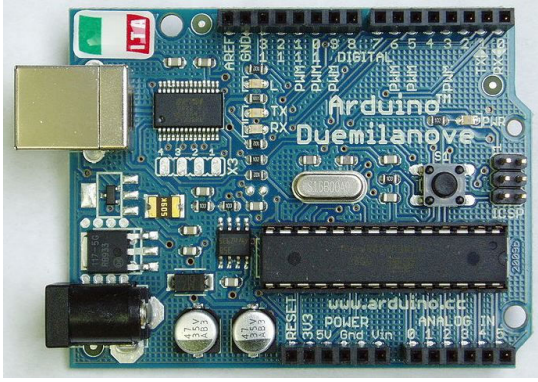
Vierte Dimension

1/2011

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.

Quelle: Florida Center for Instructional Technology (FCIT) at USF

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,

das Jahr 2011 ist ja schon ein gutes Stück zugange und unsere Konzentration ist auf die Jahrestagung gerichtet, stattfindend vom 15. bis 17. April im Zeppelin-Haus in Goslar.

„Wo bleibt denn die Vierte Dimension?“, wird sich der ein oder andere sicherlich gefragt haben. Nun — was lange währt, wird endlich gut. Ich freue mich, Euch die erste Ausgabe des 27. Jahrgangs unseres Forth–Magazins vorstellen zu dürfen.

Den Autoren dieser Ausgabe — allesamt übliche Verdächtige — sei ganz herzlich für ihre gute Mitarbeit gedankt. Unser Artikel–Vorrat ist wieder erschöpft; die Lage bleibt weiterhin unverändert kritisch: Nach wie vor brauchen wir mehr Artikel, insbesondere auch von neuen Autoren. Ich will an dieser Stelle das bereits ausgesprochene Redaktions–Wehklagen nicht erneut wiederholen, sondern verweise einfach auf das Editorial der vorherigen Ausgabe 4/2010 . . .

An Themen sollte es doch nicht mangeln. Ein Blick in Carsten Strotmanns Bericht vom Treffen der Forth Interest Group des Silicon Valleys offenbart jede Menge spannende Gebiete. Wo sind die experimentierfreudigen Forth–Programmierer, die auch noch über ihre Experimente hier in der Vierten Dimension berichten?

Fred Behringer, Michael Kalus, Willi Stricker, Carsten Strotmann und Erich Wälde haben ihre spannenden Themen gefunden. Willi schreibt über unterschiedliche Techniken, wie speicheradressierte Stacks implementiert werden können. Was für Abenteuer man mit amForth erleben kann und wie man sie dennoch erfolgreich besteht, erläutert uns Erich dann im 6. Teil seiner *Adventures in Forth*–Serie. SwiftForth von Forth, Inc. kann in einer Evaluationsversion kostenlos von ihrer Webseite geladen werden. Michael bringt uns dieses System näher. In die Tiefen der Bootsektoren und der Anfertigung von Disketten–Abbildern führt uns schließlich Fred im 8. Teil seiner Artikelserie und zeigt nebenbei, wie man 16–Bit–Forth–Systeme harmonisch um 32–Bit–Operationen ergänzt.

Anfang des Jahres erreichte uns eine Nachricht von Chris Jakeman, in der er ankündigt, dass FIG UK, das englische Pendant der Forth–Gesellschaft und mittlerweile seit 7 Jahren ohne nennenswerte Aktivitäten, Teile ihres Vereinsvermögens an unsere Forth–Freunde in den Niederlanden und an uns spenden wollen. Diese Spende nehmen wir gerne an. Da von ihrem Forth-Magazin, Forthwrite, noch nicht alle Ausgaben digitalisiert sind, soll dies nun mit den ausstehenden Ausgaben geschehen — wir haben da ja mit der Vierten Dimension unsere Erfahrungen auf diesem Gebiet.

Schaut man auf die Entwicklungen der anderen Forth–Interessengemeinschaften, so zeigt sich, dass das Einstellen der jeweiligen Forth–Publikation (Forth Dimensions, Forthwrite und vielleicht auch des Vijgeblaadjes) ein Vorbote für den Niedergang der zugehörigen Gruppen sein könnte — oder ein Symptom?

Wir sollten diskutieren, ob wir die Vierte Dimension in der bisherigen Form beibehalten wollen, oder ob nicht andere Formen — Blog, eZine, Newsletter — zeitgemäßer wären.

Schreibt mal an die Redaktion und äußert Eure Meinung dazu.

So — nun viel Freude beim Lesen dieses Hefts.

Viele Grüße, Ulli

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

iForth 4.0.6 Weihnachtsausgabe 2010

Weihnachten 2010 hat Marcel Hendrix angekündigt, dass die zweite Auflage des iForth 4.0.6xxx da ist. Sie sollte inzwischen auf `qdrive.net` bereitliegen. Die wichtigen Verbesserungen und Änderungen können im Detail in seiner Ankündigung auf comp.lang.forth nachgelesen werden [1]. iForth kostet etwas Geld, es kann durch Kagi bestellt werden [2]. Für alle iForth–Nutzer hingegen ist das update kostenlos, gegen email mit funktionierender Antwortadresse erhält man das iForth.4.0.6.tar.gz file.

Im Entwicklerteam sind Marcel Hendrix, Hanno Schwalm and Charles Turner. Es gab eine kostenlose Probierversion (evaluation version), die jedoch wegen Installationsproblemen auf OSX derzeit nicht zur Verfügung steht. Wir hoffen, dass auch das bald klappt, mit Windows und Linux gehts. mk

[1] Quelle: iForth 4.0.6 Christmas 2010 release, Marcel Hendrix, Jan 1, 8:58 pm
[2] `http://store.kagi.com/cgi-bin/store.cgi?storeID=AMP_LIVE&currency=EUR`

`Marcel Hendrix, Jan 9, 4:56 am`
`Newsgroups: comp.lang.forth`

`Die kostenlose iForth 4.0.6 Erprobungsversion für Linux und Windows ist da. Sie enthält den vollständigen 32-Bit und 64-Bit-Compiler, jedoch nicht alle Bibliotheken, aber alle Beispiele. Außerdem muss patch1.zip dazu geladen werden. http://www.qdrive.net/MarcelH/file/199774/ac83897a327e711043d2dedb4ba`

`Viel Vergnügen,`
`-marcel`

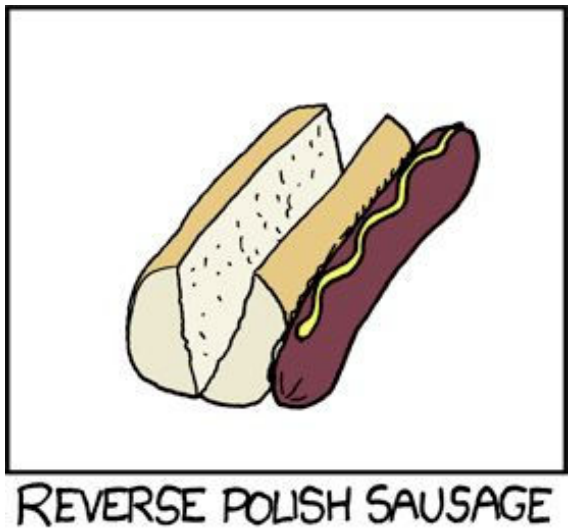
Nachricht aus Übersee

Dear Fred,

ich habe es doch noch zum SVFIG–Treffen (am 21.1.2011) geschafft, nachdem ich herausgefunden hatte, dass Carsten seine Vorführung erst am Nachmittag veranstalten würde. Ich danke dir, dass du mich informiert hast. Ich bin wirklich froh darüber, dass ich ihn getroffen habe und seinen Vortrag hören konnte. Seine Videos mit den Roboter–Vorführungen der deutschen Forth–Gesellschaft waren recht eindrucksvoll. Das VD–Heft vom Ende des letzten Jahres machte diesmal seine Runde um den ganzen Tisch und wurde, wie ich meine, von allen der 14 oder 15 anwesenden SVFIG–Mitgliedern mit Interesse angeschaut. Carsten bot mir die Übersetzung von allen Artikeln an, die unsere Gruppe interessieren könnten. Ich dankte ihm, bat ihn aber, seine freundliche Unterstützung mit Kevin und George abzusprechen, da ich nicht mehr allzu häufig zu den Treffen fahren kann. Unser Masa Kasahara, der ein paar Jahre in Deutschland gelebt hat, hörte meine Worte an Carsten und versprach, sehr zu meiner Freude, meine bisherigen Berichte an eure Gruppe (ich nehme an, auf Englisch) nach besten Kräften, soweit seine Zeit es ihm erlaubt, fortzuführen. Natürlich sind deine Übersetzungen immer willkommen und ich hoffe auch, dass Masa und Carsten miteinander in Kontakt bleiben.

Ich hatte die letzten beiden VD–Ausgaben zum Treffen mitgenommen, aber am meisten Aufmerksamkeit zog jene Ausgabe auf sich, in welcher der WikiReader beschrieben wurde. Ich hoffe, dass Carsten selbst Zeit finden wird, in der Vierten Dimension über seine Eindrücke von unserem heutigen SVFIG–Treffen zu berichten.

With heartfelt thanks and sincere regards to you,
Henry (Übers.: Fred Behringer)



Quelle: http://Xkcd.com/645

Neuigkeiten von *übern Teich*

Carsten Strotmann

Ende Januar hatte ich die Gelegenheit, an dem monatlichem Treffen der Silicon Valley Forth User Group teilzunehmen. Die SVFIG ist so etwas wie die Keimzelle der Forth Interest Group, und immer noch sehr aktiv. Die Gruppe trifft sich monatlich auf dem Campus der Stanford Universität in Palo Alto. Diesen Januar waren 12 Forthler zum Erfahrungsaustausch anwesend. Im lockerem Stil wurde über aktuelle Projekte und Entwicklungen berichtet.

Forth auf dem Arduino

Den Anfang der Vorträge machte Chen–Hanson Ting (CH Ting) über seine Erfahrungen mit Forth auf dem Arduino. Ting hat einen Arduino *Uno* bekommen, den Nachfolger des Arduino 2009 *Duemilanove*. Tings erster Versuch war, das amForth auf dem *Uno* zum Laufen zu bringen. Zwar hat der *Uno* die gleiche CPU wie der *Due* (Atmega328), aber amForth laeuft derzeit nicht 'out of the box' mit dem *Uno* (Der Bootloader und die USB–Seriell Schnittstelle ist eine andere). Leicht frustriert von diesen erfolglosen Tests, versuchte Ting, das amForth im Atmel Simulator unter Windows zu starten, um ein Gefühl für amForth zu bekommen. Obwohl alle anderen Atmel–Beispielprogramme ohne Probleme liefen, verursachte amForth jedoch einen Komplettabsturz des Atmel Simulators. Eine Untersuchung ergab, dass amForth korrekt programmiert ist, aber wohl der Simulator einen Fehler enthält der (nur) durch amForth ausgelöst wird.

Die Arduino–Boards kommen von Hause aus mit einer einfachen Programmierumgebung namens *Sketch*. Sketch ist eine C–ähnliche Sprache, welche den Einstieg in die Arduino–Programmierung für Programmier–Anfänger erleichtert, die *Sketch* IDE ist in Java geschrieben und für Anfänger relativ einfach zu bedienen. *Sketch* wird im Hintergrund der Sketch–IDE in echten C–Programmcode und dann noch auf dem PC per GNU–C–Compiler in ein Binärprogramm übersetzt und an den Arduino gesendet. Aus Mangel an direkten Alternativen implementierte Ting das eForth, geschrieben in *Sketch*, für den Arduino. Die vordefinierten Forth–Wörter werden mit der *PROGMEM*–Anweisung als Datenstrukturen direkt in den Flashspeicher geschrieben. Dies funktioniert, es wird aber unnötig viel Speicher für die *Sketch*–Routinen verbraucht. Auch ist die Erweiterung des Forth–Dictionary in diesem Sketch–Forth nur im (S)RAM des Atmel möglich, nicht im Flash. Also keine wirklich befriedigende Lösung.

Im aktuellen Projekt erstellt C. H. Ting nun ein eForth in Atmel–Assembler für den Arduino *Uno*. Dieses System soll auf dem März–Treffen der SVFIG vorgestellt werden. Der eForth–Interpreter funktioniert schon, am Compiler wird derzeit noch gearbeitet.

Nachtrag: Auf dem Februartreffen der SVFIG haben Bill und Michael Ragsdale über ihre Erfahrungen mit dem Arduino *Uno* und amForth berichtet. Auf dem gleichen Treffen berichtet Leon Wagner von Forth Inc. über das neue SwiftX für den Arduino *Uno*. Und Charley Shattuck berichtet über MyForth auf dem Arduino (tethered Forth, benutzt GForth für target compiling).

Obwohl C. H. Ting im Januar seinen Vortrag mit der Erkenntnis begann, dass der Atmel im Arduino aufgrund der Harvard–Architektur eine feindselige Umgebung für Forth darstellt, mausert sich Atmel zur Nummer *Uno* Forth Plattform. Arduino überall!

Mittagspause

Zur Mittagspause gab es Snacks im *Treehouse* der Stanford–Universität. Es war sehr angenehm, mitten im *Winter* bei 24 °C im T–Shirt in der Sonne sitzen zu können.

Nach dem Mittagessen gab es eine kleine Aufheiterung in Form des *Turbo Encabulator* Videos: `http://www.youtube.com/watch?v=rLDgQg6bq7`

Das XuLA Board

James Bowman zeigte das XuLA FPGA Board (Xilinx Spartan). Dieses Board beinhaltet auf 51 mm x 25 mm ein 50,000–gate FPGA, 8 MByte SDRAM, 2 Mbit Flash, zwei Spannungs–Regler und ein PIC 18F14K50 Mikrocontroller. Das Board ist zum Preis von 30 US$ bei der Firma XESS zu bestellen. James benutzt das Board für die Arbeit an der J1 Forth CPU, welche auf der EuroForth 2010 vorgestellt wurde.

eForth auf Green Arrays GA144

John Rible berichtete als der Kassenwart der SVFIG über die Finanzen der Gruppe und zeigte danach ein paar Bilder der GA144 Waver. Der GA144 ist der neue Chip (1cm x 1xm) mit 144 F18A Forth CPUs, entwickelt von Chuck Moore und seinem Team bei Green Arrays. Die Waver sollten eigendlich direkt von der Fab zum Chip–Gehäuse–Hersteller gehen, um dort in Gehäuse gepackt zu werden, wurden aber aus Versehen über das Green Arrays Büros in Incline Village (Lake Tahoe, Nevada) gesendet.

Die fertigen Chips sollen ab April/Mai 2011 für interessierte Kunden verfügbar sein.

Forth–Bildungs–Gruppe

Ich hatte dann meinen kleinen Auftritt mit einer Übersicht der Aktivitäten der Forth–Bildungs–Gruppe in der Forth–Gesellschaft. Gezeigt habe ich Bilder von unseren LinuxTag–Aktivitäten, speziell das Video mit dem Solitär–Roboter von Ewald Rieger und Friedel Amend wurde begeistert aufgenommen. Die SVFIG Gruppe hat ähnliche Aktivitäten in der Planung, z. B. wird die Gruppe im Mai auf der *Maker Faire Bay Area*, einer Messe

für Bastler und Erfinder der Zeitschrift *Make*, ausstellen und für Forth werben.

GForth auf eCos

John E. Harbold arbeitet an einem Port des GForth an das embedded Open Source Betriebssystem eCos. In seinem kleinen Vortrag hob John speziell die Vorzüge des GForth gegenüber anderen Forth–Systemen hervor, z. B. die Schnittstellen, um in C geschriebene Programmbibliotheken nutzen zu können.

Native Client Forth im Chrome Webbrowser

Brad Nelson ist der Autor des Rainbow Forth. Rainbow Forth ist ein Abkömmling der ColorForth–Familie und für Linux, Window und als Web–Applikation im Browser verfügbar. Die ursprüngliche Webapplikation ist mittels JavaScript im Browser und Python und Forth in der Google AppEngine gebaut. Nun ist die Ausführungsgeschwindigkeit von in JavaScript geschrieben Forth Interpretern nicht die Beste (selbst wenn JavaScript in den letzen Jahren erheblich von optimierten modernen VM Technologien profitiert hat). Google führt mit Version 10 des Chrome Webbrowsers die *Native Client* Technologie in den Browser ein.

Native Client erlaubt in sicherer Weise (per Sandbox), nativen Programmcode (x86, x86_64, ARM) im Browser ablaufen zu lassen. Es wird also nicht interpretiert wie bei JavaScript, sondern der Binärcode des Programms wird im Browser direkt auf der CPU ausgeführt. Dabei verhindert die *Native Client* Technologie, dass der Prozess aus dem Browser ausbricht und unberechtigt auf Dateien oder andere Resourcen des Rechners zugreift.

Brad arbeitet nun an einer neuen Version des Rainbow Forth, welche mit Hilfe der *Native Client* Technologie in voller Geschwindigkeit, aber im Browser ablaufen kann. Für den Benutzer ergibt sich der Vorteil, dass das Forth–System aus dem Netz geladen wird, nicht *installiert* werden muss und immer auf dem neusten Stand ist. Die erstellten Forth–Programme können wahlweise im Netz (ColorForth encoded) oder später auch auf der lokalen Platte (nach Rechtefreigabe durch den Benutzer) gespeichert werden.

