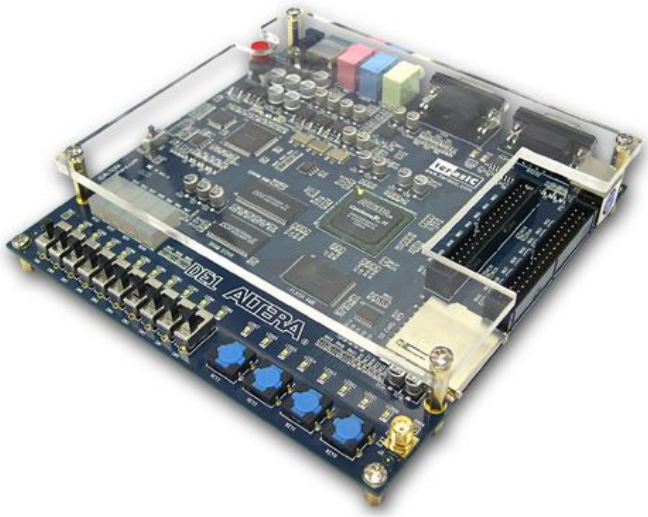
Vierte Dimension

1/2009

Das Forth–Magazin

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

In dieser Ausgabe:



Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische Informatik

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

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

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

LEGO RCX–Verleih

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

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

Inhalt:

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

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

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

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

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

KIMA Echtzeitsysteme GmbH

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

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

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

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

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

Ingenieurbüro
Klaus Kohl–Schöpe

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

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

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 32 01 24
68273 Mannheim
Tel: ++49(0)6239 9201-85, Fax: -86
E–Mail: Secretary@forth-ev.de
Direktorium@forth-ev.de
Bankverbindung: Postbank Hamburg
BLZ 200 100 20
Kto 563 211 208
IBAN: DE60 2001 0020 0563 2112 08
BIC: PBNKDEFF

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

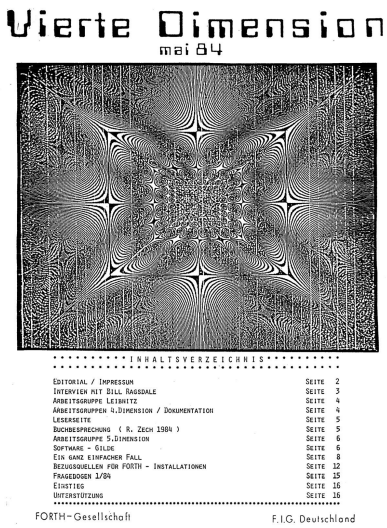
1 Ausgabe / Quartal

Einzelpreis

4,00€ + Porto u. Verpackung

Liebe Leser,

und schon ist wieder ein Jahr vergangen und die erste Ausgabe des 25. Jahrgangs liegt vor Euch. Ja! 25 Jahre Forth-Gesellschaft und 25 Jahre Forth–Magazin. Die Nullnummer der Vierten Dimension aus dem Mai 1984 war eine 16 Seiten DIN–A4 lose Blattsammlung — das Faksimile findet sich auf dem Server der Forth–Gesellschaft unter `http://www.forth-ev.de/filemgmt_data/files/4d1984_0.pdf` — und präsentierte ein ausgedehntes Interview mit Bill Radgsdale, der damals Vorsitzender der FIG/USA war. Ronald Zechs Buch *Die Programmiersprache Forth* (auch mein Einstiegsbuch) wurde besprochen und „Eine Deutsche Version des Einführungskurses *Starting Forth* von Leo Brodie erscheint in Kürze bei einem Münchener Verlag unter dem Namen *Programmieren in Forth*.“


Vierte Dimension
mai 84

INHALTSVERZEICHNIS

FORTH-Gesellschaft F.I.G. Deutschland

Auch über Arbeitsgruppen wird berichtet: Leibniz (ein deutsches Forth), FÜNFTE DIMENSION (Forth der nächsten Generation), Vierte Dimension/Dokumentation (uns gibt's immer noch) und auch die *Software Gilde* findet ihre Erwähnung. Der erste Fachartikel *Ein ganz einfacher Fall* beschreibt die Implementierung des Wortes `CASE?`[1], das für viele Arten von Fallunterscheidungen in Forth eingesetzt werden kann. Bezugsadressen für Forth-Systeme damals populärer Computer (Sharp PC 1500, C64, VC20, Apple II, TI 99/4A) und FIG–Forth–Listings, eine Anzeige von der *Forth–Quelle Angelika Flesch*, ein Fragebogen und ein Bestellformular runden das Heft 0 ab.

