabelle-Offset)
   2 field e_ovno        \? .( Overlaynummer 0 fuer den residenten Teil)
   8 field res1          \? .( reserviert)
   2 field e_oemid       \? .( OEM-Bezeichner)
   2 field e_oeminfo     \? .( OEM-Info)
  20 field e_res2        \? .( reserviert)
   4 field e_lfanew      \? .( reserviert für Offset zum NTFILE_HEADER)
endstructur
doshdr ))

pehdr ((
\? cr .( HEADER VON ) path\name count type
structur NTFILE_HEADER
  4 field Signature              \? .( 'PE')
  2 field Machine                \? .( 014Ch=i386)
  2 field NumberOfSections       \? .( Anzahl der Sektionen)
  4 field TimeDateStamp
  4 field PointerToSymbolTable   \? .( für debugging)
  4 field NumberOfSymbols        \? .( für debugging)
  2 field SizeOfOptionalHeader   \? .( Laenge vom OPTIONAL_HEADER)
  2 field Characteristic         \? .( Flags)
endstructur

structur OPTIONAL_HEADER
  2 field Magic                  \? .( 10b)
  1 field MajorLinkerVersion
  1 field MinorLinkerVersion
  4 field SizeOfCode             \? .( Code)
  4 field SizeOfInitializedData  \? .( initialisierte Daten
  4 field SizOfUninitializedData \? .( nicht initialisierte Daten)
  4 field AddressOfEntryPoint    \? .( Start relativ zur Ladeadresse)
  4 field BaseOfCode             \? .( Codebasis relativ zur Ladeadresse)
  4 field BaseOfData             \? .( Datenbasis relativ zur Ladeadresse)
```

```
56     4 field ImageBase               \? .( Ladeadresse)
57     4 field SectionAlignment        \? .( Granularitaet im Speicher)
58     4 field FileAlignment           \? .( Granularitaet in der Datei)
59     2 field MajorOsVersion
60     2 field MinorOsVersion
61     2 field MajorImageVersion
62     2 field MinorImageVersion
63     2 field MajorSubsystemVersion
64     2 field MinorSubsystemVersion
65     4 field Win32VersionValue
66     4 field SizeOfImage             \? .( Speicherverbrauch)
67     4 field SizeOfHeaders
68     4 field CheckSum
69     2 field Subsystem               \? .( 2=Windows, 3=Console)
70     2 field DllCharacteristics
71     4 field SizeOfStackReserve
72     4 field SizeOfStackCommit
73     4 field SizeOfHeapReserve
74     4 field SizeOfHeapCommit
75     4 field LoaderFlags
76     4 field NumberOfRvaAndSizes
77     4 field ExportDirectoryRva      \? .( Offset zum EXPORT_DESCRIPTOR)
78     4 field ExportSize
79     4 field ImportDirectoryRva      \? .( Offset zum IMPORT_DESCRIPTOR)
80     4 field ImportSize
81   112 field DataDirectory
82   endstructur
83   pehdr ))
84
85   sectionhdr ((
86   structur SECTION_HEADER
87     8 field SectionName
88     4 field VirtualSize             \? .( Anzahl der Bytes im Speicher)
89     4 field VirtualAddress          \? .( Offset in Speicher)
90     4 field SizeOfRawData           \? .( Laenge in der Datei)
91     4 field PointerToRawData        \? .( Offset in der Datei)
92     4 field PointerToRelocations
93     4 field PointerToLinenumbers
94     2 field NumberOfRelocations
95     2 field NumberOfLinenumbers
96     4 field Characteristics1        \? attribute
97   endstructur
98   sectionhdr ))
99
100  importhdr ((
101  structur IMPORT_DESCRIPTOR
102     4 field PointerToThunkData     \ Adresse der Struktur THUNK_DATA
103     4 field TimeDateStamp1
104     4 field ForwarderChain
105     4 field DllName                \ Name der dll-Datei
106     4 field FirstThunk
107  endstructur
108
109  structur THUNK_DATA
110     4 field ForwarderString        \ Name der Funktion: db fnummer,0,'Text',0
111     4 field Function
112     4 field Ordinal
113     4 field AddressOfData
114  endstructur
115  importhdr ))
```

Listing: dumppeh.4th

```
1  \ =============
2  \ DUMPPEH.4TH
3  \ Das Programm
```

```
4   \ =============
5
6   DECIMAL
7
8   \ Die hier definierten Worte werden vorwiegend sofort ausgefuehrt.