Links

Arduinio

Arduino *Uno*: `http://arduino.cc/en/Main/ArduinoBoardUno`

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

Arduino PROGMEM: `http://www.arduino.cc/en/Reference/PROGMEM`

Folien amForth auf dem Arduino: `http://www.forth.org/svfig/kk/02-2011-Ragsdale.pdf`

SVFIG Februar Treffen *Arduino day*: `http://www.forth.org/svfig/kk/02-2011.html`

XuLA

XuLA FPGA: `http://www.xess.com/prods/prod047.php`

J1 FPGA CPU: `http://excamera.com/sphinx/fpga-j1.html`

Green Arrays

Green Arrays: `http://www.greenarraychips.com/`

Maker Faire

Maker Faire: `http://makerfaire.com/`

eCos

eCos: `http://ecos.sourceware.org/`

Rainbow Forth

Rainbow Forth: `http://rainbowforth.sourceforge.net/`

Rainbow Forth in JavaScript: `http://rainbowforth.appspot.com/view?start=300&end=315&`

Google Chrome Native Client: `http://www.chromium.org/nativeclient`

Native Client Forth: `https://docs.google.com/present/view?id=0ARW9X5bieOAwZGRiaGQ4MmZfMjhmNW16djgzYw&hl=en&authkey=CLuf64AI`

SVFIG Webseite: `http://www.forth.org/svfig`

Bootmanager und FAT–Reparatur: Achter Fort(h)schritt (Rawwrite/Diskcopy)

Fred Behringer

Es wird ein Programm (mit Namen `getdiskimage`) entwickelt, mit dessen Hilfe man ein Abbild einer (weitestgehend beliebigen) 3.5-HD–Diskette (nicht unbedingt DOS) ins RAM schreiben kann. Die Darstellung benutzt das übliche `dump` — hier *leicht* erweitert zum 32–bittigen `dump-32`, mit dem auf das gesamte RAM linear zugegriffen werden kann. Das Programm arbeitet auch da, wo `Rawwrite` oder/und `Diskcopy` versagen. Ein Pendant (ich werde es `putdiskimage` nennen) zum Zurückschreiben (wieder auf Diskette) ist für später vorgesehen.

Ziel (für jetzt und für eventuell weitere Artikel)

In Teil 4 dieser Artikelserie habe ich über eine Verlagerung des BIOSes (mit Veränderungsmöglichkeiten) ins RAM berichtet [FB09]. Dazu hatte ich vor dem eigentlichen Booten eine Hilfsdiskette mit präpariertem Bootsektor verwendet. Ich wollte mit Diskcopy schnell eine Kopie dieser Hilfsdiskette herstellen — und erlebte eine Überraschung: Weder Diskcopy noch `Rawwrite` kamen mit dieser Aufgabe zurecht — wenn die Disketten nicht bestimmte Merkmale von DOS– oder DOS–ähnlichen Disketten aufwiesen. Da es mir seinerzeit bei der *Hilfsdiskette* nur um den Bootsektor ging, genügte es zunächst, nur den Bootsektor zu übertragen — und das war mit den davor entwickelten Mitteln meiner Serie bereits zu erreichen. Ich möchte aber nicht aufgeben und habe nun vor, per Forth eine ganze Diskette sektorweise zu kopieren, also das, was bei DOS–Disketten üblicherweise über Diskcopy erreicht werden kann. Die Diskette soll sich aber nicht vollständig dem DOS–System unterwerfen, sondern zudem auch weitestgehende Strukturfreiheit zulassen. Gleichzeitig möchte ich die Möglichkeit bieten, in den Kopierprozess einzugreifen: Der Diskenteninhalt wird (sektorweise) erst ins RAM kopiert, dort nach Wunsch verändert und dann erst auf eine neue Diskette (darf auch dieselbe sein) zurückgeschrieben.

Die Veränderung des Disketten–Abbildes könnte man, nebenbei gesagt, zumindest bei DOS–Disketten

auch per PCTOOLS

erreichen: Quelldiskette ins Laufwerk legen, *Quelldiskette wird eingelesen* arbeiten lassen und dann nach *Zieldiskette einlegen* das Laufwerk einfach unbelegt lassen und aber trotzdem zur vorgetäuschten Bestätigung die Eingabe–Taste drücken. (Vorsichtshalber sei gesagt, dass ich dabei davon ausgehe, dass nur ein einziges Disketten–Laufwerk vorhaben ist.) PCTOOLS speichert das Disketten–Abbild in einer ganze normalen temporären Datei, die dann aber nach der Betätigung der Eingabe–Taste gelöscht wird. Beim Löschen einer DOS–Datei wird bekanntlich nichts anderes gemacht, als dass in der Datei–Zuordnungstabelle (in der FAT) das erste Zeichen im Dateinamen durch ein hexadezimales e5 ersetzt wird. Die betreffende Datei kann dann also (z. B. auch per PCTOOLS) durch Ersetzen des `e5`–Bytes sofort wieder zum Leben erweckt werden. Die bisher entwickelten Hilfsmittel der vorliegenden Artikelserie können bei dieser Reanimierungs–Aktion ebenfalls gut eingesetzt werden.

Das Versagen von Rawwrite und andere Motive

Ein wichtiges Motiv für das Nachempfinden eines Programms in Forth, eines Programms, das man bei mehr oder weniger langem Suchen vielleicht auch schon im Internet findet, ist die immer wieder zu machende Erfahrung, dass man ein Programm erst dann richtig versteht, wenn man es selbst entworfen hat. Dann fällt einem das Verändern und Anpassen an eigene Vorgaben besonders leicht — und ganz besonders besonders leicht fällt einem das in Forth. Ich hatte aber noch ein weiteres Motiv: Bei der im vorigen Abschnitt erwähnten *Hilfsdiskette* aus [FB09] wollte ich schnell mit dem Alleskönner `Rawwrite` ans Werk gehen. Aber ach! Von wegen RAW–writen! Man kann nicht davon ausgehen, dass bei beliebiger Quelldiskette alles roh (Sektor für Sektor) übertragen wird. Bei der *Hilfsdiskette* aus Teil 4 meiner Artikelserie hatte ich es mit dem RAWWRITE 0.7 (für Windows) von John Newbigin [JN00] versucht und war bass erstaunt: Unmittelbar nach dem Einlegen der *Hilfsdiskette* lieferte `rawwritewin.exe` die Meldung *Error reading disk*. Und das bei einem so einfachen Vorgang wie dem Kopieren einer Diskette, bei der es zudem sogar auf nichts weiter ankam als auf den Bootsektor! Allerdings hatte ich gerade diesen Bootsektor total verändert und insbesondere alle Parameter–Angaben zur Struktur der Diskette (weil für mich überflüssig) ausgenullt.

Nicht nur für Hilfsdiskette aus Teil 4

Das im vorliegenden Artikel entwickelte Programm schafft die Aufgabe mit der *Hilfsdiskette* aus Teil 4 meiner Artikelserie sehr wohl! Und nicht nur das! Allerdings muss sich die Diskette an meine Vorgaben (80 Spuren, 2 Seiten, je 18 Sektoren pro Spur, 512 Bytes pro Sektor — siehe gleich) halten. (Ich wollte das Programm (zunächst einmal) so einfach wie möglich gestalten.) Das heißt aber auch, dass die zu lesende Diskette FAT–formatiert sein muss. (Für die *Hilfsdiskette* aus Teil 4 wäre das nicht unbedingt nötig gewesen. Die könnte man aber auch, wie

in Teil 4 geschehen, schon per `mbr>bak` und `bak>mbr` bearbeiten.)

Voraussetzungen

Ich möchte keinen unnötigen Ballast mit mir herumschleppen und begnüge mich mit 3.5–HD–Disketten. Sie sollen aber die vom DOS–Befehl Format (und auch von anderen gängigen Dateisystemen) erzeugte Aufteilung in Sektoren, Spuren und Seiten aufweisen. (Im Zweifelsfall gilt: Ich gehe von DOS 6.2 aus.) Das heißt, ich lege 80d Spuren (80d tracks), pro Spur 2 Seiten (2 heads) und 18d Sektoren pro Spur zugrunde. Dabei sollen 512d Bytes auf einen Sektor gehen. Abgesehen von dieser Formatiervorschrift braucht aber (und das bereitet eben den *professionellen* Programmen offensichtlich *Kopfzerbrechen*) auf der Diskette keinerlei Informationen verzeichnet zu sein, die es dem Kopierprogramm erlaubt, das jeweils vorliegende und in meinem Fall fest vorgegebene Strukturformat abzulesen. Insbesondere lasse ich zu, dass im Bootsektor alle nicht benötigten Parameter ausgenullt werden — so wie ich es in meinem Programm aus Teil 4 getan hatte. (Nebenbei gesagt, gibt es bei Disketten keinen MBR und die CHS–Zählung der Sektoren beginnt bei 1 — die Zählung von Seite und Spur beginnt bei 0.)

Dinge, die einen zur Verzweiflung treiben können

In Wikipedia und überall sonstwo liest man, dass eine 3.5–HD–Diskette eine Kapazität von 1,44 Megabyte hat. Meist steht übertrieben anglophil dann auch noch an der Stelle des Kommas ein Punkt. Meine Berechnung liefert (dezimal): 80*2*18*512 = 1.474.560 Bytes an Disketten–Kapazität. Aufklärung habe ich dann (ganz versteckt) im Internet gefunden (siehe [v511]):

1,00 Kilobyte = 1.024 Bytes 1,44 Megabyte = 1.440 Kilobytes = 1.474.560 Bytes !

Klar, werden die Wissenden sagen. Aber ist das auf Anhieb wirklich so klar? Ich hatte schon gedacht, mein Programm liefert mir falsche Ergebnisse oder, was noch schlimmer gewesen wäre, `um*` funktioniert in Turbo–Forth nicht richtig.

Was geht nicht?

Bei der Hilfsdiskette aus Teil 4 kam es nur auf den Bootsektor an. Der allein schon enthielt das veränderte Bootprogramm, mit welchem ein Abbild des ROM–BIOSes ins RAM gelegt wird, um die Arbeit des ROM–BIOSes zu übernehmen. Das Programm des vorliegenden Artikels verlangt mehr: Die Struktur der gesamten Diskette (und nicht nur die des Bootsektors) muss stimmen. Und da nimmt es nicht wunder, dass eine *Hilfsdiskette* im Sinne von Teil 4 mit dem vorliegenden Programm nicht funktioniert, wenn der abgeänderte Bootsektor auf eine 3.5–Diskette der Bauart DD geschrieben wurde. — Das Einlegen einer (beliebigen) 3.5–DD–Diskette ist übrigens eine schöne Methode, die Meldung auf dem Bildschirm für den Fall zu überprüfen, dass zwar eine Diskette im Laufwerk liegt, dass diese Diskette aber mit einem *Fehler* behaftet ist. Im Übrigen ist das Funktionieren

nicht vom Dateisystem abhängig.

Nur die *richtige* Sektor–Spur–Seiten–Einteilung zählt. Ausprobiert habe ich das Funktionieren mit Not– oder Boot–Disketten von: OS/2–Warp, FreeDOS, GRUB–0.94, Windows–95, Windows–98, Windows–ME, Tom Oehsers Linux auf Diskette [TO02] und anderen mehr.

Zur Überprüfung einer *Hilfsdiskette* des Formats DD

kann man wie folgt vorgehen: Das Forth–Wort `diskbootcopy` aus Teil 6 meiner Artikelserie [FB10] aufrufen und die Anweisungen auf dem Bildschirm befolgen.

Hauptlast im vorliegenden Artikel

trägt das Zubringer–Wort `(getdiskimage)`, die Klammern gehören zum Forth–Wort, das vollständig in Forth–Assembler geschrieben ist. Das High–Level–Wort getdiskimage ruft `(getdiskimage)` auf und nutzt die im Forth–System schon vorhandene Organisationsgewalt aus. Ich darf meine weiteren Erklärungen in Form von Beispielen (mit konkreten Eingabewerten) aufziehen.

200000. `getdiskimage`

legt ein Disketten–Abbild nach Adresse 200000. — der Punkt gehört zum Forth–Wort. Das zugrunde gelegte Turbo–Forth ist ein 16–Bit–System. Eine 3.5–HD–Diskette fasst 1.44 Megabyte. Mit (einfachgenauen) 16–Bit–Zahlen für die Adressen komme ich also (auch schon wegen des begrenzten PC–RAM–Bereichs) nicht durch. Punktzahlen, wie die eben genannte, sind doppeltgenau (32 Bit breit). Über mögliche RAM–Organisationen (wie beim Aufbau von Dateisystemen) habe ich hier nicht weiter nachgedacht. Ich bewege mich mit Turbo–Forth in einem DOS–System und agiere nach dem Motto: Mein RAM gehört mir und ich kann damit treiben, was ich will. (Übrigens befolgt ja DOS ein solches Motto mit `.com`-Dateien, die bei Aufruf (zunächst einmal) den gesamten RAM–Speicher zur Verfügung gestellt bekommen) auch. Wo nötig, habe ich bei meinen Experimenten während der Entwicklung neben dem *runden* Wert `200000.` auch `400000.`, `600000.`, `800000.`, `a00000.` usw. verwendet. (Der Eingabe–Parameter ist eine Punktzahl, doppeltgenau, also 32 Bit breit. Den Punkt nicht vergessen!)

Nachträgliche Veränderungen am Disk–Image

lassen sich mit den über `dump-32` gewonnenen Adress–Informationen per `!-32`, `@-32` etc. gezielt vornehmen. Über ein Zurückschreiben auf Diskette werde ich später berichten. `5 200003. c!-32` legt das Byte `5` an die Adresse `200003.`, wobei alle anderen Bytes in der Umgebung dieser Adresse unberührt bleiben. `4711 300100. cc!-32` legt das Byte `11` nach `300100.` und das Byte `47` nach `300101` (little–endian). `12345678. 400200. !-32` legt `78` nach `400200.`, `56` nach `400201.`, `34` nach `400202.` und `12` nach `400203` (wiederum streng little–endian).

Etwas anderes hätte in einer gemischten 16/32–Bit–Umgebung kaum einen Sinn. Man achte aber darauf, dass bei der üblichen Eingabe von 32–Bit–Zahlen in Turbo–Forth in der 16–Bit–Version 32–Bit–Zahlen als Punktzahlen geschrieben werden, welche auf dem Stack und in den 2Variablen Lo– und Hi–Anteil, im Gegensatz zur Little–Endian–Platzierung, vertauschen.

Achtung: Will man die eben gemachten Angaben per `dump-32` überprüfen, dann achte man darauf, vorher (oder zumindest zwischenzeitlich) `fs` = 0 zu setzen (zu erreichen über `0 fs!`), da `dump-32` mit einem nullgesetzten Segment `fs` arbeitet.

Alle Angaben im Listing sind hexadezimal

zu interpretieren. Selbstverständlich gilt das besonders für `dump-32`. Ich habe Möglichkeiten vorgesehen (siehe Listing auf Seite 13), im ASCII–Teil des Ausdrucks von `dump-32` solche Zeichen, die nicht dargestellt werden können, in verschiedener Weise auf den Bildschirm zu bringen (als Punkt oder anderswie).

Das Programm ist autark

Außer dem Listing des vorliegenden Artikels und einem 16–Bit–Turbo–Forth als System wird nichts weiter benötigt. Bisherige Artikel dieser Serie helfen dem Leser, können aber zunächst auch ebenso gut unberücksichtigt bleiben. Turbo–Forth findet man beispielsweise auf dem amerikanischen FTP–Server `taygeta.com` (mit Mirror in Bremen; siehe auch weiter unten).

Einen 32–Bit–Assembler

braucht man zum Überprüfen des im Vorliegenden Gesagten nicht. Der 16–Bit–Assembler aus dem 16–Bit–Turbo–Forth reicht. Ein paar zusätzliche Forth–Worte zur Erweiterung auf 32 Bit machen es möglich. Mit der Makro–Definition `: ebx opsize: bx ;` und dem 32–Bit–Präfix `: opsize: 66 c, ;` erreiche ich in Code–Definitionen beispielsweise einen leicht einprägsamen Zugriff auf das 32–Bit–Register gleichen Namens. Ähnliches gilt für 32–Bit–Adressen, mit dem Präfix `: adrsize: 67 c, ;` . Bei (linearem) Zugriff auf 32–Bit–Adressen beziehe ich mich auf `: fs: 64 c, ;` als Segment. (Sollte jemals eine ZF–Version meines Programms zustande kommen, so achte man auf die 64–Falle in ZF, auf welche uns Martin Bitter seinerzeit aufmerksam gemacht hat, und schreibe `064` statt `64`. Bei der Darstellung per `dump-32` wird von mir grundsätzlich `fs` = 0 gesetzt. Das erreicht man per `0 fs!` (im Real–Mode sind auch die Segmente `fs` und `gs` nur 16 Bit breit einstellbar).

Ich verwende eine 16/32-Bit–Mischung

An Stellen mit 16–Bit–Zugängen halte ich mich an die übliche PC–Segmentierung (Adressangaben als xxxx:yyyy). Das Segment xxxx kann in Turbo–Forth über das Wort `dsegment` bezogen werden. Die unselige Methode der IBM/Intel–Segmentierung musste ich beibehalten, da sich ja das Turbo–Forth–System auf das Segment xxxx bezieht und jedes Wackeln daran zu unnützem Zusatzaufwand führen würde. Das kann aber auch leicht zu Verwirrung führen: Will ich mir den Inhalt des Zwischenspeichers `trackbuf` ansehen, so kann ich beispielsweise `trackbuf 100 dump` schreiben. (`dump` ist das `dump` aus Turbo–Forth.) `trackbuf` liegt ja im Anwender–Teil des Forth–Systems, also unterhalb der 1–MB–Grenze. Will ich aber mein komfortableres `dump-32` (das auf die ganze Maschine zugreifen und über die Plus– oder Minus–Taste ganze RAM–Bereiche ganz schnell durchkämmen kann) ausnutzen, dann muss ich wie folgt vorgehen.

Trackbuf über das komfortable `dump-32` ansehen

Ich muss dazu beachten, dass das Turbo–Forth–System in einem bestimmten DOS–Segment (von 64 KB) liegt und dass der Zwischenspeicher `trackbuf` (etwa 9 KB) im Turbo–Forth–System (etwa 26 KB) eingebettet ist. Es müssen also die DOS–Adressen xxxx:yyyy in lineare 32–Bit–Adressen (als reine Offset–Werte) mit vernünftigerweise *Segment-Wert* 0. umgewandelt werden. Konkret:

```
trackbuf 100 dump
```

läuft bei Betrachtung per `dump-32` auf folgende Sequenz hinaus:

```
trackbuf s>d dsegment 10 um* d+ 100. dump-32
```

Es ist immer wieder interessant zu beobachten, dass auch Forth zwar *kinderleicht* in der Anwendung ist, dass man aber auch da erst richtig loslegen kann, wenn man weiß, was man tut. `s>d` arbeitet *vorzeichenrichtig*. Das heißt aber, dass man erwartungsgemäß

```
2000 s>d d. → 2000
```

angezeigt bekommt (alles in Hexwerten gesehen), dass aber

```
9000 s>d d. → -7000
```

liefert. Es nützt auch nichts, wenn man meint, mit `ud.` besser zu liegen:

```
9000 s>d ud. → ffff9000
```

Abhilfe schafft ein Sichbesinnen auf die (durchaus sinnvolle) Konstruktion von `s>d`, die man sich in Turbo–Forth per `help s>d` (oder natürlich auch mit einem schnellen Blick in den Quelltext `kernel.txt` von 82 KB oder aber über `see s>d`) beschaffen kann:

```
9000 0 d. → 9000
```

`s>d` liefert eben nicht die vorzeichenUNabhängige Erweiterung einer einfach genauen zu einer doppeltgenauen Zahl!