Doch zurück zum vorliegenden Heft. Ich hoffe, es ist wieder für jeden etwas Interessantes dabei.

Wir werden zum Lösen einer Programmieraufgabe herausgefordert, lernen über *Logical Block Addressing* und über 2000 Jahre dauernden Zinseszinsen. Wir erfahren, wie man Forth nach Assembler übersetzt, wie man die Stackreihenfolge für double–Zahlen richtig wählt und worüber unsere Forth–Freunde in den Niederlanden diskutieren.

Und wir machen erste Schritte mit dem Altera–DE1–FPGA–Entwicklungsboard, auf dem wir die Forth-CPUs b16 und MicroCore zum Laufen bringen wollen. In den kommenden Ausgaben wollen wir uns schrittweise diesem Ziel nähern.

Viel Spaß beim Lesen!

Ulrich Hoffmann <uho@forth-ev.de>

[1]: `: case? ( n0 n1 - tf / n0 ff ) over = dup IF nip THEN ;`

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

R32C statt R8C

Ich könnte mir vorstellen, dass es in der FG (mindestens) einen Bastler gibt, der (aus purem Spaß am Werkeln) den (kostengünstigen) 32–Bit–MC R32C (mit eingebauter FPU) so aufbereitet, dass Gforth–R8C mit vertretbarem Aufwand übertragen werden kann. Wenn diese Zeilen veröffentlicht sind, wird das März–Heft der Zeitschrift Elektor erschienen sein, das uns über neuere Bemühungen der Elektor–Redaktion auch um den R32C unterrichten wird (siehe Vorschau im Februar–Heft). Ich habe mal ein paar Informationen aus dem Internet zusammengesucht (die allerdings schon etwa ein Jahr alt sind):

Renesas definiert den 32–Bit CISC–Markt neu

Rutronik vertreibt neue 32–Bit Mikrocontroller von Renesas. Die R32C/111 von Renesas sind mit On–Chip–Flash–Speicher ausgestattet und basieren auf dem 32–Bit CPU–Kern R32C/100.

Die Microcontroller–Gruppe besteht aus zwölf Modellen und eignet sich für zahlreiche Anwendungen — von Verbrauchsmessern über industrielle Applikationen bis hin zur Consumer–Elektronik. Beim R32C/100 handelt es sich um einen CISC–Mikrocontroller (Complex Instruction Set Computer). Der Prozessorkern ist abwärtskompatibel zu den existierenden CPU–Cores der Plattform und, wie Rutronik mitteilt, hinsichtlich Verarbeitungsleistung, Busauslastungs–Effizienz und Codeeffizienz verbessert worden.

Die Gruppe R32C/111 kann mit maximal 50 MHz getaktet werden, die Verarbeitungsleistung beträgt 42 DMIPS. Dabei kann der Code direkt vom eingebauten Flash–Speicher ohne Wait ausgeführt werden. Zum Leistungsumfang gehören ein 32–Bit Multiplizierer, eine Gleitkomma–Einheit (Floating–Point Unit–FPU) mit einfacher Genauigkeit und ein verbesserter 32–Bit Barrel Shifter für eine Steigerung der Verarbeitungsleistung. Der 64–Bit breite Bus zum integrierten Flashspeicher soll die Bus–Effizienz optimieren.

Die Produkte der R32C/111–Gruppe werden in kleinen, 14 x 14 mm großen LQFP–100–Gehäusen angeboten. Rutronik liefert erste Muster des R32C/111 bereits aus. Der Preis für einen Baustein im LQFP–100–Gehäuse beginnt in Stückzahlen ab 6,90 Euro.