9
10  warning off        \ nicht für Bigforth
11  marker dumpmarker  \ Wegen den Redefinitionen, am Ende alles 'vergessen'.
12
13  \ ------------------------------------------------------------------------
14  \ Kommentar über mehreren Zeilen für Bigforth.
15
16  : (( ( -- )
17     BEGIN
18       bl word dup @
19       [ hex ffffff decimal ] literal and
20       [ hex 292902 decimal ] literal <>
21     WHILE
22       c@ 0= IF refill drop THEN
23     REPEAT
24     drop
25  ; immediate
26
27  \ ------------------------------------------------------------------------
28  \ Definition der Strukturworte.
29
30  : structur
31     CREATE here 0 , 0    \ ( "bname" -- dfa ofs=0 )
32     DOES>  @             \ ( dfa -- nbytes )
33     CREATE allot         \ ( "sname" nbytes -- )
34  ;
35
36  : field                 \ ( "fname" ofs field.dfa -- ofs+fsize )
37     CREATE over , +
38     DOES> @ +            \ ( base dfa -- base+ofs )
39  ;
40
41  : endstructur           \ ( dfa nbytes -- )
42     swap !
43  ;
44
45  \ ------------------------------------------------------------------------
46  \ Redefinition der Strukturworte.
47
48  24 CONSTANT ntab      \ Tabulatorwert
49  0  VALUE fval         \ Aktueller Feld-Wert
50  0  VALUE ofsh         \ laufender Offset beim einlesen
51  0  VALUE defflag      \ -1 bei Redefinitionen
52
53  \ Konversion.
54  : >fstring ( string0 -- str len )
55     0 BEGIN 2dup + c@ WHILE 1+ REPEAT
56  ;
57
58  \ Sektionname ausgeben.
59  : .section_name ( string0 -- )
60     >fstring 8 min 8 over - spaces type
61  ;
62
63  \ Strukturname anzeigen.
64  : structur2 ( ccc<*> -- ) cr cr bl word count type ;
65
66  \ Feldwert in hex ausgeben, Sektion-Name ausgeben.
67  : field2 ( size -- )
68     base @ swap hex
69     bl word count dup >r cr type
```

```
70      ntab r> - spaces
71      ( size ) dup
72      CASE
73        1 OF  ofsh c@ dup to fval 8 u.r  ENDOF
74        2 OF  ofsh w@ dup to fval 8 u.r  ENDOF
75        4 OF  ofsh  @ dup to fval 8 u.r  ENDOF
76        8 OF  ofsh .section_name         ENDOF
77      DUP OF  8 spaces                   ENDOF
78      ENDCASE
79      2 spaces
80      ( size ) ofsh + to ofsh
81      base !
82  ;
83
84  : endstructur2 ( -- )  noop ;
85
86  \ Sektion-Attribute anzeigen.
87  : attribute ( -- )
88     [ hex ]
89     fval 20000000 and IF  ." ausfuehren "  THEN
90     fval 40000000 and IF  ." lesen "       THEN
91     fval 80000000 and IF  ." schreiben "   THEN
92     [ decimal ]
93  ;
94
95  : rem ( -- ) source >in ! drop ;
96
97  \ Steuerung der Strukturworte, bedingtes Kommentarwort.
98  : structur     defflag IF structur2    ELSE structur    THEN ;
99  : field        defflag IF field2       ELSE field       THEN ;
100 : endstructur  defflag IF endstructur2 ELSE endstructur THEN ;
101 : \?           defflag IF noop         ELSE rem         THEN ;
102
103 \ Zum gezieltes 'include' der Strukturen, werden diese Worte spaeter mit
104 \ 'noop' oder 'rem' belegt.
105 DEFER doshdr
106 DEFER pehdr
107 DEFER sectionhdr
108 DEFER importhdr
109
110 \ ------------------------------------------------------------------------
111 \ Struktur-Prototypen anlegen, Windows-Header laden und Zeiger setzen.
112
113 24   CONSTANT ntfile_size
114 40   CONSTANT section_size
115 1000 CONSTANT header_size    \ Programmkopf-Groesse
116 \
117 \ Basis der Strukturen
118 0 VALUE dos{}
119 0 VALUE ntfile{}
120 0 VALUE optional{}
121 0 VALUE section0{}
122 \
123 0 VALUE  peheader   \ vorhanden ( -1 )
124 0 VALUE  #section   \ Anzahl der Sektionen
125 0 VALUE  fid        \ Datei-Kennung
126
127 CREATE basepage header_size chars allot  \ Buffer
128
129 \ Fehlermeldungen.