Zugriffe auf beliebige DOS–Segmente per dump–32

An sich hat man keine Veranlassung, irgendetwas aus dem Turbo–Forth–System mit `dump-32` zu betrachten. dump aus Turbo–Forth tut es ja auch. `dump-32` mit seiner 100h–Byte–Fortschaltung ist aber komfortabler — und man könnte das ausnutzen wollen. (Aufgabe: Man konstruiere ein Forth–Wort, das den Zugriff auf ein vorgegebenes DOS–Segment so automatisiert, dass

man sich keine komplizierte Adress–Arithmetik zu merken braucht!)

Wie schon in [FB98], so auch hier betrachte ich die Forth–Worte

@-32 und !-32 als echte 32–Bit–Erweiterungen

so, als ob sie einem 32–Bit–Forth–System entstammen. Mir erschien das schon in der Arbeit [FB98] als vernünftig. Ich halte mich aber Gegenargumenten gegenüber offen. Einen Überblick verschafft man sich wohl am ehesten, wenn man sich eine RAM–Stelle im Forth–Segment vorgibt (`pad` ist gut dazu geeignet) und `@-32` und `!-32` einerseits mit andererseits den *üblichen* (Turbo–)Forth–Worten `2@` und `2!` (ich rede hier ja vorwiegend über 32–Bit–Zahlen) vergleicht. Es gilt (alles hexadezimal):

```
12345678. swap pad 2!
pad 2@ swap  d. → 12345678
pad  40 dump → 78 56 34 12
    (im RAM strikt little-endian!)

12345678. pad 2!
pad 2@       d. → 12345678
pad  40 dump → 34 12 78 56
    (little-endian mit Hi-Lo-Vertauschung)

0 fs!
    (Bei dump-32 nicht vergessen!)

12345678. swap pad 2!
pad 0 dsegment 10 um* d+ @-32     d. → 12345678
pad 0 dsegment 10 um* d+ 40. dump-32 → 78 56 34 12

12345678. pad 0 dsegment 10 um* d+ !-32
pad 2@ swap  d. → 12345678
pad 0 dsegment 10 um* d+ @-32  d. → 12345678
pad  40 dump → 78 56 34 12
```

Damit sind, meine ich, alle Überkreuzbeziehungen erfasst. Ich betrachte das Einbringen einer Punktzahl (32 Bit) ins RAM in Form eines strikten Little–Endian–Wertes als ein natürliches Vorgehen — zumindest bei meinen Vorhaben aus [FB98] und im Umfeld des vorliegenden Artikels.

`pad` lag bei mir an Adresse 2680:93c7 und lieferte

```
pad s>d ud. → ffff93c7 .
```

Erwartet hatte ich 000093c7, also

```
pad s>d ud. →     93c7  als 32-Bit-Zahl
```

(Vergleiche das zwei Abschnitte weiter vorn Gesagte.) Solche *Kleinigkeiten* der eigenen Unzulänglichkeit kosten enorm viel Zeit!

dump-32 ist eine leicht aufpolierte Version

des Forth–Wortes gleichen Namens aus meinem VD–Artikel *Real–Mode–32–Bit–Erweiterung für Turbo–Forth* aus dem Jahre 1998 [FB98] Auch die anderen turbo–forth–erweiternden Worte, wie `@-32` und `!-32`, stammen aus derselben Quelle. Die Plus–Taste schaltet den am Bildschirm angezeigten Bereich in Schritten zu je 100h Bytes weiter. Das macht sich sehr bezahlt: Lässt man den Finger auf der Plus–Taste, dann kann man verhältnismäßig schnell zig Kilobytes durchkämmen. Ist die gesuchte Information dabei zu schnell am Auge vorbeigerauscht, dann leistet ein Innehalten und anschließendes Drücken der Minus–Taste den entsprechenden Korrekturdienst in Rückwärts–Richtung (je 100h Adress–Bytes weniger per Druck auf die Minus–Taste). Raus geht es aus `dump-32`, wenn man die Return–Taste drückt.

Die Last des Protected–Modes

wollte ich mir nicht aufbürden. Ich wollte unbedingt im Real–Modus bleiben. Mit den Hilfsmitteln aus [FB98] wäre das ja ohne Weiteres möglich. Sehr zu meinem Erstaunen habe ich dann aber bemerkt, dass das Aufbohren des Systems auf einen 4–Gigabyte–Zugang gar nicht nötig ist. Adressen lassen sich über die erweiterten Register `exx` selbstverständlich schon im üblichen Real–Mode (sprich Sofort–Mode) gleich 32–Bit–weit ansprechen — und vom eigentlichen Turbo–Forth–Programm liegt nichts oberhalb 1 MB im RAM. (Ich werde mir das für eine Inangriffname der Entwicklung einer Forth–RAM–Disk für später vormerken!)

Für die Einbeziehung einer RAM–Disk

gäbe es einen triftigen Grund: Man könnte dann in begrifflich direkter Weise das Disketten–Abbild in eine Datei legen und die Datei beispielsweise als E-Mail–Anhang verschicken (die Diskette *nach Amerika beamen* — Teleportation durch Übertragung der Träger–Information). Ob dabei die RAM–Disk eine Forth–Sonderanfertigung ist oder beispielsweise als `XMSDSK` [FU98] vorliegt, tut nichts zur Sache (Aufgabe für später oder für den interessierten Leser).

Tritt im vorliegenden Programm ein Lesefehler auf,

dann wird der Leseversuch für die anstehende Disketten–Spur wiederholt. Nach drei hintereinander aufgetretenen Lesefehlern wird das Programm abgebrochen. Am Bildschirm wird beim Aussprung dann die erreichte Seite, die erreichte Spur, die Adresse des zuletzt eingelesenen Bytes und die Angabe, ob ein Fehler vorliegt oder nicht, angezeigt.

Turbo–Forth, und kein ZF?

Turbo–Forth liegt in verschiedenen Versionen auf dem amerikanischen FTP–Server `taygeta.com`. Auch ZF ist dort zu finden. Beide Systeme fußen auf F83 von Laxen and Perry [LP84] aus dem Jahre 1984 ff. und haben das Ziel, neben der Block–Orientierung (auch) eine Anbindung an die DOS–Datei–Organisation zur Verfügung zu stellen (1987 ff.) Offensichtlich gab es bei den Entwicklungen keine Absprachen, was dazu führte, dass viele wichtige Elemente für Forth als System unterschiedlich definiert wurden. Manche lassen sich ganz schnell 1:1 anpassen. Beispiel: `l@` in Turbo–Forth und `@l` in ZF. Andere würden bei der Anpassung einen tieferen Eingriff in das System erfordern. Beispiel: `include` in Turbo–Forth und `fload` in ZF. Ich habe im vorliegenden Artikel nach

längerem Zögern ZF beiseitegeschoben und mich nur auf Turbo–Forth konzentriert. Das Thema Anpassung an ZF könnte man später gesondert behandeln.

Die Reihenfolge der Sektoren: Eine Hiobsbotschaft

Die einzige unverrückbare Einheit bei der Abspeicherung der Bytes auf die Diskette scheint die des *Sektors* zu sein: Der Sektor wird vom Controller als ein bestimmtes Maß von hintereinanderliegenden Bytes auf ein und derselben Spur betrachtet (bei 3.5–HD–Disketten üblich: 512d Bytes). Für die Durchnummerierung der Sektoren ist es eigentlich egal, auf welche Weise sie geschieht. Sie muss halt eindeutig sein (eine lineare Ordnung ergeben, d. h., es darf kein Sektor ausgelassen und keiner doppelt nummeriert werden).

Als *natürliche Ordnung* hatte ich in meinem Programm ursprünglich die Einteilung [Seite:Spur:Sektor] betrachtet. Zur Überprüfung habe ich vorwiegend die drei Disketten–Editoren diskedit.exe von Martin Kalisch [MK92], `cwdskedt.exe` von Christoph Walter [CW99] und `pctlsdlx.exe` von Central Point [CP88] verwandt. Dabei fiel mir ganz zum Schluss, als es für Änderungen im Programm fast schon zu spät war (der Artikel sollte noch ins anstehende VD–Heft), auf, dass `diskedit.exe` die Nummerierung der Sektoren nach [Spur:Seite:Sektor] vornimmt. Daher (oder umgekehrt) wohl auch die Bezeichnung CHS (und nicht HCS) — und das scheint auch beim Begriff LBA (Logical Block Addressing) mitzuspielen.

Sehr zu meiner Freude konnte ich feststellen, dass die nötige Umstellung im Programm keinen großen Aufwand erforderte. Aber wohl auch nur deshalb, weil ich inzwischen praktisch mit meinem Programm verschmolzen war (*Welt am Draht*). Ich habe bei dieser Beschäftigung wieder enorm viel gelernt! — Learning by doing.

Zur Überprüfung der Nummerierung

habe ich eine DOS–Diskette genommen und sie mit `diskedit.exe` analysiert: Die erste Datei–Zuordnungs–Tabelle (the File Allocation Table FAT) liegt ab Sektor 19d. Das Disketten–Abbild im RAM meines Experimentes hatte ich nach 200000h gelegt. 38d–mal auf die Plus–Taste gedrückt (100h Bytes = ein halber Sektor), lieferte das erhoffte Ergebnis: Voilá, das Datei–Verzeichnis aus der FAT! Im Datei–Abbbild, im RAM!

Fehler–Überprüfung

Und schließlich habe ich noch Nutzen aus meiner *konservierenden* Haltung gezogen, alles im Leben aufzuheben, wie unnütz es auch geworden sein mag: Ich habe das Programm aus dem vorliegenden Artikel auch noch auf eine DOS–Diskette angesetzt, von welcher ich wusste, dass sie Stellen aufweist, die nicht gelesen werden können (feststellbar bespielsweise (auch) über PCTOOLS).

Am Bildschirm erschien die Meldung:

```
================================================
Fehler beim Einlesen des Disketten-Abbildes
------------------------------------------------
Erstes  eingelesenes Byte an RAM-Adresse : 200000
Letztes eingelesenes Byte an RAM-Adresse : 3453FF
Letzte Seite; oder erste, die Fehler hat : 1
Letzte Spur;  oder erste, die Fehler hat : 48
------------------------------------------------
Falls Sp/S = 0/0 : Wirklich Diskette im Laufwerk?
================================================
```

Das heißt aber unter anderem auch, dass ich diese Diskette ohne Weiteres bis zu der Stelle, die der Image–Adresse 3453ffh in Spur 48h, Seite 1 entspricht, *ganz normal* verwenden kann, z.B. auch für die Anfertigung einer *Hilfsdiskette im Sinne von [FB09] zur Verpflanzung des BIOS* ins RAM.

Und um noch zu zeigen, wie ich im vorliegenden Programm den nicht ganz abwegigen Fall aufgefangen habe, dass vergessen wurde, eine Diskette ins Laufwerk zu legen, schnell noch den dann erhaltenen Bildschirm–Ausdruck:

```
================================================
Fehler beim Einlesen des Disketten-Abbildes
------------------------------------------------
Erstes  eingelesenes Byte an RAM-Adresse : 200000
Letztes eingelesenes Byte an RAM-Adresse : 1FFFFF
Letzte Seite; oder erste, die Fehler hat : 0
Letzte Spur;  oder erste, die Fehler hat : 0
------------------------------------------------
Falls Sp/S = 0/0 : Wirklich Diskette im Laufwerk?
================================================
```

Interpretation von Fehlermeldungen

Ich will es am Beispiel erklären: Bei einer meiner Disketten, die Fehler enthielt, bekam ich die Meldung, dass das letzte eingelesene Byte an der Adresse 9463ffh liegt und dass die letzte erreichte Spur 48h und die letzte erreichte Seite 1 ist. Pro Spurdurchgang verarbeitet das Programm 2400h Bytes. Will man also (zu Überprüfungszwecken) an den Anfang desjenigen Spurdurchgangs gelangen, an welchem das Programm mit der Fehlermeldung ausgestiegen ist, dann muss man 9463ffh + 1 - 2400h = 944000h bilden. Die Anfangs–Adresse des Abbildes der Diskette hatte ich zu 800000h gewählt. Die Differenz beträgt 144000h. Das entspricht aber genau der Formel 48*2*12*200h, wenn man statt der möglichen 50h Spuren die gemeldeten 48h Spuren bis zur fehlerhaften Stelle berücksichtig.

Literatur

[CP98] 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.

[FB08] Behringer, Fred: Erster Teil meiner VD–Artikel–Serie. Vierte Dimension 3/2008.

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

[FB10] Behringer, Fred: Sechster Teil meiner VD–Artikel–Serie. Vierte Dimension 1/2010.

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

[JN00] Newbigin, John: http://www.chrysocome.net/rawwrite ... (2000).

[LP84] Laxen, Henry, and Mike Perry: Public–Domain–Implementation von F83. Auf dem amerikanischen FTP–Server taygeta.com

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

[TO02] Oehser, Tom: http://www.toms.net/rb/ (Version 1.7.361). Heute, im Jahre 2011, liegt die Version 2.0.103 im Internet. Das erhältliche Paket besteht aus Dateien für DOS, Linux und CD aus den Jahren 2001 und 2002.

[v511] http://www.dm17.com/ /Programmcodes/v5PROT.