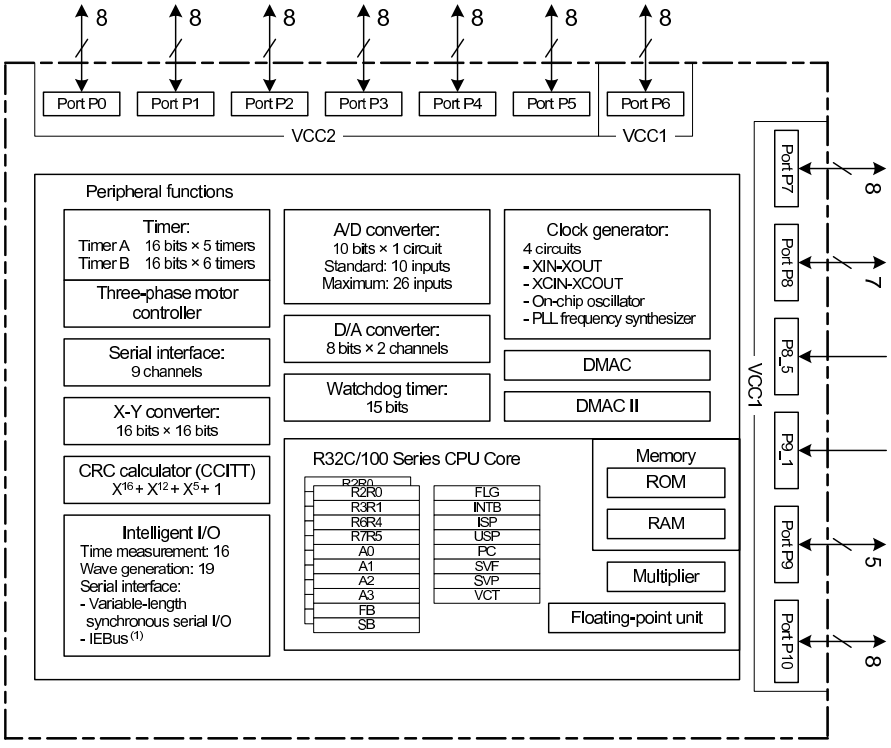
Alexandra Hose,
elektroniknet.de <mailto:ahose@elektroniknet.de>

Weitere Links:

Rutronik
`http://www.elektroniknet.de/`
`?id=anbieterdatenbank&mode=detail&fid=1002232`

Renesas
`http://www.elektroniknet.de/`
`?id=anbieterdatenbank&mode=detail&fid=8867488`

Fred Behringer



Überblick über den Renesas–R32C/111 (Quelle: R32C/111–Datenblatt unter `http://www.renesas.com`)

Gehaltvolles

zusammengestellt und übertragen von *Fred Behringer*

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 71, Januar 2009

SD–kaart besturing (deel 2 van 3) — Willem Ouwerkerk

Das ist der zweite Teil einer dreiteiligen Artikelserie von Willem Ouwerkerk über die Ansteuerung von SD–Karten. Der erste Teil erschien im Vijgeblaadje 69 vom August 2008. Es folgt eine Übersetzung der Einleitung des vorliegenden zweiten Teils:

„Im ersten Teil bin ich bis zur Lese– und Schreib–Routine gekommen. Im vorliegenden Teil setzen wir da noch ein Forth–Blocksystem obenauf. SD–Karten unterhalb von 256 Megabyte sind nicht mehr oder kaum noch zu bekommen und das Adressieren einer ganzen Karte über eine einzige 16–Bit–Zahl geht dann also nicht mehr. Mit einer 16–Bit–Zahl können wir maximal 64 Megabyte adressieren (65536 Blöcke von 1 Kilobyte). Zum Lesen einer Karte von beispielsweise 256 Megabyte könnten wir auf 32–Bit–Zahlen ausweichen. Damit könnten wir bis zu 4 Gigabyte adressieren. Oder aber wir teilen die SD–Karte in mehrere virtuelle Laufwerke (SD–Karten) auf. Ich habe mich in der zu beschreibenden Software für mehrere virtuelle Laufwerke entschieden. Für eine 128–Megabyte–Karte sind 2 Laufwerke nötig, für 256 Megabyte 4 Laufwerke, für 512 Megabyte 8 Laufwerke usw. Neben dem Block–Treiber schließe ich die Software mit einem minimalen Block–Editor ab.“

Soweit der Autor. Weitere Stichworte: Block–System, Software auf elementarer Stufe, Software auf Editor–Stufe, der Forth–Code.