130 : dumpend1 ( -- ) cr ." Datei nicht vorhanden" dumpmarker abort ;
131 : dumpend2 ( -- ) cr ." Kein Windows-Header vorhanden" dumpmarker abort ;
132
133 \ Kontrolle.
134 : ?openerror ( n -- )          IF  dumpend1  THEN ;
135 : ?winheader ( n -- ) $5a4d <> IF  dumpend2  THEN ;
```

```
136  : ?peheader  ( n -- ) $4550 <> IF  dumpend2  THEN ;
137
138  \ Struktur-Definition aktivieren.
139  \ alle Strukturen anlegen.
140  0 to defflag
141  ' rem is doshdr
142  ' rem is pehdr
143  ' rem is sectionhdr
144  ' rem is importhdr
145
146  \ Struktur-Prototypen anlegen.
147  includepath_pe INCLUDED
148
149  : windows_header_laden ( -- )
150     path\name count r/o OPEN-FILE ?openerror to fid
151     basepage header_size fid READ-FILE 2drop
152     basepage to dos{}
153     dos{} e_magic w@ ?winheader                  \ 'MZ' ?
154     dos{} e_lfanew @ ( ofs )                     \ NTFILE_HEADER-Offset
155     dup $100 u<
156     IF
157        ( ofs ) dos{} + to ntfile{}               \ NTFILE_HEADER-Basis
158        ntfile{} Signature @ ?peheader            \ 'PE' ?
159        -1 to peheader
160        ntfile{} NumberOfSections w@ to #section  \ Anzahl der Sektionen
161        ntfile{} ntfile_size + to optional{}      \ OPTIONAL_HEADER-Basis
162        ntfile{} SizeOfOptionalHeader w@          \ OPTIONAL_HEADER-Laenge
163        optional{} + to section0{}                \ SECTION_HEADER-Basis
164     ELSE
165        drop                                      \ DOS-Header
166     THEN
167  ;
168
169  windows_header_laden
170
171  \ ------------------------------------------------------------------------------
172  \ Strukturen mit redefinierten Worten nochmal laden. Sie werden jetzt angezeigt.
173
174  \ Index nach Adresse transformieren.
175  : section{} ( i -- adr ) section_size * section0{} + ;
176
177  \ Windows-Header anzeigen.
178  : .winheader ( -- )
179     page
180     -1 to defflag                 \ Struktur-Redefinition aktivieren
181     peheader
182     IF                            \ pe_header anzeigen
183        ntfile{} to ofsh
184        ['] rem  is pehdr
185        ['] noop is doshdr
186        ['] noop is sectionhdr
187        ['] noop is importhdr
188       includepath_pe INCLUDED
189     ELSE                          \ dos_header anzeigen
190        dos{} to ofsh
191        ['] rem  is doshdr
192        ['] noop is pehdr
193        ['] noop is sectionhdr
194        ['] noop is importhdr
195        includepath_pe INCLUDED
196        dumpmarker quit            \ Ende
197     THEN
198  ;
199
200  \ Sektionen anzeigen.
201  : .sections ( -- )
```

```
202      peheader
203      IF                              \ section_header anzeigen
204         ['] rem  is sectionhdr
205         ['] noop is pehdr
206         ['] noop is doshdr
207         ['] noop is importhdr
208         #section 0
209         DO
210            i section{} to ofsh
211            includepath_pe INCLUDED
212         LOOP
213      THEN
214   ;
215
216   .winheader
217   .sections
218
219   \ ------------------------------------------------------------------------
220   \ Sektion suchen, wo die Import-Funktionen liegen.
221
222   0 VALUE virtual_size      \ VirtualSize
223   0 VALUE virtual_address   \ VirtualAddress
224   0 VALUE raw_size          \ SizeOfRawData
225   0 VALUE raw_pointer       \ PointerToRawData
226
227   \ Die Adresse ImportDirectoryRva @ (Eintrag vom OPTIONAL_HEADER) muss zwischen
228   \ VirtualAddress und VirtualAddress+VirtualSize der Sektion liegen.
229
230   : search_importsection ( -- )
231      #section 0
232      DO
233         i section{}
234         dup SizeOfRawData @ to raw_size
235         dup PointerToRawData @ to raw_pointer
236         dup VirtualAddress @ to virtual_address
237         dup VirtualSize @ to virtual_size
238         drop
239         optional{} ImportDirectoryRva @
240         virtual_address dup virtual_size + WITHIN IF leave THEN
241      LOOP
242   ;
243
244   search_importsection  \ Virtualaddress (Offset)
245
246   \ ------------------------------------------------------------------------
247   \ Import-Funktionen anzeigen.