Listing

```
1   \ ******************************************************
2   \ *                                                    *
3   \ * BOOTMA-8.FTH                                       *
4   \ *                                                    *
5   \ * Zutaten fuer FAT-Reparatur und Bootmanager unter   *
6   \ * Turbo-FORTH-83                                     *
7   \ *                                                    *
8   \ * Fred Behringer - Forth-Gesellschaft - 24.3.2011    *
9   \ *                                                    *
10  \ ******************************************************
11
12  \ ======================================================
13  \ Ich habe im vorliegenden Artikel  darauf verzichtet,
14  \ die Programme auch  fuer ein Arbeiten mit ZF  einzu-
15  \ richten. Sie sind hier nur fuer Turbo-Forth gedacht!
16  \ ======================================================
17
18  hex
19
20  \ 32-Bit-Assembler-Erweiterung
21  \ ----------------------------
22  \ Im weiter unten folgenden Programm zur Anfertigung eines Disketten-Images
23  \ wird neben Punktzahlen (32-Bit-Zahlen) viel mit 32-Bit-Maschinenbefehlen
24  \ gearbeitet. Ein kompletter 32-Bit-Forth-Assembler wird dafuer aber nicht
25  \ benoetigt. Ein paar Anpassungen des in Turbo-Forth ohnehin zur Verfuegung
26  \ stehenden 16-Bit-Assemblers genuegen. Es wird von einer (PC-kompatiblen) CPU
27  \ von mindestens dem Typ 80486 ausgegangen. Im Zweifelsfall gilt: Meine
28  \ Experimente wurden mit einem AMD-K6-2/500 durchgefuehrt.
29
30  only forth also assembler definitions
31
32  : fs:      64 c, ; : gs:      65 c, ; : opsize:  66 c, ; : adrsize: 67 c, ;
33  : eax opsize: ax ; : ecx opsize: cx ; : edx opsize: dx ; : ebx opsize: bx ;
34  : esp opsize: sp ; : ebp opsize: bp ; : esi opsize: si ; : edi opsize: di ;
35
36  \ Freischalten der Adressleitung a20
37  \ ----------------------------------
38  \ Mit himem.sys in der config.sys wird die Adressleitung 20 normalerweise
39  \ freigeschaltet. Hat man aus irgendeinem Grunde keine himem.sys in der
40  \ config.sys, dann kann man zur Freigabe des Zugriffs auf Adressen oberhalb
41  \ ffff:ffff das folgende Wort free-a20 verwenden.
```

```
42
43  forth definitions
44
45  code free-a20 ( -- )                          cli
46       begin 64 # al in  02 # al and  0= until  0d1 # al mov  64 # al out
47       begin 64 # al in  02 # al and  0= until  0df # al mov  60 # al out
48       begin 64 # al in  02 # al and  0= until  sti  next end-code
49
50  : a20? ( -- fl ) 0ffff 10 l@ 0. l@ <> 0= ;    \ fl=-1  -->  a20 gesperrt
51
52
53  \ Segmente fs und gs (im PC ab 80486 vorhanden) bearbeiten
54  \ -------------------------------------------------
55  \ cs ds es benoetigen im vorliegenden Artikel keine Sonderbehandlung.
56
57  code fs@ ( -- cc ) 0f c, 0a0 c, next end-code \ f-Segment  holen: fs push
58  code gs@ ( -- cc ) 0f c, 0a8 c, next end-code \ g-Segment  holen: gs push
59  code fs! ( cc -- ) 0f c, 0a1 c, next end-code \ f-Segment setzen: fs pop
60  code gs! ( cc -- ) 0f c, 0a9 c, next end-code \ g-Segment setzen: gs pop
61
62
63  \ Datenverkehr ueber den gesamten 32-Bit-Systembereich
64  \ -----------------------------------------------------
65  \ Achtung: Die folgenden Store- und Fetch-Befehle laufen ueber das spezielle
66  \ Segment-Register fs . Will man die echten physikalischen Adressen erreichen,
67  \ so muss man fs vorher (per 0 fs!) auf 0 setzen. Im Wort dump-32 wird das
68  \ Segment fs vorher sichergestellt, waehrend des Dumpens mit fs=0 betrieben und
69  \ nach Verlassen von dump-32 wieder restauriert.
70
71  \ Fuer die folgenden Worte c@-32 bis !-32  bedeutet:
72
73  \ Punktzahl-Eingaben werden immer so uebertragen, dass sie im RAM in
74  \ Little-Endian-Form erscheinen. Beispiel: 12345678. geht in 78563412 ueber.
75  \ (Dabei werden je zwei Hexziffern in einem Byte untergebracht. Bei 12 steht
76  \ die 1 in den oberen vier Bits, auf dem Bildschirm links, die 2 in den
77  \ unteren, usw.) Umgekehrt wird aus 78563412 im RAM nach Einholen auf den
78  \ Stack (per @-32) und Ausdrucken (per ud.) wieder 12345678 .
79
80  \ ad. = 32-Bit-Adresse, Stack-Eingabe als Punktzahl.
81  \   c = Unteres Byte von 16-Bit-Zahl. Beispiel: 0034 von 1234
82  \  cc = Normale 16-Bit-Zahl. Beispiel: 1234, abgespeichert als 3412
83  \   d = 32-Bit-Wert, Ein- oder Ausgabe als Punktzahl
84
85  code  c@-32 ( ad. -- c )  \ Unmittelbares Byte c von Adresse ad. zum Stack
86       ebx pop  opsize: 0c1 c, 0c3 c, 10 c, ( 10 # ebx rol )  ax ax xor
87       fs: adrsize: 8a c, 03 c, ( fs:[ebx] al mov )  1push end-code
88
89  code  c!-32 ( c ad. -- )  \ Byte c vom Stack nach Adresse ad. speichern
90       ebx pop  opsize: 0c1 c, 0c3 c, 10 c, ( 10 # ebx rol )     ax pop
91       fs: adrsize: 88 c, 03 c, ( al fs:[ebx] mov )   next end-code
92
93  code cc@-32 ( ad. -- cc ) \ Doppelbyte cc von Adresse ad. zum Stack
94       ebx pop  opsize: 0c1 c, 0c3 c, 10 c, ( 10 # ebx rol )
95       fs: adrsize: 8B c, 03 c, ( fs:[ebx] ax mov )  1push end-code
96
97  code cc!-32 ( cc ad. -- ) \ Doppelbyte cc nach Adresse ad. speichern
98       ebx pop  opsize: 0c1 c, 0c3 c, 10 c, ( 10 # ebx rol )     ax pop
99       fs: adrsize: 89 c, 03 c,  ( ax fs:[ebx] mov )  next end-code
100
101 code   @-32 ( ad. -- d )  \ 32-Bit-Wert d von Adresse ad. zum Stack (32-Bit)
```

```
102       ebx pop  opsize: 0c1 c, 0c3 c, 10 c, ( 10 # ebx rol )
103                opsize: fs: adrsize: 8b c, 03 c, ( fs:[ebx] eax mov )
104                opsize: 0c1 c, 0c0 c, 10 c, ( 10 # eax rol )    eax push
105                                                        next end-code
106   code   !-32 ( d ad. -- )  \ 32-Bit-Wert d in 4 Bytes ab Adresse ad. legen
107       ebx pop  opsize: 0c1 c, 0c3 c, 10 c, ( 10 # ebx rol )
108       eax pop  opsize: 0c1 c, 0c0 c, 10 c, ( 10 # eax rol )
109       opsize: fs: adrsize: 89 c, 03 c, ( eax fs:[ebx] mov )
110                                                        next end-code
111
112   \ Absicherung gegen RAM-Bereichs-Ueberschreitung
113   \ ---------------------------------------------
114   \ Man achte auf fs (lineare Adressen erfordern 0 fs!). Dieser RAM-Test schleift
115   \ beliebige Eingaben fuer ad. durch. Im weiter unten stehenden Forth-Wort
116   \ getdiskimage wird vor der Fehlermeldung erst noch 161f00 hinzuaddiert, um
117   \ sicherzustellen, dass das gesamte Disk-Image bis zu seinem Ende ins (wirklich
118   \ vorhandene) RAM passt. Anschliessend erscheint wieder die Anfangsadresse auf
119   \ dem Stack.
120
121   : ramtest ( ad. -- ad./abort )      \ ad. = Punktzahl (doppeltgenau)
122       fs@  -rot                       \ fs    aufbewahren
123       2dup @-32 2swap                 \ (ad.) aufbewahren
124       5a5a5a5a. 2over !-32 2dup @-32 5a5a5a5a. d= -rot
125       a5a5a5a5. 2over !-32 2dup @-32 a5a5a5a5. d= -rot
126       2swap and not >r
127       2swap 2over !-32
128       rot fs!
129       r> cr abort" Bei dieser 32-Bit-Eingabe wird RAMmax ueberschritten!" ;
130
131   \ Speicherbereich im 32-Bit-System dumpen
132   \ ---------------------------------------
133   \ Ohne das gleich folgende Wort dump-32 sind Analysen des vollen RAM-Bereichs
134   \ kaum denkbar. Will man in einem 'RawWrite'- oder 'DiskCopy'-Verfahren das
135   \ Disketten-Image im RAM (auf dem Kopierwege von Diskette zu Diskette) aendern
136   \ oder/und anpassen, so kommt man mit diesem dump-32 und den obigen
137   \ 32-Bit-@-und-!-Worten durch, und ich kann die weitere Aufbereitung dem Leser
138   \ oder der Leserin ueberlassen.
139
140   code  2tuck  ( d1 d2 -- d2 d1 d2 )
141      eax pop   ecx pop
142      eax push  ecx push  eax push
143      next end-code
144
145   : d0<= ( d -- fl ) 0. d> not ;
146
147   variable ascfilt                    \ Bei dump nur ASCII-Zeichen ausgeben ?
148            ascfilt on                 \ Vorgabe: Ja
149   variable fs-dump-32                 \ Segment-Wert fuer dump-32
150          0 fs-dump-32 !               \ Vorgabe 0 (Speicher absolut linear)
151
152   : ?ascii ( n1 -- n2 )               \ Nicht-ASCII-Zeichen --> Punkt ?
153       ascfilt @                       \ true = ja
154       if
155         dup 20 7e between not         \ nicht ASCII ?
156         if  drop 2e  then             \ dann Punkt
157       else
158         dup    7  =                   \ Bell
159         over   8  =  or               \ Del
160         over   a  =  or               \ Linefeed
161         over   d  =  or               \ cr
```

```
162         over  1b  =  or                \ esc
163         over 0ff  =  or                \ Bell usw. ?
164         if  drop bl  then              \ dann Blank
165       then ;
166
167   : dump^ ( -- )
168       ." 0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F  0123456789ABCDEF" ;
169
170   : dump-32 ( ad. len. -- )          \ Stop: SPACE. Weiter: SPACE. Exit: CR CR
171       cr >r >r >r >r
172       fs@ fs-dump-32 @ fs!             \ Mehr: + . Nochmal zurueck: -
173       base @ hex r> r> r> r> 2over     \ Anzeige in Hex
174       10 mu/mod 2drop >r 2swap r@ 0 d- r>
175       dup 3 * dup 17 > if 1+ then 0a + (cursor) nip 0b swap at dump^
176       (cursor) nip at bold ." \/" 3c + (cursor) nip at ." V" attoff cr
177       begin
178         begin
179           2dup <# # # # # # # # # #> type space space     \ Adresse anzeigen
180            8 0 do 2dup c@-32 0 <# # # #> type space 1. d+ loop space
181            8 0 do 2dup c@-32 0 <# # # #> type space 1. d+ loop space  10. d-
182           10 0 do 2dup c@-32           ?ascii  emit 1. d+ loop cr
183           2swap 10. d- 2tuck
184           d0<= stop? or                 \ Ende oder Abbruch ?
185         until
186         (cursor) dup 18 = if 0 0 at 0b spaces dump^ then at
187         bold (cursor) -1 0 (frame) ." Weiter: +   Zurueck: -   Exit: RET"
188         attoff at
189         key
190         dup ascii + = if  drop 2swap 2drop 100. 2swap
191                           false else
192             ascii - = if  2swap 2drop 100. 2swap 200. d- dark 0b 0 at dump^ cr
193                           false else
194         2drop 2drop base ! fs! true  then  then
195       until 0c 0d (frame) cr ;
196
197   \ ------------------------------------------------------------------------
198
199   hex
200
201   \ Das folgende Wort (getdiskimage) ist die Vorstufe zu getdiskimage. In
202   \ seinem Kern habe ich mich an (getdiskbootsect) aus dem Artikel 'Bootmanager
203   \ und FAT-Reparatur: Vierter Fort(h)schritt (patchbares BIOS im RAM)'
204   \ orientiert (siehe VD-Heft 2/2009). Fuer das Vorliegende wollte ich eine
205   \ 3.5-Zoll-HD-Diskette 'am Stueck' ins RAM schreiben und von dort aus
206   \ weiterverarbeiten und gegebenenfalls auch wieder in eine ebensolche oder
207   \ dieselbe Diskette zurueckschreiben koennen. Meine Experimente mit dem
208   \ erweiterten Interrupt 13h/42h brachten keinen Erfolg. Die Disketten liessen
209   \ sich gar nicht erst ansprechen. Auch ein Versuch mit Int 25h verlief
210   \ ergebnislos. Dabei hatte ich gedacht, mit meiner 4-Gigabyte-Erweiterung
211   \ 'Real-Mode-32-Bit-Erweiterung fuer Turbo-Forth' aus dem Artikel im VD-Heft
212   \ 2/1998 weiterzukommen [FB98]. Das vorliegende Programm arbeitet in 80d
213   \ Bloecken, die jeweils noch auf Seite 0 und Seite 1 geschaltet werden koennen,
214   \ zu je 18d Sektoren (das ist genau das Schema der 3.5-HD-Disketten, naemlich
215   \ 80d Spuren (tracks) mit den Seiten (heads) 0 und 1 pro Spur zu je 18d
216   \ Sektoren mit je 512d Bytes pro Sektor) im Real-Modus und kommt ohne
217   \ 4-Gigabyte-Erweiterung im Sinne von [FB98] aus.
218
219   \ (getdiskimage) uebertraegt den Inhalt einer 3.5-HD-Diskette sektorweise ins
220   \ RAM, und zwar ab der in der 2Variablen imgbuf aufbewahrten Adresse. Diese
221   \ Adresse kann (und muss) dem (das Low-Level-Wort (getdiskimage) enthaltenden)
```

```
222  \ High-Level-Wort getdiskimage als Parameter mitgegeben werden. Das Wort
223  \ ramtest liefert eine Absicherung gegen den moeglichen Fehler, in imgbuf eine
224  \ Anfangs-Adresse anzugeben, die fuer den Disk-Image-Puffer nicht genuegend
225  \ Platz uebrig laesst.
226
227  \ imgbuf ist als Punktzahl (32 Bit) einzugeben! Der Wert 200000. waere ein
228  \ guter Wert fuer den Beginn von Eigenexperimenten. Er liegt im RAM-Speicher
229  \ als Adresse deutlich ueber 1 Megabyte und kann dort leicht an einem Stueck
230  \ abgelegt werden, ohne anderen Vorhaben in die Quere zu kommen. Mit einfachen
231  \ 16-Bit-Zahlen kaeme man, wie leicht ersichtlich, nicht durch. Der naechstgute
232  \ Wert fuer Eigenexperimente waere dann 400000. (auch wieder Punktzahl). Mit
233  \ 400000. bliebe dann das gesamte schon im ersten Experiment ermittelte
234  \ Disk-Image im (erweiterten) RAM der Analyse zugaengig. Dann 600000. usw. Hat
235  \ man mehr als 280 Megabyte RAM, dann koennte man das Disketten-Abbild
236  \ natuerlich auch beispielsweise nach 10000000. legen - usw., alles hexadezimal
237  \ gesehen.
238
239
240  \ Zu beachten ist, dass in Code-Definitionen (auf einem PC-Kompatiblen) auch
241  \ 32-Bit-Werte streng little-endian erscheinen muessen. Bei Punktzahlen in
242  \ 16-Bit-Forth-Systemen (auf jeden Fall in Turbo-Forth) treten High- und
243  \ Low-Anteil auf dem Stack (und auch in den 2Variablen und im RAM, was man per
244  \ 12345678. pad 2! und dann pad 50 dump nachpruefen kann) in sich
245  \ vertauscht auf. Genauer: Eine Tastatureingabe von 12345678. findet sich auf
246  \ dem Stack (von den niedrigen Adressen her gesehen) als 43218765 wieder. Dabei
247  \ beachte man dann noch, dass beim Ausdumpen (auf dem Bildschirm) die hoeheren
248  \ Bits im Byte links stehen, dass die einzelnen Bytes (1 Byte = 2 Hexziffern)
249  \ also (als Paar) so wiedergegeben werden, wie wir sie lesen, eben nicht in
250  \ Little-Endian-Manier, sondern beispielsweise 43 zu lesen als 34, 21 als 12
251  \ usw.
252
253  \ Im Uebrigen besteht auch die Moeglichkeit, im DOS-System (wo Turbo-Forth
254  \ laeuft) eine RAM-Disk von guten 1.44d Megabyte einzurichten und dort eine
255  \ (beliebige) Textdatei von 1.44 MB Laenge bereitzustellen. Das Abbild der
256  \ Diskette (das Disk-Image) koennte man dann am Anfang dieser Datei beginnen
257  \ lassen und haette zu ihrer Weiterverarbeitung die gesamte Datei-Organisation
258  \ des DOS-Systems zur Verfuegung, in voller Laenge. Das waere dann also
259  \ das, was man beispielsweise auch von Rawwrite erwarten wuerde.
260
261  \ Es werden bei (getdiskimage) immer 18d Sektoren (eine Spur) von der Diskette
262  \ in den (unterhalb von einem Megabyte, innerhalb eines 'normalen' Segmentes
263  \ des DOS-Systems liegenden) Puffer trackbuf gelesen. Dann wird (jedesmal) der
264  \ Inhalt von trackbuf ins 'Diskimage' (mit Anfangsadresse in der 2Variablen
265  \ imgbuf) uebertragen. Das Ganze wiederholt sich 80d*2-mal (= Anzahl der Spuren
266  \ auf einer 3.5-HD-Diskette mit jeweils Seite 0 und Seite 1). Die Nummer der
267  \ jeweiligen Spur (mit 18d Sektoren) steht im Register ch. Die jeweilige Seite
268  \ steht in dh und wird bei Uebertragung der jeweiligen Spur von Seite 0 auf 1
269  \ geschaltet und mit derselben Spur wiederholt. Spuren und Seiten sind ab 0 zu
270  \ zaehlen, Sektoren werden (bei der in der Code-Definition (getdiskimage)
271  \ verwendeten CHS-Auffassung ab 1 gezaehlt! ebx wird doppelbyteweise (je
272  \ 16 Bit) per ebx inc ebx inc fortgeschaltet und enthaelt die Adresse des
273  \ naechsten freien Bytes nach dem jeweils gerade uebertragenen Doppelbyte.
274
275  \ Im RAM des urspruenglichen IBM-PCs (unterhalb der 1-Megabyte-Grenze) laesst
276  \ sich keine ganze 3.5-HD-Diskette (1,44 MB) unterbringen. Also muss ein Puffer
277  \ (unterhalb der 1-MB-Grenze) eingerichtet werden, der die Uebertragung der
278  \ einzulesenden Disk portionsweise erledigt. Als 'Portionen' bieten sich die
279  \ einzelnen Spuren der Diskette an, 80d an der Zahl. Pro Spur dann jeweils noch
280  \ Seite 0 und 1. Pro Spur 18d Sektoren zu je 512d Bytes. Insgesamt sind das
281  \ 80d * 2 * 18d * 512d = 50h * 2 * 12h * 200h = 168000h = 1474560d Bytes.
```

```
282
283  \ Es folgt der 18d-Sektor-Puffer (unter dem Namen trackbuf). Bei Aufruf von
284  \ trackbuf wird die Anfangsadresse dieses Puffers (als Punktzahl zu
285  \ interpretieren) als Offset, bezogen auf das DSegment des gerade in Betrieb
286  \ befindlichen Turbo-Forth-Systems, auf den Stack gelegt.
287
288  2410 allot here       \ Platz fuer mindestens 18d Sektoren = 512d * 18d Bytes
289  here 0f and -         \ trackbuf an Paragraphenanfang
290  2400 -                \ Anfang des Uebertragungs-Puffers
291  constant trackbuf     \ Liefert Anfangsadresse des Transfer-Puffers fuer 1 Spur
292
293  hex                   \ Alle als Eingabe zu wertenden Zahlen in diesem Listing
294                        \ sind hexadezimal zu verstehen!
295
296  \ Der RAM-Bereich zur Aufnahme des eigentlich interessierenden Puffers fuer
297  \ das gesamte Disketten-Image kann mit der Eingabe xxxxxxxx. getdiskimage
298  \ beliebig gewaehlt werden. Dabei wird im Vorliegenden keinerlei Ruecksicht auf
299  \ irgendwelche RAM-Verwaltungen genommen. Allerdings wird bei der Eingabe (mit
300  \ dem Wort ramtest) dafuer gesorgt, dass das Programm aussteigt, wenn versucht
301  \ wird, RAM zu verwenden, das gar nicht vorhanden ist. Hier der Image-Puffer:
302
303  2variable imgbuf      \ Enthaelt Anfangsadresse des Disk-Image-Puffers. Diese
304                        \ wird mit getdiskimage (als Punktzahl) eingegeben und
305                        \ steht auch nach dem Einlesen der Diskette (zur eventl.
306                        \ Bearbeitung des Disk-Images) weiterhin zur Verfuegung.
307
308  2variable imgbyte     \ Enthaelt (als Punktzahl) die Adresse des zuletzt
309                        \ eingelesenen Doppelbytes.
310
311   variable spur        \ Letzte erreichte Spur
312   variable seite       \ Letzte erreichte Seite
313   variable flag        \ Fehler-Flag (0/1) fuer (getdiskimage) :  1 = Fehler
314
315  \ Das Disketten-Laufwerk ist ein stark fehleranfaelliges Geraet, das viel
316  \ Mechanik enthaelt. Im folgenden Wort (getdiskimage) habe ich deshalb eine
317  \ Schleife mit drei Versuchen zur Abfrage eingebaut. Diese Schleife tritt bei
318  \ jeder der 80d Spuren (zu je Seite 0 und Seite 1) einer einzulesenden
319  \ 3.5-HD-Diskette in Aktion. Ob ein Fehler vorliegt, wird bei jeder Spur (pro
320  \ Seite nach dem Lese-Interrupt 13h/2 durch Abfrage des Flag-Register-Bits cf
321  \ ermittelt. Wenn es bis nach dem dritten Lese-Versuch nicht geklappt hat,
322  \ bricht das Programm ab und hinterlaesst in den 2Variablen imgbuf und imgbyte
323  \ und in den Variablen spur seite flag die bis dahin erreichten
324  \ (Fehler-)Parameter.
325
326  code (getdiskimage)   ( -- )
327            ds push     \ ds nach es uebertragen,
328            es pop      \ mit Stack als Zwischenstation.
329         3 # di mov     \ Zaehler fuer drei Disketten-Leseversuche ansetzen.
330                begin   \ ------------- Bis zu 3 Leseversuche -------------
331    0 # flag #) cmp 0<>  \ ------------- Laufwerk initiieren ? -------------
332                if
333          dl dl xor     \ dl =  0 = Diskettenlaufwerk a:
334          ah ah xor     \ Funktion 0 von Int 13h aufrufen, also
335             13 int     \ das Disketten-System zuruecksetzen.
336                then    \ ---------- Bemerkung hierzu siehe unten ---------
337                        \ --------- 1 Spur (18d Sektoren) einlesen --------
338    seite #) dh mov     \ Seitennummer dieses Spurendurchgangs
339    spur  #) ch mov     \ Spurennummer dieses Spurendurchgangs
340           trackbuf     \ 18d Sektoren (= 1 Spur = 9216d = 2400h Byte) einlesen
341           # bx mov     \ bx = Anfangs-Offset des Spur-Transfer-Puffers
```

```
342         1 # cl mov      \ cl = Sektor 1 in der momentanen Spur (von 18d Sektoren)
343          dl dl xor      \ dl = 0 = Diskettenlaufwerk a:
344         2 # ah mov      \ ah = 2 (Lesefunktion von Int 13h)
345        12 # al mov      \ al = 12h = 18d Sektoren = 1 Spur von ch/dh einlesen
346             13 int      \ Lese-Interrupt aufrufen.
347           u>= if        \ Wenn danach cf = 0 (kein Lesefehler), dann Aussprung
348          di di xor      \ aus der Schleife per di = 0 vorbereiten und
349    0 # flag #) mov      \ flag = 0 (kein Fehler) setzen.
350                else      \ Wenn jedoch cf <> 0 (Lesefehler),
351             di dec       \ dann Zaehler fuer 3 Leseversuche dekrementieren
352    1 # flag #) mov      \ und flag = 1 (Fehler) setzen.
353                then
354         0 # di cmp 0=   \ Aussprung aus der 3er-Schleife mit flag = 0 oder 1.
355                until    \ ------------------------------------------------
356    0 # flag #) cmp 0=
357                if       \ di ist jetzt wieder frei verwendbar.
358         trackbuf        \ Anfang des Puffers zur Uebertragung der naechsten Spur
359           # di mov      \ nach di legen.
360    imgbyte     #) push  \ Hi-Anteil der 2Variablen imgbyte auf den Stack legen.
361    imgbyte 2 + #) push  \ Lo-Anteil der 2Variablen imgbyte auf den Stack legen.
362           ebx pop       \ Disk-Image-Zeiger liegt jetzt little-endian in ebx.
363               clc       \ ------------------------------------------------
364           1200 do       \ trackbuf>imgbuf: 1200h = 4608d Doppelbytes = 9216 Bytes
365                         \ ------------------------------------------------
366                         \ do-loop benoetigt cx als Zaehler. cl wird neu gesetzt.
367      0 [di] ax mov      \ di = trackbuf (16 Bit). Keine edi-Erweiterung noetig.
368             di inc      \ ax uebertraegt ein Doppelbyte. Die Adresse di muss
369             di inc      \ sich also anschliessend um 2 erhoehen: di = di+2.
370       adrsize: fs:      \ Auf Segment fs bezogen und mit 32-Bit-Adressbreite.
371           89 c, 03 c,   \ Das entspricht ax [ebx] mov (wegen adrsize).
372                         \ ------------------------------------------------
373           ebx inc       \ ebx muss 1,44 Megabyte sammeln. bx allein wuerde nicht
374           ebx inc       \ genuegen. ax = 2 Bytes --> ebx = ebx + 2.
375               loop      \ ------------------------------------------------
376           ebx push      \ Adresse des ersten freien Bytes nach dem zuletzt
377    imgbyte 2 + #) pop   \ eingelesenen Doppelbyte. Lo-Anteil davon forthrichtig
378    imgbyte     #) pop   \ zur 2Variablen imgbyte legen, Hi-Anteil entsprechend.
379                then      \ ------------------------------------------------
380       next end-code
381                         \ -- Bemerkung zu int 13h/0 (Laufwerk initiieren) -
382                         \ Es klingt logisch, das Disketten-Laufwerk vor dem Lesen
383                         \ einer jeden weiteren Spur (davon enthaelt die 3.5-Disk
384                         \ 80d pro Seite, also 160d Stueck) 'vorsichtshalber'
385                         \ frisch zu initiieren. Das hatte ich urspruenglich so
386                         \ gemacht. Das vom Laufwerk dabei erzeugte Geraeusch war
387                         \ nicht zu ueberhoeren. Und man spuerte foermlich, wie
388                         \ der Laufwerk-Mechanismus bei jeder Spur nochmal
389                         \ 'nachfassen' musste. Die jetzt gefundene Loesung ist
390                         \ wesentlich besser: Das Laufwerksgeraeusch ist kaum noch
391                         \ wahrnehmbar, kaum merklich lauter als der Luefter. Die
392                         \ einzelnen Spuren fuegen sich geschmeidig aneinander -
393                         \ und verloren geht nichts, da ja im Fehlerfall erst zwei
394                         \ weitere Leseversuche vorgenommen werden, bevor das
395                         \ System einen Fehler vermeldet und aussteigt.
396
397   : printparams  ( -- ) \ Ausdruck von flag-Meldung imgbuf imgbyte seite spur
398       cr cr
399       ." ================================================" cr
400       flag @ 1 = if ." Fehler beim Einlesen des Disketten-Abbildes"    cr then
401       flag @ 0 = if ." Die Diskette wurde ohne Lesefehler eingelesen"  cr then
```

```
402       ." ------------------------------------------------" cr
403       ." Erstes  eingelesenes Byte an RAM-Adresse : " imgbuf  2@       ud. cr
404       ." Letztes eingelesenes Byte an RAM-Adresse : " imgbyte 2@ 1. d- ud. cr
405       ." Letzte Seite; oder erste, die Fehler hat : " seite    @        u. cr
406       ." Letzte Spur ; oder erste, die Fehler hat : " spur     @        u. cr
407       ." ------------------------------------------------" cr
408       ." Falls Sp/S = 0/0 : Wirklich Diskette im Laufwerk?" cr
409       ." ================================================" cr cr ;
410
411   : getdiskimage ( imgbuf -- )
412       0 fs!            \ Disk-Image im RAM auf f-Segment = 0 beziehen.
413       161f00. d+       \ Laenge der Diskette (in Bytes) hinzuaddieren.
414       ramtest          \ Passt das Disk-Image ueberhaupt ins RAM?
415       161f00. d-       \ Wieder zum Anfang des Disk-Image-Puffers gehen.
416       2dup             \ Zunaechst lege ich den
417       imgbuf  2!       \ Anfang des Disk-Image-Puffers in die 2Variable imgbuf
418       imgbyte 2!       \ und dann   (zunaechst auch)   in die 2Variable imgbyte.
419       0 spur   !       \ Letzte erreichte Spur  zunaechst auf 0 setzen.
420       0 seite  !       \ Letzte erreichte Seite zunaechst auf 0 setzen.
421       0 flag   !       \ Fehler-Flag            zunaechst auf 0 setzen.
422       begin
423         (getdiskimage)
424         seite @ 0 =    \ Spuren- und Seitenzaehlung beginnt bei 0. Seiten: 0/1.
425         if             \ Wenn bei einer bestimmten Spur die Seite auf 0 steht,
426           1 seite !    \ dann Seite auf 1 schalten und Spur beibehalten.
427         else           \ Ansonsten steht die Seite auf 1.
428           0 seite !    \ Dann Seite wieder auf 0 setzen
429           spur 1+!     \ und Spur (es gibt 50h davon) weiterschalten.
430         then
431         spur @ 50 =    \ Verwendbare Spuren: 0-4f. Wenn 50h erreicht ist,
432         flag @  0<> or \ oder auch schon vorher, wenn naemlich ein Fehler
433       until            \ aufgetreten ist, dann begin-until-Schleife verlassen.
434       seite @ 1 =      \ Spuren- und Seiten-Fortschaltung rueckgaengig machen.
435       if               \ if-then hier analog zu if-then eben.
436         0 seite !
437       else
438         1 seite !
439         spur 1-!
440       then
441       printparams ;    \ Bildschirm-Ausgabe der (Fehler-)Parameter.
442
443   \ ============================================================================
```