Het Vijgeblaadje (valt dan toch!?) — Mededeling van Willem Ouwerkerk, voorzitter HCC Forth gg

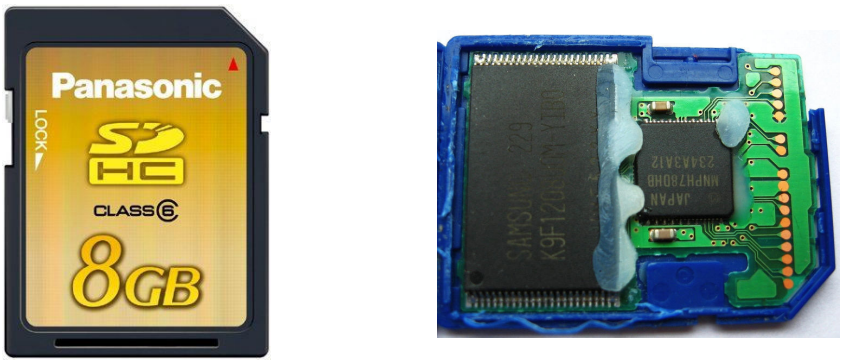
Der Rezensent übersetzt im Folgenden die gesamte Mitteilung des Vorsitzenden der Forth–gebruikersgroep (eine Untergruppierung des (H)obby–(C)omputer–(C)lubs): *Das Vijgeblaadje (läuft dann doch weiter!?)*

Schon vor Jahren nahte einst die Dämmerung in einer Ausgabe, die damals noch von Cees Moerman redigiert wurde. Jetzt scheint es nun doch so weit zu sein: Die Subvention für das Vijgeblaadje hört auf, der HCC trägt nichts mehr zur Herstellung des Blattes bei. Die Wahrscheinlichkeit ist groß, dass das vorliegende Blatt das allerletzte papierene Vijgeblaadje ist.

Jammerschade. Es besteht aber noch eine kleine Chance, den Druck fortzusetzen. Wenn sich nämlich eine genügende Zahl von Teilnehmern meldet, die die papierene Version beibehalten möchte. Für 12,50 Euro pro Jahr können wir das Vijgeblaadje weiterhin anfertigen und versenden.

Wenn ihr euch also zahlreich meldet, können wir diese (papierene) Tradition fortsetzen. Anmeldung über einen kurzen Brief an den Absender (des vorliegenden Blattes, die Forth–gg, Boslaan 1, Bilthoven) oder per E–Mail an: w.ouwerkerk@kader.hcc.nl. Als Angabe genügt „ja, ik wil (ja, ich will)“ nebst Absender — und wir legen dann eine neue Datenliste für bezahlende Mitglieder an. (Rezensent: HCC–Mitglieder, die neben ihren Tribut an die Dachorganisation HCC auch noch einen Obolus an die Fgg für die papierene Ausgabe des Vijgeblaadjes berappen).

Auf geht’s, wir werden doch dieses kleine grüne Blättchen nicht auf dem Komposthaufen enden lassen!



SD–Karten von außen und geöffnet (Quelle: `http://de.wikipedia.org/wiki/SD_Card`)

Cedrics Codierwettstreit — eine Lösung in Forth

Samuel A. Falvo II (Übersetzt und aufbereitet von Michael Kalus)

Der Wettstreit

Neulich fand ich einen Wettstreit im Netz:
`http://beust.com/weblog/archives/000491.html`
Posted by cedric at June 27, 2008 11:48 PM

Cedric schrieb: „Hier ist eine interessante Codieraufgabe: Schreibe einen Zähler, der von 1 bis MAX zählt, aber nur die Zahlen wiedergibt, deren Ziffern sich nicht wiederholen. Als Beispiel diene dieser Teil der Ausgabe:

8, 9, 10, 12 (11 ist ungültig)
98, 102, 103 (99, 100 und 101 sind ungültig)
5432, 5436, 5437 (5433, 5434 und 5435 sind ungültig)

Außerdem:

- Gebe den größten Sprung an; im Beispiel ist es 4: 98 → 102.
- Gebe an, wieviele Zahlen ausgegeben wurden.