248   \
249   \ Die IMPORT_DESCRIPTOREN bilden ein Array von Strukturen.
250   \ Die letzte ist eine Struktur mit lauter Nullen.
251
252   CREATE importsection{} raw_size chars allot   \ Array von IMPORT_DESCRIPTOREN
253
254   0  VALUE import0{}       \ 1. IMPORT_DESCRIPTOR
255   20 CONSTANT importsize   \ Länge
256
257   \ Weitere IMPORT_DESCRIPTOREN.
258   : import{} ( i -- adr ) importsize * import0{} + ;
259
260   \ Offset nach absolute Adresse transformieren.
261   : ofs>mem ( offset -- mem_adr )
262      virtual_address - importsection{} +
263   ;
264
265   \ Importsection laden und Zeiger setzen
266   : importsektion_laden ( -- )
267      raw_pointer s>d fid REPOSITION-FILE drop
```

```
268     importsection{} raw_size fid READ-FILE 2drop
269     fid CLOSE-FILE drop
270     optional{} ImportDirectoryRva @ ofs>mem to import0{}
271   ;
272
273   \ Import-Funktinen anzeigen.
274   : .import ( -- )
275     base @ hex
276     10 0 DO
277       cr
278       i import{} DllName @ dup 0<>
279       IF
280         ( Name der dll-Datei anzeigen )
281         cr ." IMPORTIERTE FUNKTIONEN AUS " ofs>mem >fstring type
282         i import{} PointerToThunkData @
283         BEGIN
284           dup ofs>mem @ dup 0<>
285         WHILE
286           ( Nummer und Name der Funktionen anzeigen )
287           dup ofs>mem w@ cr 4 .r space
288           2 + ofs>mem >fstring type
289           4 +
290         REPEAT
291         2drop
292       ELSE
293         drop leave
294       THEN
295     LOOP
296     base !
297     cr
298   ;
299
300   importsektion_laden
301   .import
302
303   dumpmarker  \ alles 'vergessen'
```

Listing: dumpword.4th

```
1   \ ====================
2   \ DUMPWORD.4TH
3   \ Das Ausfuehrungswort
4   \ ====================
5
6   DECIMAL
7
8   CREATE path\name 64 chars allot  \ Dateiname
9
10  : includepath_pe   ( -- str len ) s" demos\winheader\pe.inc" ;
11  : includepath_dump ( -- str len ) s" demos\winheader\dumppeh.4th" ;
12
13  : DUMPPEH ( "pfad\name" -- )
14     nostack                       \ nur fuer Win32for
15     bl word count path\name place
16     includepath_dump INCLUDED