SwiftForth–IDE
Interactiv Development Environment — Evaluation Version
Kurz vorgestellt

Michael Kalus

Als stolzer Besitzer eines gebrauchten Laptop[1] mit Windows XP habe ich mir das swiftForth[2] angesehen, das von der Forth Inc. zum kostenlosen download angeboten wird.

Die selbst entpackende exe–Datei war schnell geholt, reibungslos installiert und swiftForth startete anstandslos seine Kommando–Konsole:

```
SwiftForth i386-Win32 3.2.5 10-Feb-2011
```

Es trägt sich ordnungsgemäß unter *alle Programme* ein und macht ein Icon auf den Desktop, wenn man will, benimmt sich also anständig in der Windows–Umgebung, es waren keinerlei Klimmzüge nötig.

Die IDE besteht aus einer Forth–Konsole mit zwei Kopfzeilen, eine Menüzeile und darunter ein toolbar, ist also übersichtlich ohne Schnickschnack und erschließt sich schnell dem Benutzer. Im Menüeintrag *File* sind *Include, Edit, Print, Save command window, save keyboard history* und *session log* da. *Edit* öffnet den Windows–Editor, um Forth–Quellen zu editieren. Das Konsolenfenster selbst kennt auch copy und paste. In den *Options* kann man den Editor bestimmen, Fonts und Farben, zwei Systemwarnungen aktivieren (redefinitions und case ambiguity), und Monitoring beim File–Laden auswählen (s. Bilder).

Der *ToolsButton* ist erklärungsbedürftig. Darin gibt es *Words, Watch, Memory, History, Run sowie Optional Packages.*

Das, was dort kurz *Watch* genannt wird, meint eigentlich ein separates Fenster, in dem eine Liste von *watch points* eingesehen werden kann. Von der Konsole aus werden diese Beobachtungspunkte mit `(adr -- ) WATCH` gesetzt und über einen automatischen refresh dann in jenem Fenster angezeigt, derweil man in der swiftForth–Konsole weiterarbeitet, also Code ausprobiert. Im swiftForth Reference Manual ist gut erklärt, wie man mit diesen *watch points* umgeht. Der Button bei den Tools bringt lediglich das extra Fenster zur Anzeige oder blendet es aus, gesetzt werden müssen die *watch points* von der Konsole aus.

Auch nett ist das *Memory* tool. Es stellt ein dynamisches Memory Display in eigenem Fenster zur Verfügung. Darin wird ein ausgewählter Speicherbereich abgebildet und man kann dessen Veränderungen verfolgen, während man im Forth arbeitet — im Gegensatz zu dem klassischen `DUMP` welches einen Bereich ja nur einmalig listet. `( adr -- ) MEM` ruft dieses tool von der Konsole aus auf. Auch hier gilt: Setzen der gewünschten Stelle geschieht von der Forth–Konsole aus, der tool button der IDE blendet den Bereich nur ein und aus.

History öffnet das key history Fenster, aus dem man per copy&paste schöpfen kann. Und *Run* ermöglicht es, aus dem Forth heraus andere Programme zu starten, indem eine Kommandozeile dafür geöffnet wird.

Die *Optional Packages* fand ich besonders interessant und hilfreich, werden hier doch eine Menge Demoprogramme angeboten, die einwandfrei eingebunden und gestartet werden können. Wie der in einem seperaten Window hüpfende Ball. Oder, nützlicher, das `terminal.f`, das nach `9600 TERM COM1` ein einfaches Terminal realisiert und wirklich auf Anhieb ging — was mich besonders gefreut hat, da mein alter Laptop noch einen `COM1`–Stecker hat und ich gerne schon mal mit Micros wie dem atmega168 und amforth experimentiere. Die serielle Verbindung zu so einem Board gelang auf Anhieb.

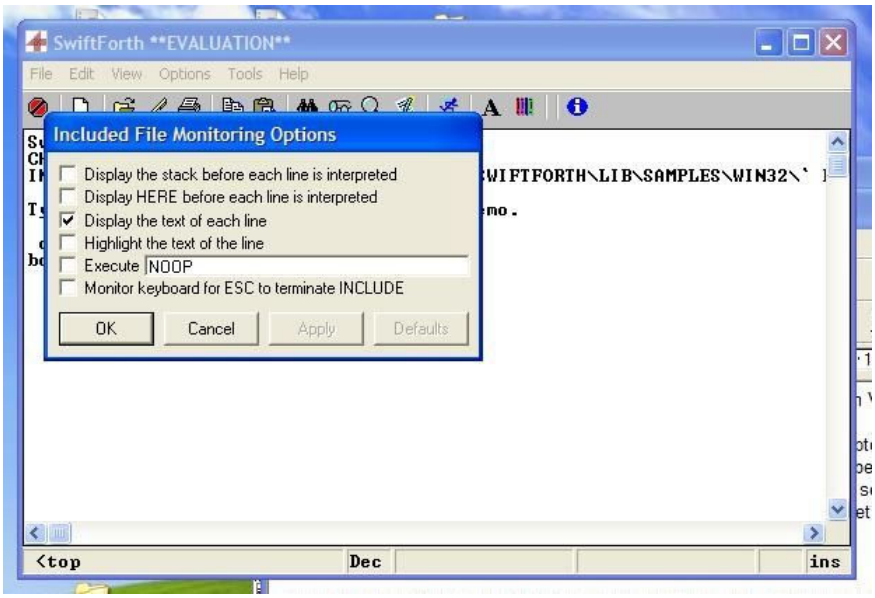
Und *Help* hat seinen Namen zu Recht, führt es doch zu einer umfangreichen Dokumentation, die der Evaluationsversion beigelegt ist. Da ist die Microsoft® Win32® Programmer's Reference drin, und als PDF–Dateien das komplette Forth Programmer's Handbook, das swiftForth Reference Manual und der ANS Forth Standard. Damit kann man schon mal was anfangen. Dann gibt es noch einen Button, der via Browser gleich zur Forth Inc. führt, zu all den schönen Sachen, die man dort als Programmierer erwerben kann, will man Windows programmieren mit Forth vertiefen. Man erhält den Eindruck, gut aufgehoben und unterstützt zu werden. Die Preise dafür mag sich jeder selbst ansehen.

Der toolbar (Werkzeugleiste) enthält einige der genannnten Features aus dem Menü, das erleichtert den Zugriff, erscheint praktisch und auf das Wesentliche fürs Arbeiten beschränkt.

Die Forth–Konsole selbst lässt sich zeilenweise wie gewohnt editeren. Während einer Sitzung kann man mit den up– und down–Tasten hin und her blättern zu seinen vorherigen Eingaben. Diese *command history* ist bei der nächsten Sitzung allerdings wieder leer. Da nützt

[1] 2002; hp compac nx9005, AthlonXP 2800+,1.25GHz, 192MB Ram, Windows XP home edition

[2] http://www.forth.com/swiftforth/index.html



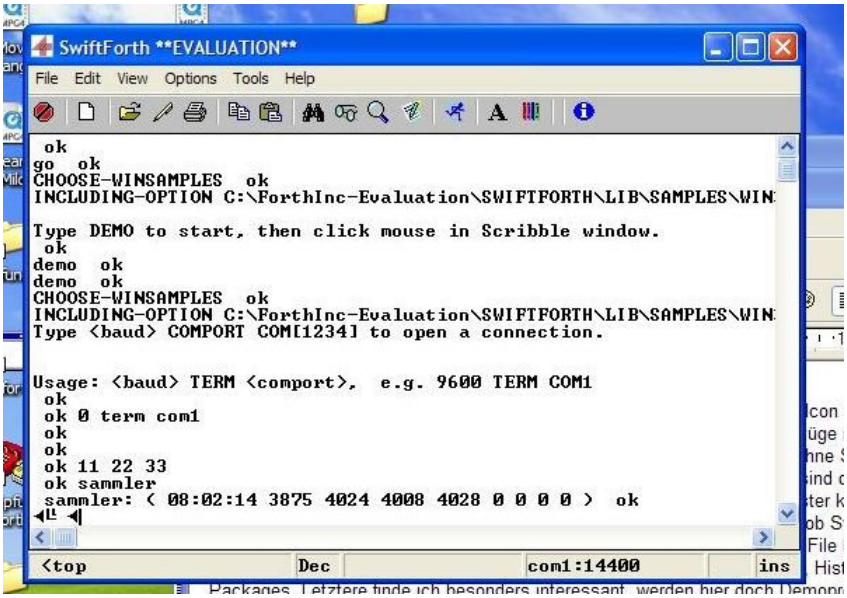
Screenshot der SwiftForth–IDE in Aktion

auch das *save keyboard history* des Menüs in eine Datei nichts. Aber man kann dann in jener Datei nachsehen und per copy&paste zurückholen was man braucht. Das *command history window* zeigt alle Eingaben der Sitzung.