Gib beides an für MAX = 10000.

Postet Eure Lösungen im Kommentar oder auf Eurem eigenen Blog. Es sind alle Sprachen und alle Herangehensweisen willkommen. Ich bin gespannt, ob jemand eine Lösung hat, ich habe keine...“

Cedrics Herausforderung wurde angenommen, es trafen viele Lösungen in allen möglichen Sprachen bei ihm ein. Samuel A. Falvo II postete seine Lösung in Forth am June 29, 2008 02:19 AM. Er war so freundlich, auf meine Mail zu antworten, und sehr entgegenkommend, hilfsbereit und gleich einverstanden, einen kleinen Beitrag für die VD daraus zu machen — herzlichen Dank, Samuel.

Der im Blog etwas ramponierte Forth–Code — der Blog entfernte alle *überschüssigen* Blanks — wurde von ihm in seinem Stil restauriert. Wer sein Video gesehen hat, wird schon eine Vorstellung von seinem Programmierstil haben.

Listing

```
1   vocabulary cedrics_challenge
2   cedrics_challenge definitions
3
4   10000 constant largest
5   : rdrop     postpone r> postpone drop ; immediate
6
7
8   ( a simple approximation of integer sets using bit-masks )
9   ( good enough to satisfy the challenge )
10
11  : mask        ( n -- n' )              1 swap lshift ;
12  : union       ( n a -- )               swap mask over @ or swap ! ;
13  : intersects  ( n a -- f )             @ swap mask and ;
14
15
16  ( the proper challenge solution  )
17
18  variable seen-set
19  variable recurs-set
20  variable t
21  variable max-j
22  variable j
23
24  : digits     ( n -- c-addr u )          0 <# #s #> ;
25  : digit      ( c-addr u -- c-addr u n ) over c@ [char] 0 - ;
26  : recurs     ( n -- )                   seen-set intersects recurs-set @ or recurs-set ! ;
27  : seen       ( n -- )                   seen-set union ;
28  : analyze    ( c-addr u -- )            seen-set off recurs-set off begin dup while
29                                          digit dup recurs seen 1 /string repeat 2drop ;
30  : recur      ( c-addr u -- f )          analyze recurs-set @ ;
31  : unique     ( n -- n )                 dup digits recur if 1 j +! rdrop exit then ;
32                                          ( filters numbers for uniqueness )
33  : j0         ( -- )                     1 j ! ;
```

```
34  : initialize ( -- )                  t off j0 max-j off ;
35  : jumps      ( -- )                  j @ max-j @ max max-j ! j0 ;
36  : total      ( -- )                  1 t +! ;
37  : stats      ( -- )                  jumps total ;
38  : number     ( n -- n )              unique dup u. cr stats ;
39  : report     ( -- )                  max-j @ . ." max jump" cr
40                                       t @ . ." total numbers displayed" cr ;
41  : numbers    ( -- )                  1 begin number 1+ dup largest > until drop ;
42  : challenge  ( -- )                  initialize numbers report ;
43
44  \ challenge
45  words
46
47  0 [if]
48  Um das laufen zu lassen, fuege den obigen Code in eine Datei "challenge.fs" ein,
49  und fuehre "gforth challenge.fs" im Terminal nach dem shell prompt aus.
50  Posted by: Samuel A. Falvo II at June 29, 2008 02:19 AM
51  Uebersetzt und tested ok auf gforth; mka 28.12.2008.
52  [then]
53
```

Links

`http://beust.com/weblog/archives/000491.html` Lies auch die *CODING CHALLENGE WRAP-UP*
`http://www.falvotech.com/blog/`
`http://www.falvotech.com/blog/index.php?/archives/372-Over-the-Shoulder-1-Text-Preprocessing-in-Forth.html`

Das *Making of ...*

(Aus der email-Korrespondenz zu Falvos Lösung)

mk: Wie kamst du zu deinem Programmierstil?