17  ;
```

Top–One–Partitur

Hannes Teich

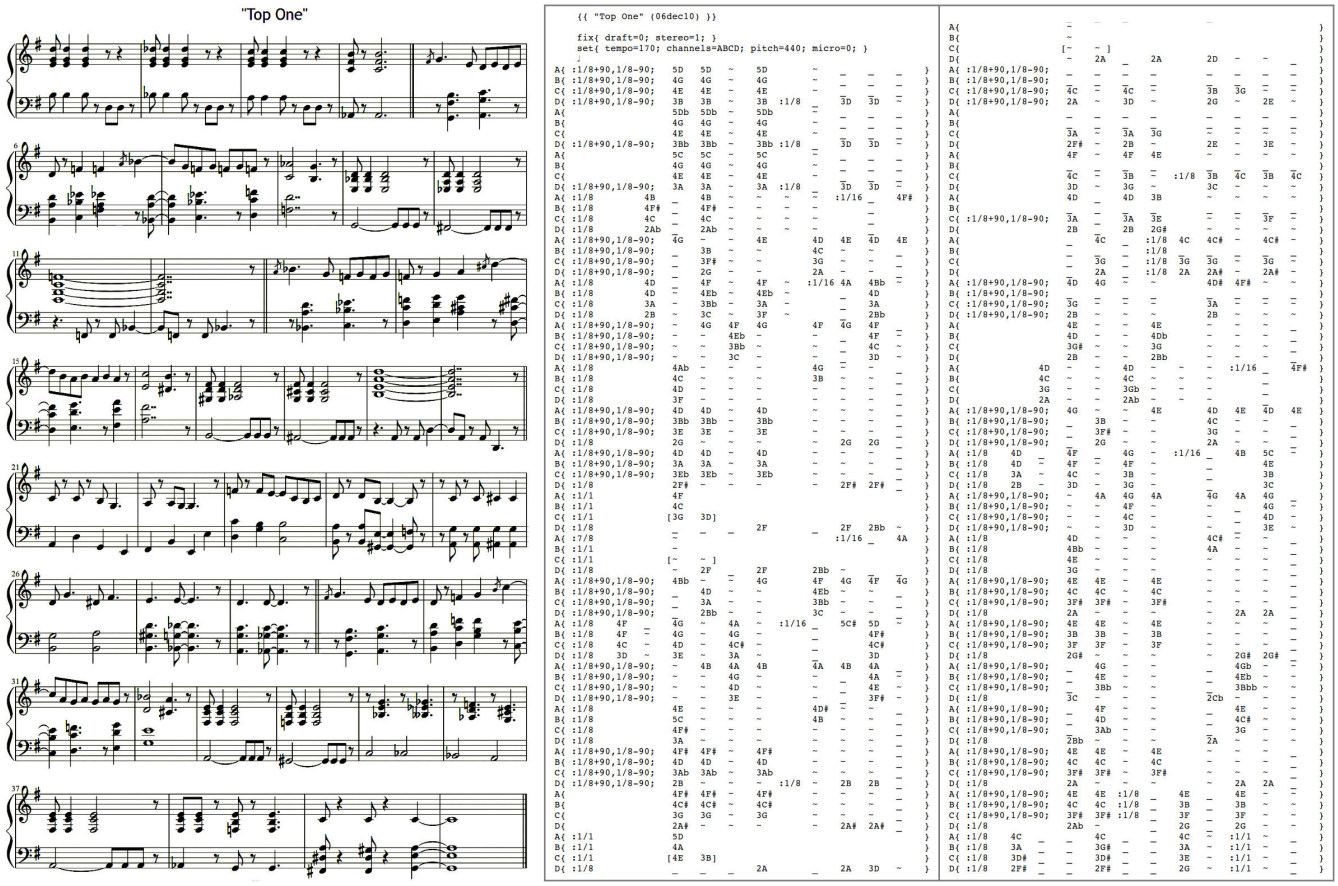
Von alleine wäre ich wohl nicht auf die Idee gekommen, das Thema hier auszubreiten, aber ich bin so freundlich ermuntert worden, dass meine Bedenken schwanden.

Ganz offensichtlich geht es hier um Musikalisches. Ein Musikstück ist auf zweierlei Art dargestellt, sowohl in herkömmlicher Notenschrift als auch in etwas kryptischer „maschinengerechter“ Form. Auf die Frage, welche Maschine sich da angesprochen fühlen könnte, sei verraten: ein PC, mit Gforth bestückt und mit einem passenden Programm geladen. Der Output ist eine abspielbare WAVE–Datei, deren Erzeugung einige Minuten in Anspruch nimmt.

An dem Programm, das solcherlei „Partituren“ zu verarbeiten imstande ist, wird seit Jahren gebastelt, und es wächst sich allmählich zu einem Monstrum aus. Ein Glück, dass Speicherbedarf kein Thema mehr ist! Das gesteckte Ziel ist denn auch nicht trivial: Das Musikstück wird nicht einfach „runtergedudelt“, sondern wird mit erheblichem Aufwand „entsterilisiert“. Das Gesamtkonzept wirft eine Menge Teilfragen auf, die nach cleveren Antworten verlangen, und das macht die Sache reizvoll.

Aus Sicht des Partitur–Schreibers handelt es sich um eine viermanualige Orgel mit großem Tonumfang, und es lassen sich viele Register ziehen, um Klangfarben und Effekte zu erzeugen. Im Konzept steckt eine Menge Gedankenarbeit, aber die Realisierung hinkt noch hinterher. Von daher riecht die Sache nach einer Fortsetzungsstory. Jedenfalls geht jetzt und hier der Raum zur Neige und der Redaktionsschluss naht. Es muss daher auf das Weihnachtsheft vertröstet werden, und dann hört man auch andere Klänge als diese Geradeaus–Version.

Wer sich fragt, welche Motive ein Mensch haben kann, ein solch „unpraktisches“ Projekt vom Zaun zu brechen, möge meinen Artikel unter `http://zirkel53.blogspot.com` streifen (und sich nicht am Pseudonym stören). Dort wird über Sinn und Unsinn der üblichen „gleichstufigen“ Musikstimmung gegrübelt. Die PC–Orgel ist jedenfalls nicht auf sie angewiesen, und mit vertrackten Rhythmen kommt sie auch gut klar.