Dann gibt es unten am Rand eine Zeile zur Status–Anzeige, in der man sehen kann, was auf dem Stack liegt, welche Zahlenbasis eingestellt ist, und wie der editmodus steht — hübsches feature: Klickt man auf die Angaben in der Statuszeile, rollt man durch die Einstellungen *hex, dec, oct, bin* bzw *ins, ovr*. Hübsch auch der *Words browser*. Klickt man darauf, werden die Worte des aktuellen Vokabulars in einem separaten Fenster angeboten. Klickt man dort auf ein Wort, wird dessen Quelle in der Forth–Konsole angezeigt — einfach und praktisch.

Beendet man die Arbeit mit der Evaluationsversion, versucht swiftForth noch schnell den Browser zu öffnen und die Forth–Inc.–Seite anzuzeigen, was für meinen Geschmack etwas zu viel Reklame ist. Aber ich denke mal, das tut die Vollversion dann nicht mehr.

Man hat den Eindruck, dass man mit *swiftForth* tatsächlich ein Werkzeug bekommt, mit dem in der Windows–Umgebung etwas Brauchbares erstellt werden kann. Und dann lohnen sich auch die Euros für eine lizensierte Vollversion.



SwiftForth auf PC wird Terminal via COM1 an Forth in atmega168 (Sammler).

amforth: rs485 half duplex, mpc und andere, seltsame Tiere

Erich Wälde

Auf dem Linuxtag 2010 habe ich (am Stand der Forth–Gesellschaft) ein paar Platinen mit atmega–32–Kontrollern gezeigt, die zusammen diverse Daten erfassten: Temperatur, Luftfeuchte, Luftdruck und den Stromverbrauch am Stand. Jeder Kontroller fragt selbstständig z. B. alle 10 Sekunden die Werte der angeschlossenen Sensoren ab. Die Werte werden in physikalische Einheiten umgerechnet und dann so lange gesammelt (genauer aufaddiert), bis *jemand* die Daten abholt. Dann wird der Mittelwert gebildet, und zusammen mit Minimum, Maximum und der Anzahl der Messwerte ausgegeben. Ähnlich wird mit den aufgesammelten Zählwerten des Stromverbrauchs verfahren.

Das wirft natürlich Fragen auf: Wer ist dieser *jemand*, der *Einsammler*, wie kommuniziert der mit dem Kontroller? Hängen alle Kontroller an einem Kabel? Und wenn ja, wie geschieht es, dass nicht alle Kontroller wild durcheinander quasseln? Dieser Artikel zeigt, wie man Ordnung auf das Verbindungskabel bringt.

Mein Dank gilt Matthias Trute, Luboš Pěkný, Bernd Paysan, Michael Kalus, Martin Bitter und sicher noch anderen für ihre Hilfe.

Assembler oder Forth?

*Afterwards everything is **obvious**.* Auf die Frage von Bernd, warum denn das alles in Assembler und nicht in Forth geschrieben sei, wusste ich zunächst keine Antwort. Es gibt folgende Stellen, die zu `amforth` gehören, in Assembler geschrieben sind, und die sich nicht so ganz einfach verbiegen lassen:

- `tx` muss den Lesen/Schreiben–Pin bedienen, während es den Ausgabepuffer abarbeitet.
- Die Adresse dieses Pins wollte ich konfigurierbar haben. Außerdem wird diese Information in den Assembler–Worten `tx` und `txc-isr` benutzt.
- (`txc-isr`) am Ende einer Übertragung, die asynchron abgearbeitet wird, muss dieser Pin wieder auf Lesen zurückgesetzt werden. Der usart–transmit–complete interrupt bietet sich dafür an. Das Bit zu löschen, war in Assembler viel zu einfach.
- in `rx` wird die Stationadresse benötigt, wenn sich der Kontroller im mpc–Modus befindet. Die muss also unverlierbar und von Assembler aus zugänglich sein.
- Die Prompts sind nicht *deferred*, d.h. auch an dieser Stelle ist es einfacher, den Assembler–Code zu ändern.

Das Initialisieren des Lesen/Schreiben–Pins, das gute Benehmen am Bus (schweigsam soll der Kontroller sein und nur nach Aufforderung sprechen) und weitere Worte wären direkt in Forth schneller machbar. Teilweise habe ich sogar `g4` von Michael Kalus benutzt, um Assembler zu erzeugen — teilweise dann aber mit manueller Nacharbeit verziert.

Ich wollte ein System, welches sich am Bus sofort korrekt benimmt. Es kann auch über den Bus in Forth programmiert werden. Das funktioniert allerdings nur, wenn die Bedienung des Lesen/Schreiben–Pins schon direkt nach dem Flashen lückenlos klappt. Und ich habe ganz nebenbei eine Menge über das Innenleben von `amforth` gelernt.

RS485: ein Bus für viele Zwecke

Der Einsammler ist ein (perl–)Programm auf einem (Linux–)Rechner, welches über einen Bus, den RS485–Bus [3], mit den Kontrollern redet. Das Signal wird differentiell über ein Adernpaar übertragen. Um ein Bit zu übertragen, wird die Spannungsdifferenz zwischen den Adern A und B entweder positiv (0) oder negativ (1). Die Anordnung der Bits und der zeitliche Ablauf sind gleich, wie bei der normalen seriellen Verbindung (RS232). Will man gleichzeitig Daten lesen und schreiben, dann braucht man zwei Adernpaare, entsprechende Bausteine zur Anschaltung und man redet von einer *full duplex* Konfiguration.

Es geht aber auch mit einem Adernpaar. Dann kann ein Busteilnehmer zu einer bestimmten Zeit nur lesen oder schreiben, nicht beides. Diese Konfiguration heißt *half duplex* und benötigt eine Steuerung, die über Lesen oder Schreiben entscheidet. Viel mehr ist für RS485 nicht definiert. Insbesondere ist kein Protokoll definiert, welches zum Einsatz kommen muss. Der Bus kann beträchtliche Längen überbrücken, bis mindestens 1200 m bei 100 kBit/s. Man kann damit also bequem, falls vorhanden, die Hacienda verkabeln und muss nicht täglich zum Briefkasten reiten.

Um die Anbindung in `amforth` zu machen, sind folgende Aufgaben zu lösen:

- Die Unterstützung für RS485 soll beim Assemblieren wahlweise eingebunden werden können (WANT_RS485)
- Der RS485 W/R Pin soll konfiguierbar sein (z. B. PortD.7)
- applturnkey muss den W/R Pin initialisieren (output, low)
- `tx` muss den Pin vor dem Senden auf *Schreiben* setzen
- Nach der Übertragung muss der Pin wieder auf *Lesen* gesetzt werden (Usart Transmit Complete Interrupt)

RS485: Ansteuerung einbauen

main.asm: WANT_RS485

Um alle Quelltextstellen, die der Unterstützung des RS485 half duplex Modus dienen, selektiv ein– oder auszuschalten, wird in der Datei `main.asm` eine Assembler–Variable definiert:

```
.set WANT_RS485 = 1 ; RS485 half duplex
```

Im Assembler–Quelltext finden sich dann Blöcke der Bauart:

```
.if  WANT_RS485 == 1
  ...
.else
  ...
.endif
```

main.asm: den W/R Pin konfigurieren

Es war mir wichtig, dass der Pin, welcher den Lesen/Schreiben–Zustand am Transceiver einstellt, beim Erstellen von `amforth` konfiguriert werden kann. Zwei weitere Assembler–Variablen werden dafür benötigt:

```
; file: main.asm
...
; rs485_rw_pin: D.7==W/R, low==read
.if  WANT_RS485 == 1
  ; PORTx: use io location, not mem mapped
  .set RS485_RW_PORT = PORTD
  .set RS485_RW_PIN  = 7
.endif
```

`PORTD` wird in den Tiefen der von device.asm eingeschlossenen Dateien auf den Wert `$12` gesetzt. Um diese Definitionen aus `amforth` heraus benutzen zu können, wurden sie entsprechend in `constants` verfrachtet:

```
; file: words/rs485.asm
; asm(RS485_RW_PORT)
; PORTD constant rs485_rw_port

VE_RS485_RW_PORT:
    .dw $FF0D
    .db "rs485_rw_port",0
    .dw VE_HEAD
    .set VE_HEAD = VE_RS485_RW_PORT
XT_RS485_RW_PORT:
    .dw PFA_DOVARIABLE
PFA_RS485_RW_PORT:
    ; use mem mapped location!
    .dw (RS485_RW_PORT+$20)

; asm(RS485_RW_PIN)
; $07 constant rs485_rw_pin

VE_RS485_RW_PIN:
    .dw $FF0C
    .db "rs485_rw_pin"
    .dw VE_HEAD
    .set VE_HEAD = VE_RS485_RW_PIN
XT_RS485_RW_PIN:
    .dw PFA_DOVARIABLE
PFA_RS485_RW_PIN:
    .dw RS485_RW_PIN
```

Das dient nur der Bequemlichkeit.

applturnkey.asm: den W/R Pin initialisieren

Der RS485 W/R Pin soll beim Starten von `amforth` als Ausgang geschaltet werden und auf null gestellt (nicht auf eins). Dazu wird ein entsprechendes Wort in Assembler definiert:

```
; file: words/rs485-init.asm

; : rs485.init ( -- )
;   rs485_rw_port rs485_rw_pin portpin: rs485_rw
;   rs485_rw low
;   rs485_rw pin_output
; ;

VE_RS485_INIT:
    .dw $ff0a
    .db "rs485.init"
    .dw VE_HEAD
    .set VE_HEAD = VE_RS485_INIT
XT_RS485_INIT:
    .dw PFA_RS485_INIT
PFA_RS485_INIT:
    ; set DDR bit (pin_output)
    sbi  (RS485_RW_PORT-1),RS485_RW_PIN
    ; clear PORT bit (receive)
    cbi  RS485_RW_PORT,RS485_RW_PIN
    rjmp DO_NEXT
```

Das Wort `rs485.init` wird in `applturnkey` aufgerufen:

```
; file: words/applturnkey.asm
...
.if WANT_RS485 == 1
    .dw XT_RS485_INIT           ; rs485.init
.endif
```

usart-isr-tx.asm: Schreibpin bedienen

Bis jetzt sind das alles Vorarbeiten. Der W/R Pin muss in der Funktion `tx` bedient werden. `tx` versendet ein Byte über die USART, die serielle Schnittstelle. Vorher muss der W/R Pin auf Schreiben (1) gesetzt werden:

Ich habe auf meinen Kontrollern das Schreiben auf die serielle Schnittstelle mittels Interrupt–Service–Routine eingeschaltet (`.set WANT_ISR_TX = 1`), so dass die hier gezeigten Änderungen eben in der Interrupt–Service–Routine gemacht werden. Das ist reine Geschmacksache.

```
; file: drivers/usart-isr-tx.asm
...
usart_udre_isr:
...
usart_udre_next:
.if WANT_RS485 == 1
  sbi RS485_RW_PORT,RS485_RW_PIN
.endif
  inc xh
  andi xh,usart_tx_mask
  sts usart_tx_out,xh

  ldi zl,low(usart_tx_data)
  ldi zh,high(usart_tx_data)
  add zl,xh
  adc zh,zeroh

usart_udre_send:
  ld xl,z
```

```
  sts USART_DATA,xl
...
```

Die Funktion `usart_udre_isr` schreibt das nächste zu versendende Byte in das Datenregister der seriellen Schnittstelle, räumt auf und kehrt dann zurück — **ohne** das Verschicken des Bytes **abzuwarten**. Die Frage ist also, wer wann den W/R Pin zurück auf Lesen setzt.

Glücklicherweise gibt es dafür einen Interrupt, den *usart transfer complete interrupt*. Den aktivieren wir und bringen der zugehörigen Servicefunktion bei, den W/R Pin auf null zu setzen:

```
; file: drivers/usart-isr-tx.asm
...
.if WANT_RS485 == 1
.set pc_ = pc
.org UTXCaddr
  jmp_ usart_utx_isr
.org pc_

; : txc-isr ( -- )
;   rs485_rw low
; ;
; $1E constant USART_TXCAddr
; ' txc-isr USART_TXCAddr int!

; serial transfer complete interrupt
; clear rs485_rw_pin
usart_utx_isr:
  cbi RS485_RW_PORT,RS485_RW_PIN
  reti
.endif
```

Um den Interrupt zu aktivieren, setzen wir in der Datei `main.asm` ein zusätzliches Bit

```
 .set USART_B_VALUE = ... | (1<<TXCIE0)
```

selbstverständlich eingeklammert in die entsprechenden `.if ... .endif` Zeilen.

Mit diesen Änderungen im Assembler–Quelltext sollte der Kontroller einen RS485–Bus einwandfrei ansprechen.

Anschlussstelle

Will man mit einem RS485–Bus–Teilnehmer vom Rechner aus reden, so braucht es einen geeigneten Konverter, der auf dem Rechner eine serielle Schnittstelle zur Verfügung stellt und sich auf dem Bus mit anständigen Pegeln meldet. Solche Konverter gibt es käuflich zu erwerben oder auch selbst zu löten.

Referenz? Schaltung?

Dann ist man mit seinem bevorzugten Terminal–Programm schon ausreichend ausgestattet, um mit einem Bus–Teilnehmer zu reden.

Funkstille 1

Für einen Kontroller an einem Bus mit mehreren Teilnehmern geziemt es sich nicht, spontan und lautstark die Welt mit

```
amforth 4.2 ATmega32  ok
>
```

zu bgrüßen.

Mein erster Versuch, `ver` aus `applturnkey` zu eliminieren, war natürlich zu kurz gedacht: der Prompt kommt immer noch. Schade. Nun wär's denkbar, die Prompt–Worte entsprechend umzubiegen. Stellt sich aber raus, dass die nicht *deferred* sind. Die drei Prompt–Worte zu vektorisieren ist natürlich möglich. Allerdings muss man sich dann die Adressen der ursprünglichen Worte merken, das im `user`–Bereich ablegen und immer schön buchführen. Die Änderungen betreffen mehrere Stellen im `amforth` Quelltext. Es geht aber auch einfacher.

Der Prompt muss ja schließlich auch ausgegeben werden. Das bewerkstelligt am Ende die Funktion `emit`. Die ist schon *deferred* definiert und kann einfach umgebogen werden:

```
: -emit   ['] drop is emit ;
: +emit   ['] tx is emit ;
```

Das ist etwas simplistisch gedacht, denn `emit` könnte ja auch auf etwas anderes als `tx` zeigen. Will man diesen Fall korrekt behandeln, dann etwa so:

```
variable tmpemit
: -emit ['] emit defer@ tmpemit !
        ['] drop is emit ;
: +emit tmpemit @ is emit ;
```

Ich habe mich hier für den simplen Ansatz entschieden. Um `-emit` schon in `applturnkey` zur Verfügung zu haben, habe ich die beiden Worte mit Hilfe von Michael Kalus' Prorgamm `g4.fs` in Assembler umgewandelt und etwas von Hand editiert. Die Datei wird dann via `dict_appl.inc` geladen.

```
; words/emit-on-off.asm

; : +emit   ['] tx is emit ;

VE_EMIT_ON:
    .dw $FF05
    .db "+emit",0
    .dw VE_HEAD
    .set VE_HEAD = VE_EMIT_ON
XT_EMIT_ON:
    .dw DO_COLON
PFA_EMIT_ON:
    .dw XT_DOLITERAL
    .dw XT_TX
    .dw XT_DOLITERAL
    .dw XT_EMIT
    .dw XT_DEFERSTORE
    .dw XT_EXIT

; : -emit   ['] drop is emit ;

VE_EMIT_OFF:
    .dw $FF05
    .db "-emit",0
    .dw VE_HEAD
    .set VE_HEAD = VE_EMIT_OFF
XT_EMIT_OFF:
    .dw DO_COLON
PFA_EMIT_OFF:
    .dw XT_DOLITERAL
    .dw XT_DROP
    .dw XT_DOLITERAL
    .dw XT_EMIT
    .dw XT_DEFERSTORE
    .dw XT_EXIT
```

Dann lässt sich `applturnkey` entsprechend ändern:

```
; words/applturnkey.asm
; ( -- ) System
; R( -- )
; application specific turnkey action
VE_APPLTURNKEY:
    .dw $ff0b
    .db "applturnkey",0
    .dw VE_HEAD
    .set VE_HEAD = VE_APPLTURNKEY
XT_APPLTURNKEY:
    .dw DO_COLON
PFA_APPLTURNKEY:
    .dw XT_INITUSER     ; init-user
    .dw XT_USART        ; +usart
    .dw XT_INTON        ; +int
.if WANT_RS485 == 1
    .dw XT_RS485_INIT   ; rs485.init
    .dw XT_EMIT_OFF     ; -emit
.endif
    .dw XT_VER          ; ver
    .dw XT_EXIT

; : init-turnkey
;   init-user
;   +usart +int
;   rs485.init -emit
; ;
```

Jetzt ist Funkstille, wenn man den Kontroller einschaltet. Allerdings muss man sich daran auch gewöhnen, und erst mal blind `+emit` eingeben, bevor man beschließt, dass mehr Funkstille herrscht als gewünscht.

Funkstille 2: mpc

Verbindet man den Rechner mit zwei oder mehr Kontrollern an einem RS485–Bus, dann möchte man auch gerne mit denen reden. Aber bitte einzeln und nicht mit allen gleichzeitig. Der bisherige Stand erzeugt hemmungslose echo–Schleifen, die nicht selten darin enden, dass mindestens einer der Kontroller richtig beleidigt ist und auch nach einem Reset gar nicht mehr reden will.

Die Atmel–Kontroller sind für diesen Fall vorbereitet. Es gibt eine Einrichtung mit dem Namen *multi processor communication (mpc)*. Schaltet man mpc ein, dann werden alle vom USART empfangenen Bytes verworfen und keine Interrupts ausgelöst, bis ein Byte empfangen wird, bei dem das höchste Bit gesetzt ist. Dieses Byte wird als Adressbyte behandelt. Die Interrupt–Service–Routine inspiziert dieses Byte und vergleicht es mit der gespeicherten Stationsadresse. Sind sie gleich, wird der mpc–Modus beendet und die Schnittstelle funktioniert ganz normal.

An einem Bus mit mehreren Kontrollern (Stationen) werten alle Kontroller das Adressbyte aus. Aber nur der angesprochene beendet den mpc–Modus. Am Ende der Kommunikation muss er den mpc–Modus wieder aktivieren. Solange man nur ASCII Zeichen überträgt und keine binären Daten, muss man keine weiteren Vorkehrungen treffen. Alle Messwerte, die die Stationen aufgesammelt haben, werden ganz normal als ASCII–Zeichenketten ausgegeben. Diese Zeichen haben das höchste Bit nicht gesetzt und animieren daher auch die Stationen nicht, dieses als Busadresse zu behandeln. Mit Umlauten oder binären Daten geht das so nicht. Außerdem hat das den Vorteil, dass man auf dem Bus einfach so mitlesen kann.

Um den mpc–Modus zu nutzen, sind folgende Aufgaben zu lösen:

- Die Unterstützung für den mpc–Modus soll beim Assemblieren wahlweise eingebunden werden können (WANT_MPC)
- Die Stationsadresse muss gesetzt werden (mpc_ID)
- mpc–Modus einschalten (`+mpc`: *7 Datenbits, kein Paritätsbit, 2 Stopbits*)
- mpc–Modus beenden (`-mpc`: *8 Datenbits, kein Paritätsbit, 1 Stopbit*)
- Die Assemblerfunktion `usart_rx_isr` muss prüfen, ob der mpc–Modus an ist, und wenn ja, ob diese Station gemeint ist.
- Die Stationsadresse muss persistent gespeichert und beim Starten von `amforth` wieder gesetzt werden
- Im `amforth`–Programm braucht es ein paar Worte zur Bedienung, z.B `~id`, `~call`, `~end`

mpc–Modus einbauen

main.asm: WANT_MPC

Wie vorher wird eine Assembler–Variable definiert, die zum Aktivieren des entsprechenden Quelltexts benutzt wird:

```
; file: main.asm
.set WANT_MPC = 1 ; mpc mode (7N2, 8N1)
```

Stationsadresse in RAM

Die Stationsadresse wird zum Vergleich mit einer empfangenen Adresse gebraucht. Sie soll gleichzeitig als normale `amforth`–Variable benutzbar sein. Dafür wird die Adresse der Speicherstelle gleichzeitig in der Assembler–Variablen `var_mpcid` abgelegt.

```
; file: drivers/usart-isr-rx.asm
...
.if WANT_MPC == 1
; ( -- addr ) variable mpc_ID, RS485
; module address
VE_MPCID:
    .dw $ff06
    .db "mpc_ID"
    .dw VE_HEAD
    .set VE_HEAD = VE_MPCID
XT_MPCID:
    .dw PFA_DOVARIABLE
PFA_MPCID:
    .dw here
    .set var_mpcid = here
    .set here = here + CELLSIZE
.endif
```

mpc–Modus einschalten

Um den mpc–Modus einzuschalten, muss das Bit `MPCM` im Register `UCSRA` gesetzt, sowie die serielle Schnittstelle

auf 7 Datenbits, kein Paritätsbit, 2 Stopbits konfiguriert werden (Register UCSRC). Dabei ist zu beachten, dass beim Schreiben von UCSRC das höchste Bit URSEL gesetzt werden muss, weil sonst das Register UBRRH beschrieben wird. An dieser Stelle habe ich länglich gegrübelt, was dem aufmerksamen Leser erspart werden kann.

```
; file: ewlib/mpc.fs
: +mpc7 (  --  )
    txc             \ wait for tx complete
    UCSRA c@ $01 or UCSRA c!    \ set MPCM
    $8C UCSRC c!                \ 7N2
;
```

mpc–Modus beenden

Entsprechend wird zum Beenden des mpc–Modus das Bit MPCM gelöscht und die serielle Schnittstelle auf 8 Datenbits, kein Paritätsbit, 1 Stopbit gesetzt.

```
; file: ewlib/mpc.fs
: -mpc7 (  --  )
    UCSRA c@ $FE and UCSRA c! \ clear MPCM
    $86 UCSRC c!              \ 8N1
;
```

Empfangenes Adressbyte auswerten

Zunächst wird `variable mpc_ID` in Assembler definiert.

```
: file: drivers/usart-isr-rx.asm
...
.if WANT_MPC == 1
; ( -- addr ) variable mpc_ID, RS485
; module address
VE_MPCID:
    .dw $ff06
    .db "mpc_ID"
    .dw VE_HEAD
    .set VE_HEAD = VE_MPCID
XT_MPCID:
    .dw PFA_DOVARIABLE
PFA_MPCID:
    .dw here
    .set var_mpcid = here
    .set here = here + CELLSIZE
.endif
```

Dann wird die Interrupt–Service–Routine für den Empfang von Daten auf der seriellen Schnittstelle aufgebessert.

```
; file: drivers/usart-isr-rx.asm
...
. if WANT_MPC == 0          ; original version
...
.else                       ; MPC version
usart_rx_isr:
  push xl                   ; save registers
  in xl, SREG
  push xl
  push xh
  push zl
  push zh
  push temp0

  in_ xh, UCSRA
  mov temp0, xh             ; temp0 = UCSRA
  andi xh, (1<<FE) | (1<<DOR) | (1<<PE)
                            ; set zero flag
  lds xh, USART_DATA        ; xh = UDR
  brne usart_rx_isr_finish  ; test Z flag,
                            ; finish on error
  lds xl,usart_rx_in        ; xl = i_in
  ldi zl, low(usart_rx_data) ; Z = &buf[0]
  ldi zh, high(usart_rx_data);
  add zl, xl                ; Z += i_in
  adc zh, zeroh             ; .

  sbrc temp0, MPCM          ; if MPCM == 0
  rjmp usart_rx_isr_mpcmode ; {
                            ; NORMAL mode
  st Z, xh                  ; . buf[i_in]=xh

  inc xl                    ; . xl += 1
  andi xl,usart_rx_mask     ; . xl %= siz
  sts usart_rx_in, xl       ; . i_in = xl

  rjmp usart_rx_isr_finish  ; } else {

usart_rx_isr_mpcmode:       ; MPC mode

  ldi zl, low(var_mpcid)    ; . Z=&var_mpcid
  ldi zh, high(var_mpcid)   ; . .
  ld  xl, Z                 ; . xl=var_mpcid
  cp  xl, xh                ; . xl == xh?
  brne usart_rx_isr_finish  ; . if ( yes )
  andi temp0, (~(1<<MPCM))  ; . { called!
  out_ UCSRA, temp0         ; . . clear MPCM
  ldi  temp0, (USART_C_VALUE); . .
  out_ UCSRC, temp0         ; . . 8N1

usart_rx_isr_finish:        ; } _finish

  pop temp0                 ; restore
  pop zh                    ; registers
  pop zl
  pop xh
  pop xl
  out SREG, xl
  pop xl

  reti
.endif
```

Stationsadresse in EEPROM

Die Stationsadresse soll persistent aufbewahrt werden und einen Stromausfall überstehen. Der amforth–Datentyp `value` stellt eine Konstante zur Verfügung, deren Inhalt nicht im Flash–Speicher, sondern im EEPROM aufbewahrt wird. Jetzt soll aber gleichzeitig dieser Wert schon beim Assemblieren von amforth angelegt werden, wenngleich mit einer default–Adresse ($7F). Das hat sich als etwas umständlich erwiesen. Ich musste diesen Eintrag tatsächlich in der Datei amforth.asm vornehmen, also quasi tief in den Eingeweiden. Vielleicht findet sich ein besserer Weg, das zu tun. Dafür müsste wahrscheinlich die Sprungadresse `edp:` in eine Assembler–Variable (`.set edp = ...`) umgewandelt werden.

```
; file: amforth.asm
...
.if WANT_MPC
```

```
EE_STATIONID:
       .dw $007F
.endif
; 1st free address in EEPROM.
edp:
.cseg
```

```
; file: words/mpc.asm
VE_STATIONID:
       .dw $ff09
       .db "stationID",0
       .dw VE_HEAD
       .set VE_HEAD = VE_STATIONID
XT_STATIONID:
       .dw PFA_DOVALUE
PFA_STATIONID:
       .dw EE_STATIONID
```

Die Stationsadresse ist jetzt als `value` verfügbar. In der Datei, die das `amforth`–Programm für eine bestimmte Station enthält, steht daher beispielsweise `$60 to stationID`. Damit wird die default–Adresse überschrieben. Wenn das geschehen ist, dann wird bei nachfolgenden Starts diese Stations–Adresse in die Variable `mpc_ID` übertragen:

```
; file: words/applturnkey.asm
...
.if WANT_MPC
    .dw XT_STATIONID
    .dw XT_MPCID
    .dw XT_STORE
.endif
```

Verwendung

Für die interaktive Benutzung des mpc–Modus definiert man sich noch ein paar Worte:

- `~id` legt die Stationsadresse auf den Stack
- `~end` beendet die Kommunikation und schaltet den mpc–Modus wieder ein
- `~call` sendet das Adressbyte einer anderen Station mit zusätzlich gesetztem höchstwertigem Bit.

```
include ewlib/mpc.fs
: ~end  -emit +mpc7 ;
: ~id   mpc_ID c@ ;
: ~call mpc_call ;
```

```
; file: ewlib/mpc.fs
\ send address for other station
: mpc_call ( c -- ) \ ID
  dup mpc_ID @ = if
    drop
  else
    $0 tx             \ delay
    $80 or tx         \ set 7.bit+ID, for slave
    -emit             \ drop any output!
    +mpc7             \ modul off, wait for ID
  then
;
```

Einer kleinen Unterhaltung mit zwei Stationen steht also jetzt nichts mehr im Weg:

```
> ver
amforth 4.2 ATmega32 ok
> ~id hex .
60  ok
> $61 ~call
unsichtbar: +emit
 ok
> ~id hex .
61  ok
>
```

Bei meinem ersten Versuch mit RS485 und mpc hatte ich das Versenden des Prompts nicht unterbunden. Ich hatte nur festgestellt, dass auf dem Bus nach einem `~end` ein paar krause Zeichen erscheinen (nicht druckbar). Inzwischen weiß ich, dass das die Zeichen des ok– und des ready–Prompts sind, wenn man sie durch *7 Datenbits, kein Paritätsbit, 2 Stopbits* verstümmelt, d. h., das höchste Bit setzt. Das hat im Dateneinsammler zu Mehraufwand geführt, um ggf. unbrauchbare Zeichen zu lesen und zu filtern. Seit `-emit` ist Ruhe.

Die Stationsadresse hält sieben signifikante Bits, man kann also Adressen von Null bis 127 ($7F) vergeben.

mpc: prompt

Nach einer Weile hat es mich gestört, dass ich immer erst `~id .` eingeben muss, um zu sehen, auf welcher Station ich gerade bin. Und da ich sowieso schon einen Blick in die Prompts geworfen hatte, habe ich beschlossen, den ready–Prompt zu ändern. Ich wollte die Stationsnummer in hexadezimal sehen, etwa so: `~61>`. Also erweitert man das Wort `p_rd` sinngemäß um diese Zeilen:

```
...
[char] ~ emit
base @ $10 base !
mpc_ID @ 2 u0.r
base !
...
```

```
; file: words/prompts.asm
...

; : p_rd
;   [char] ~ emit
;   base @ $10 base !
;   mpc_ID @ 2 u0.r
;   base !
;   [char] > emit space
; ;

; send the READY prompt to the command line
;VE_PROMPTRDY:
;    .dw $ff04
;    .db "p_rd"
;    .dw VE_HEAD
;    .set VE_HEAD = VE_PROMPTRDY
XT_PROMPTRDY:
    .dw DO_COLON
PFA_PROMPTRDY:
    .dw XT_CR
.if WANT_MPC == 1
    .dw XT_DOLITERAL
    .dw $7e             ; '~'
    .dw XT_EMIT         ; emit
    .dw XT_BASE         ; base @ $10 base !
    .dw XT_FETCH
    .dw XT_DOLITERAL
    .dw $10
```

```
    .dw XT_BASE
    .dw XT_STORE
    .dw XT_MPCID        ; mpc_ID @ 2 u0.r
    .dw XT_FETCH
    .dw XT_DOLITERAL
    .dw $2
    .dw XT_UZERODOTR
    .dw XT_BASE         ; base !
    .dw XT_STORE
.endif
    .dw XT_SLITERAL
    .dw 2
    .db "> "
    .dw XT_ITYPE
    .dw XT_EXIT
```

Die Unterhaltung mit den Stationen wird jetzt *intuitiver*:

```
~60> ver
amforth 4.2 ATmega32 ok
~60> ~id hex .
60  ok
~60> $62 ~call
unsichtbar: +emit
 ok
~62> $61 ~call
unsichtbar: +emit
 ok
~61>
```

So ist das Ganze recht nett geworden, jedenfalls nach meinen Maßstäben für nett.

Ausblick

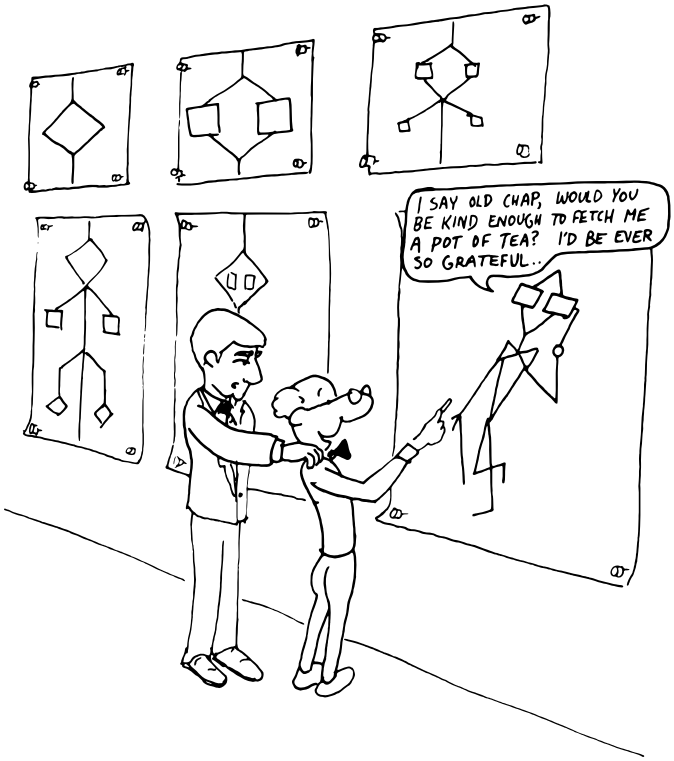
Der Artikel ist mehr eine Übung in Assembler als in Forth geworden. Aber ich habe dazugelernt.

Die eigentliche Aufgabe der Stationen (Messdatenerfassung) ist hier nicht weiter gezeigt. Ich benutze den Multitasker und baue zwei Tasks. Der eine bedient den prompt an der seriellen Schnittstelle. Der andere zählt die Zeit, sammelt die Sensordaten ein und bereitet sie auf, bis der Dateneinsammler sie abholt. Der Dateneinsammler schickt den Befehl `~data`, und der Kontroller versendet daraufhin seine gesammelten Erkenntnisse — alles streng als Text, etwa so:

```
> ~data
__Q 41:0001 1:12,+25.50,+25.54,+25.56 C-- ok
```

Referenzen

1. `amforth`–Seite: `amforth.sourceforge.net/`
2. Luboš Pěknýs Seite: `http://www.forth.cz`
3. Wikipedia (RS485) `de.wikipedia.org/wiki/RS485`
4. `www.mikrocontroller.net/articles/RS-485`
5. Michael Kalus: g4.fs `www.forth-ev.de/repos/g4/g4.fs`
6. atmega32–Datenblatt (doc2503.pdf) `www.atmel.com`



Tobias, ich denke, Du hast die schrittweise Verfeinerung dieses Moduls weit genug getrieben.

Quelle: Thinking Forth

Listings

Die Listings des kompletten Projekts sind etwa 700 Zeilen lang, m. E. viel zu lang für die Druckausgabe. Die Dateien stehen unter `www.forth-ev.de` zum Herunterladen zur Verfügung.