Eine Hörprobe der Top–One–Partitur findet sich im Dateibereich der Webseite der Forth–Gesellschaft unter `http://www.forth-ev.de/filemgmt/singlefile.php?lid=365`

Forth–Gruppen regional

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

München **Bernd Paysan**
Tel.: (0 89) – 46 22 14 91 (p)
`bernd.paysan@gmx.de`
Treffen: Jeden 4. Donnerstag im Monat um 19:00, im Sommer (Mai–September) im Chilli Asia Dachauer Str. 151, im Winter im Sloveija Grill, Dachauer Str. 147, 80335 München.

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–Schöpe** Tel.: **(0 70 44)–90 87 89** (p)
Homecomputer-Forth 6502 (Commodore, Atari, Apple 2) Z80 (Amstrad, CP/M) M68K (Atari ST, Apple Mac)	**Carsten Strotmann** `cstrotm@forth-ev.de`

Termine

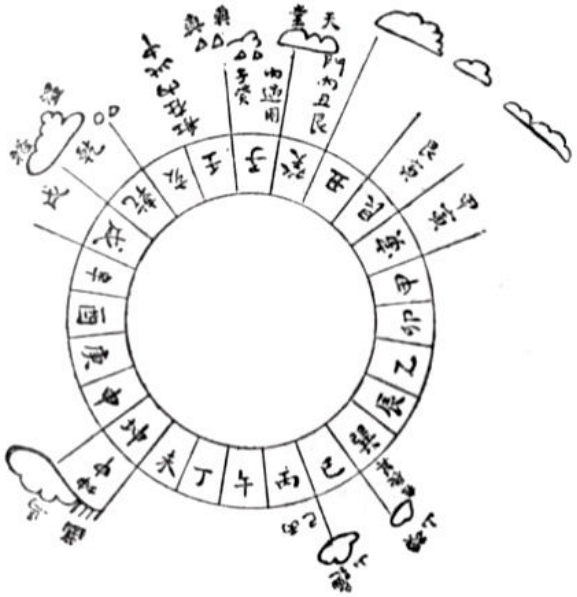
Mittwochs ab 20:00Uhr
Forth-Chat IRC #forth-ev

20-Aug–21-Aug FrOsCon
`froscon.de`

23-Sep–25-Sep EuroForth
`www.complang.tuwien.ac.at/anton/euroforth/ef11`

05-Nov Brandenburger Linux–Infotag (BLIT)
`blit.org`

12-Nov–13-Nov OpenRheinRuhr
`openrheinruhr.de`



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.

EuroForth 2011
27th EuroForth Conference
TU Wien, Vienna, Austria
September, 23rd to 25th, 2011

TECHNISCHE UNIVERSITÄT WIEN
VIENNA UNIVERSITY OF TECHNOLOGY
www.tuwien.ac.at

EuroForth is an annual conference on the Forth programming language, stack machines, and related topics, and has been held since 1985. The 27th EuroForth will be held in Vienna, Austria. The conference will be preceded by a Forth 200x standards meeting.

July 11:	Deadline for draft papers (academic stream)
August 11:	Registration deadline (later registration uncertain)
August 17:	Notification of acceptance of academic stream papers
September 14:	Deadline for camera-ready paper submission (academic and industrial stream)
September 21-23:	Forth200x meeting
September 23-25:	EuroForth 2011 conference

Information on earlier conferences can be found at the EuroForth home page. This year's EuroForth will be organized by Ewa Vesely and Anton Ertl.

Accommodation will be in the 4-star Hotel „Erzherzog Rainer“, a limited number of reservations have been made that will expire after August 11.

Links

- Registration Form and Invitation: `http://www.complang.tuwien.ac.at/anton/euroforth/ef11/registration.pdf`
- Travel Information: `http://www.complang.tuwien.ac.at/anton/euroforth/ef11/travel.html`
- Call for Papers `http://www.complang.tuwien.ac.at/anton/euroforth/ef11/cfp.html`
- EuroForth 2009 Home page `http://www.complang.tuwien.ac.at/anton/euroforth/ef11/`
- EuroForth Home `http://www.complang.tuwien.ac.at/anton/euroforth/index.html`

Hotel „Erzherzog Rainer“, Wien (Photos: `http://www.schick-hotels.com/hotels-wien-fotos-rainer.de.htm`)