Lediglich die Datei `main.asm`, eine geänderte Kopie der Datei `amforth/releases/4.2/appl/template/template.asm`, sei hier gezeigt, weil die ältere Generation der atmega–Kontroller ein paar mehr Informationen im Assemblerquelltext verlangt.

```
1   ; 2010-12-30  main.asm  EW
2   ;
3   ; this file is derived from
4   ; amforth/releases/4.2/appl/template/template.asm
5   ;
6   ; The order of the entries (esp the include order) must not be
7   ; changed since it is very important that the settings are in the
8   ; right order
9   ;
10  ; note: .set is like a variable, .equ is like a constant
11  ;
12  ; first is to include the macros from the amforth directory
13  .include "macros.asm"
14
15  ; config settings
16  .set WANT_ISR_RX = 1 ; interrupt driven receive
17  .set WANT_ISR_TX = 1 ; interrupt driven send
18  .set WANT_RS485  = 1 ; rs485 half duplex connection
19  .set WANT_MPC    = 1 ; multi-processor-communication mode (7N2, 8N1 modes)
20
21  ; include the amforth device definition file. These
22  ; files include the *def.inc from atmel internally.
23  .include "device.asm"
24
25  ; amforth needs two essential parameters
26  ; cpu clock in hertz, 1MHz is factory default
27  .equ F_CPU = 11059200
28
29  ; initial baud rate of terminal
30  .equ BAUD = 115200
31  ; additional .equs for "old fashioned" mcu with usart, not usart0
32  .equ TXEN0  = TXEN
33  .equ RXEN0  = RXEN
34  .equ RXCIE0 = RXCIE
35  .equ UCSZ00 = UCSZ0
36  .equ TXCIE0 = TXCIE
37
38  .set USART_B_VALUE = (1<<TXEN0) | (1<<RXEN0)
39
40  .if WANT_ISR_RX == 1
41    .if WANT_RS485 == 1
42      .set USART_B_VALUE = (1<<TXEN0) | (1<<RXEN0)| (1<<RXCIE0) | (1<<TXCIE0)
43    .else
44      .set USART_B_VALUE = (1<<TXEN0) | (1<<RXEN0)| (1<<RXCIE0)
45    .endif
46  .else
47    .set USART_B_VALUE = (1<<TXEN0) | (1<<RXEN0)
48  .endif
49  ; 8N1 is commonly used
50  .equ USART_C_VALUE = (1<<URSEL) | (3<<UCSZ00)
51
52  ; .include "drivers/usart_0.asm"
53  .include "drivers/usart.asm"
54
55  ; rs485 transceiver on tx, rx, D.7==w/r, low==read
56  .if  WANT_RS485 == 1
57    .set RS485_RW_PORT = PORTD    ; io location, not mem mapped
58    .set RS485_RW_PIN  = 7
```

```
59  .endif
60
61  .equ TIBSIZE  = $64    ; ANS94 needs at least 80 characters per line
62  .equ APPUSERSIZE = 10  ; size of application specific user area in bytes
63
64  ; addresses of various data segments
65  .set here = ramstart           ; initial HERE at compile time, grows upward
66  .set rstackstart = RAMEND      ; start address of return stack, grows downward
67  .set stackstart  = RAMEND - 80 ; start address of data stack, grows downward
68  .equ amforth_interpreter = max_dict_addr ; the same value as NRWW_START_ADDR
69  ; change only if you know what to you do
70  .equ NUMWORDLISTS = 8 ; number of word lists in the searh order, at least 8
71
72  .equ want_fun = 0 ; in case of an error out print an additional line with
73                    ; a caret indicating the error position
74
75  ; include the whole source tree.
76  .include "amforth.asm"
77
78  ; fin
```

Schaltung

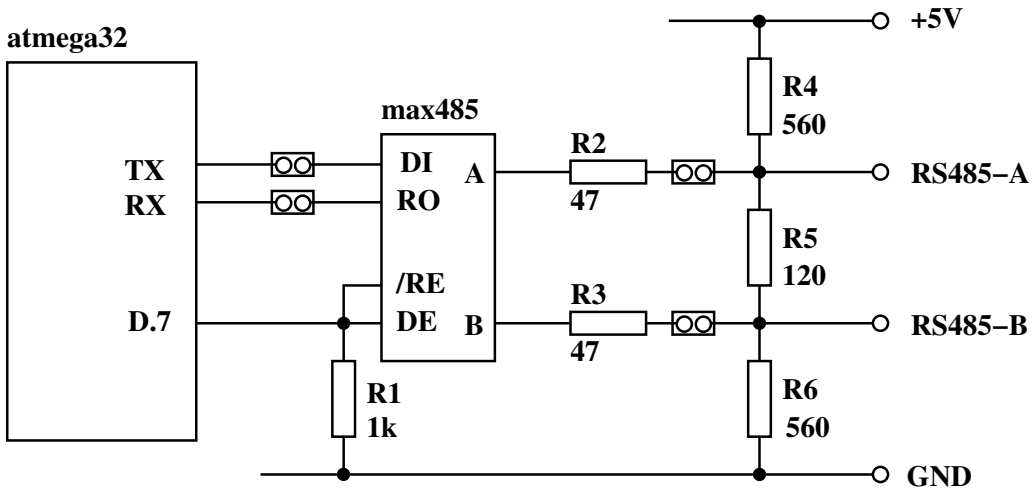


Abbildung 1: Anschluss des RS485 Transceivers an den Kontroller. R1 ist ein pull–down Widerstand, so dass diese Leitung auch dann low ist, wenn der D.7 Pin noch auf Eingang steht. Die Steckbrücken sind manchmal praktisch, um den Transceiver vorübergehend vom Bus und/oder Kontroller abzuhängen. Die Widerstände R4, R5 und R6 werden nur unter bestimmten Umständen benötigt (Definition der Pegel und Busabschluss).

Kleine Stack–Philosophie

Willi Stricker

In der Programmiersprache FORTH spielt der Stack eine dominierende Rolle, der Gebrauch des Stacks ist gewissermaßen das Markenzeichen von FORTH. Es ist also sinnvoll, einige Anmerkungen zum Prinzip des Stacks zu machen.

Der Stack (deutsch: Stapel) ist ein LIFO–Speicher (last in, first out), d. h. das zuletzt auf den Stack gelegte Element ist das erste, das ausgelesen wird, wie bei Paketen, die aufgestapelt sind. Er hat normalerweise zwei unterschiedliche Funktionen:

1. Speicherung der Rücksprung–Adressen (Return Addresses) bei Unterprogramm–Aufrufen unter Benutzung von Call– und Return–Befehlen,
2. Hilfs–Speicher zur Zwischen–Ablage von Daten unter Benutzung von PUSH– und POP–Befehlen.

Üblicherweise wird der Stack im RAM am obersten Ende an der höchsten Adresse gestartet und besetzt bei zunehmender Belegung den Speicher zu niedrigeren Adressen hin. Er *läuft abwärts*! Die Speicherplätze für Variablen werden dagegen am unteren Ende aufsteigend platziert. Auf diese Weise wird erreicht, dass der gesamte RAM–Bereich oberhalb der Variablen für den Stack zur Verfügung steht. Das ist sinnvoll, da zwar die Variablen durch das Programm festgelegt sind, der Stack aber prinzipiell als *unendlich groß* angenommen wird. Folglich kann der Stack je nach Programm–Ablauf *überlaufen*, was in der Regel einen Programmabsturz zur Folge hat.

Werden jedoch, wie bei FORTH, mehrere Stacks benötigt oder besitzt der Stack einen eigenen abgeschlossenen Speicher, so ist diese Überlegung nicht mehr zwingend, er kann also auch am unteren Ende starten und *aufwärts laufen*. Anmerkung: Ich habe einmal eine FORTH–Implementierung gesehen, bei der die beiden Stacks gegeneinander liefen.

Der Stack benötigt einen Stackpointer (Stapelzeiger), der die jeweilige Adresse der aktuellen Stackposition enthält. Mit ihm wird die Startadresse als Beginn des Stacks und die *Laufrichtung* definiert. Dabei gibt es verschiedene Möglichkeiten des Aufbaus je nach Richtung und Startadresse:

Stackrichtung im RAM:

- aufwärts (aufsteigende RAM–Adressen); die Stackelemente werden nacheinander in aufsteigende Adressen abgelegt, das zuletzt abgelegte Element liegt auf der höchsten Adresse,
- abwärts (absteigende RAM–Adressen).

Initialisierung:

- Stackpointer *zeigt* auf das oberste Stackelement,
- Stackpointer *zeigt* auf den nächsten freien Stackplatz.

Daraus ergeben sich 4 Fälle:

1. aufwärts, Stackpointer zeigt auf oberstes Stackelement,
2. aufwärts, Stackpointer zeigt auf freien Stackplatz,
3. abwärts, Stackpointer zeigt auf oberstes Stackelement,
4. abwärts, Stackpointer zeigt auf freien Stackplatz.

Die Fälle 1. und 2. sind unüblich, benutzt nur bei Prozessoren, die einen festgelegten Stack besitzen, z. B. Intel 8051 und deren Derivate. Der Fall 4. wird z. B. bei den 8–Bit–Prozessoren von Freescale (Motorola) eingesetzt, der Fall 3. wird bei den heute üblichen Mikroprozessoren meist benutzt, er ist so etwas wie ein Standard.

Auf die *normale* Arbeit mit dem Stack unter Benutzung der oben genannten Befehle haben die unterschiedlichen Konfigurationen keinen Einfluss. Lediglich die Initialisierung des Stackpointers muss entsprechend angepasst werden.

Stackpointer, Bytezahl

Das RAM, in dem sich der Stack befindet, ist üblicherweise in Bytes organisiert. Damit hat der Stackpointer abhängig von der internen Organisation ausschließlich gerade Adressen bei 16–Bit–Organisation (Adress–Bit 0 ist null) oder durch 4 dividierbare (ohne Rest) Adressen bei 32–Bit–Organisation (Adress–Bits 0 und 1 sind null).

Praxis

Wie erwähnt, ist die Stackorganistion bei Mikroprozessoren meist rückwärts laufend und der Stackpointer zeigt auf das oberste Stack–Element:

Ablauf:

- Push: Stackpointer mit predecrement,
- Pop: Stackpointer mit postincrement.

Initialisierung:

- Stackpointer zeigt auf das „nicht vorhandene“ Element unterhalb des Stackbereichs.

Stacks beim STRIP–FORTH–Prozessor

Der STRIP hat, wie bei FORTH üblich, zwei Stacks:

1. **Return–Stack (RP)** zur Speicherung der Rücksprung–Adressen,
2. **Parameter–Stack (SP)** zur Speicherung von allgemeinen Daten und Adressen (Parametern).

Die Stacks sind jeweils in einem separaten RAM hardwaremäßig vom allgemeinen RAM getrennt untergebracht und nicht durch Datentransfer–Befehle zugreifbar. Sie werden folgendermaßen organisiert (abweichend vom üblichen Standard):

- Vorwärts laufend,

- der Stackpointer zeigt auf den nächsten freien Stackplatz.

Die Stackpointer werden mit Null initialisiert (kein Stackelement vorhanden). Daraus folgt der Ablauf:

- Push: Stackpointer mit postincrement,
- Pop: Stackpointer mit predecrement.

Der Inhalt des jeweiligen Stackpointers ist immer identisch mit der Anzahl der abgelegten Elemente, es gilt (unabhängig von der Bytezahl pro Stackelement):

RP@ (→ Anzahl der Stackelemente auf dem Returnstack)

SP@ (→ Anzahl der Stackelemente auf dem Parameterstack)

RP! (vorgegebene Returnstack–Position →)

SP! (vorgegebene Parameterstack–Position →)

Anmerkung: Der Befehl SP@ ist dann identisch mit dem Befehl DEPTH.

Anmerkungen zur Programmierung

Die Definition der STRIP–Befehle RP@, SP@, RP! und SP! sind letztlich unabhängig von der Stack–Konfiguration und der Hardware. Sie könnten damit als allgemein gültige Prozessor– und Bitbreite–unabhängige Definition der Stack–Zugriffe auch für Mikroprozessor–Forth–Befehle folgendermaßen definiert werden:

```
: RP@allg ( - RP ) R0 @ RP@ - U2/ ;
: RP!allg ( RP - ) U2* R0 @ SWAP - RP! ;
: SP@allg ( - SP ) S0 @ SP@ - U2/ ; \ = DEPTH
: SP!allg ( SP - ) U2* S0 @ SWAP - SP! ;
```

Als Programmbeispiel soll ein Befehl definiert werden, der n Elemente vom Stack löscht:

STRIP–FORTH–Prozessor:

```
: NDROP ( n - )
    1+ SP@ + - SP! ;
```

Mikroprozessor–FORTH (16–Bit–System):

```
: NDROP ( n - )
    1+ 2* SP@ + + SP! ;
```

Über Flags in Forth

Fred Behringer

Willi Stricker hat in einer sehr interessanten Arbeit [S1] gezeigt, wie man mit lediglich 26 Primitives (Code–Definitionen) ein Forth–System hochziehen kann, das im Übrigen nur High–Level–Worte enthält. (Eine ähnliche Arbeit zum Thema ist die bekannte Internet–Notiz von Bernd Paysan [BP]. Weitere Arbeiten zum Thema: [RA],[FB].) Willi Stricker benötigt zum Aufbau der Colon–Definition `d+` aus Low– und High–Bestandteil das Carry–Flag. *In Forth gibt es keine Flags.* (Wirklich? Implizit schon!) Willi Stricker führt eigens für dieses Problem `+c` in seinen Basis–Satz von Primitives ein, aus welchem man sich neben dem carry–Flag auch das übliche `+` (per High–Level–drop) herausholen kann. Das carry–Flag ist implizit in `d+` enthalten! Albert Nijhof hat gezeigt [AN], wie carry aus `d+` herausgeholt werden kann. `+c` ist in Forth unüblich und nimmt sich überdies als unverzichtbares Basis–Wort (Primitive) etwas fremd aus.

Ich zeige im gleich folgenden Listing, dass man `+c` (mit allen Strickerschen Eigenschaften) als Colon–Definition fassen kann, wenn man statt seiner `d+` als Primitive aufnimmt.

Ich beschränke mich auf einen PC–Kompatiblen mit einem Prozessor ab 80486.

Aus `+c` als Colon–Definition kann man (siehe Listing) neben carry auch `+` genau so herausholen wie in Willi Strickers Kernel–Vorschlag angegeben. Ansonsten wird `+c` in Willi Strickers Kernel nicht weiter benötigt und es fällt daher leicht einzusehen, dass durch diesen Austausch (`+c` Kernel, `d+` Primitive statt `+c` Primitive, `d+` Kernel) nichts verloren ist. Gewonnen wird: `d+` ist in Forth–Systemen üblich, `+c` nicht.

Quellen

[AN] Nijhof, Albert: E–Mail an [FB], wiedergegeben in Übersetzung. Vierte Dimension 1/2006, S.10.

[BP] Paysan, Bernd: Beitrag 'aus dem Netz', besprochen in [RA].

[FB] Behringer, Fred: Drei Primitives weniger in Willi Strickers Forth–Minimalbasis. Vierte Dimension 4/2009.

[RA] Allwright, Ray: From the Net – Minimal Word Sets. Forthwrite (FIGUK) 95, März 1998.

[S1] Stricker, Willi: Minimaler Basis–Befehlssatz für ein Forth–System. Vierte Dimension 3/2009.

[S2] Stricker, Willi: Von der virtuellen Forth–Maschine zum realen Forth–Prozessor. Vierte Dimension 4/2010.

Listing

```
1
2   hex
3
4   : opsize: 66 c, ;      \ Praefix, das aus ax (16 Bit) eax (32 Bit) macht
5   usw.
6
7   code d+   ( d1 d2 -- d1+d2 )  \ Alles Punktzahlen (doppelt genau)
8            ax pop       \ d2-hi
9            bx pop       \ d2-lo
10           cx pop       \ d1-hi
11           dx pop       \ d1-lo
12           cx push      \ d1-hi
13           dx push      \ d1-lo
14           ax push      \ d2-hi
15           bx push      \ d2-lo
16    opsize: ax pop       \ d2-lo..hi    im  Reg eax   (little endian)
17    opsize: bx pop       \ d1-lo..hi    im  Reg ebx   (little endian)
18    opsize: ax bx add    \ d1+d2-lo..hi im  Reg ebx   (little endian)
19    opsize: bx push      \ d1+d2-lo..hi auf den Stack (vom TOS her nach oben)
20           ax pop       \ d1+d2-lo     von dem Stack
21           bx pop       \ d1+d2-hi     von dem Stack
22           ax push      \ d1+d2-lo     auf den Stack
23           bx push      \ d1+d2-hi     auf den Stack
24           \  Jetzt liegt d1+d2-hi..lo auf dem Stack (vom TOS her nach oben)
25           next end-code
26
27  \ Man haette auch mit (dem ueber opsize: erweiterten) Assembler-Befehl rol
28  \ arbeiten koennen.
29
30  \ Bildung der Summe zweier einfachgenauer Ganzzahlen n1 und n2 mit Anzeige
31  \ des Uebertrags, carry (= 0 oder 1). carry ist der Inhalt des carry-Flags
32  \ NACH der Operation + in +c. Nicht zu verwechseln mit adc (add WITH carry),
33  \ das das carry-Flag VOR Beginn der Operation zum Ergebnis der Operation
34  \ addiert. Das carry-Flag interpretiert n1 und n2 als vorzeichenlos, so wie
35  \ es beim Zusammenstueckeln der Addition zweier doppeltgenauer Zahlen aus
36  \ zwei Additionen aus je zwei einfachgenauen Zahlen benoetigt wird.
37
38  : +c ( n1 n2 -- n1+n2 carry ) >r 0 r> 0 d+ ;
39
40  \ Und um die Sache perfekt zu machen, noch schnell die +-Operation aus +c
41  \ heraus, jetzt als Colon-Definition:
42
43  : +  ( n1 n2 -- n1+n2 ) +c drop ;
44
45  \ Das stimmt mit dem Kernel-Vorschlag von Willi Stricker [S2] ueberein. Man
46  \ pruefe nach, dass alle Ergebnisse richtig herauskommen, und ueberlege sich,
47  \ warum.
48
```

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

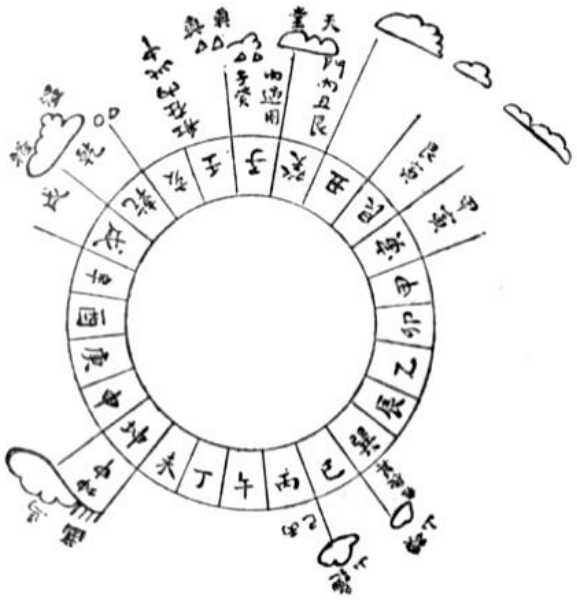
Carsten Strotmann
`microcontrollerverleih@forth-ev.de`
`mcv@forth-ev.de`

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)



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.

Einladung zur

Forth–Tagung 2011
vom 15. bis 17. April 2011

im Bildungshaus Zeppelin
Zeppelinstraße 7, 38640 Goslar

http://www.bildungshaus-zeppelin.de

Geplantes Programm

Donnerstag, 14.04.2011
nachmittags Treffen der Frühankommer
Forth–200x–Standard–Treffen

Freitag, 15.04.2011
nachmittags Beginn der Forth–Tagung
Vorträge und Workshops

Samstag, 16.04.2011
vormittags Vorträge und Workshops
nachmittags Exkursion

Sonntag, 17.04.2011
09:00 Uhr Mitgliederversammlung
nachmittags Ende der Tagung

Anreise siehe: http://www.bildungshaus-zeppelin.de/Anreise.156.0.html

Die Innenstadt von Goslar

Quellen: www.bildungshaus-zeppelin.de, wikimedia.org