sf: Der Stil, in dem ich codiere, entstand so. Mein Bildschirm ist recht groß und hat eine hohe Auflösung. Daher würde eine vertikale Anordnung des Codes sehr viel Platz verschwenden. Horizontal zu schreiben, gefällt mir mehr, denn ich neige dazu, *fließend* Forth zu schreiben. Das liest sich dann wie ein Wörterbuch, und ich strebe an, dass jede Definition mehr oder weniger wie ein englischer Satz gelesen werden kann. Das hilft mir auch, auf Kommentare verzichten zu können. Die meisten meiner Kommentare sind nur noch Prüfvermerke. Meine Bildschirmauflösung, und die relativ kleine Schrift, ermöglichen es mir, die Regel einzuhalten, nur eine Zeile für eine Definition zu brauchen, selten mal eine zweite Zeile. Und aussagekräftige Namen voranzustellen und in der Definition dann leserliche Spalten einzuteilen. Die Namen kommen ganz nach links, Stackbilder mehr in die Mitte und die Definitionen nach rechts. Den Code beim Edieren *hübsch* zu halten macht zwar etwas mehr Mühe, aber es bringt mir auch eine bessere Produktivität, und lohnt daher.

mk: Möchtest du eine Erklärung geben, was `UNIQUE` macht, speziell was `RDROP` da tut? Ich vermute, du fällst damit aus `NUMBER` heraus, ohne dass die Zahl gedruckt wird — so eine Art blitzartiger `EXIT`?

sf: Das stimmt: `UNIQUE` prüft, ob eine Zahl (number) die Kriterien der Einzigartigkeit (uniqueness requirements) aus Cedrics Aufgabe erfüllt. Falls sie das nicht tut, nehme ich den *vorzeitigen Ausgang* (early-exit) aus dem aktuellen Schritt, um in den nächsten Schleifendurchgang zu kommen. Das beseitigt eine Menge an visuellem und mentalem Wirrwar, man vermeidet so, tief verschachtelte Kontrollstrukturen entziffern zu müssen.

Die Art von Logik wird übrigens *structured returns* genannt. Strukturierte Rücksprünge wurden in den 70igern unbeliebt wegen des starken Einflusses, den Dijkstra genommen hat, als er strukturierte Programmiersprachen implementierte. Überdies war es ohne die starre und sehr strikte Typprüfung — die dem C und Pascal offenbar fehlten, was die strukturierte Programmierung für Jahrzehnte auf den Weg brachte — einem Compiler gar nicht möglich, Korrektheit des erzeugten Codes zu verifizieren, wenn dieser keine eindeutigen Eintritts– und Austrittspunkte für alle Programmstrukturen hatte.

Zum Glück kamen funktional höher entwickelte Sprachen auf, wie OCaml und Haskell, und strukturierte Rücksprünge erleben ihr comeback, insofern als die Compiler nun in der Lage sind, statistisch den resultierenden Code zu verifizieren, dank der Eigenschaft referentieller Transparenz und sehr starker Typprüfung.

Wenn man beim Softwareschreiben erst mal mit dem Stil vertraut ist, Fortsetzungen für einen Datenfluss zu übergeben (continuation-passing style of writing software), wird der Returnstack tatsächlich wie ein Pseudo-Parameterstack behandelt, auf dem mögliche Fortsetzungen liegen. Ich suche aus, welche Fortsetzung ausgeführt werden soll, indem der Returnstack entsprechend manipuliert wird.

Programmierer, die mit dem Z–80 Mikroprozessor vertraut sind, erinnern sich möglicherweise an die Instruktion für einen bedingten Rücksprung aus einem Unterprogramm (`RET Z`) und vermissen die in modernen Architekturen. Freigelegt wie der Returnstack im Forth nun mal ist, können wir diese Freiheit ja auch genausogut genießen und wieder strukturierte Rücksprünge machen.

mk: Man sieht es dem `UNIQUE` auf den ersten Blick nicht an, dass es so einen Blitzabgang macht, und wundert sich zunächst einmal, wieso manche Zahlen nicht gedruckt werden.

sf: Mit dem Namen, den ich da ausgesucht hatte, bin ich auch nicht mehr so glücklich inzwischen, aber es ist der beste aus einigen Alternativen, finde ich. Versucht hatte ich auch folgende, aber dann verworfen aus den genannten Gründen:

- `unique:` — Der angehängte Doppelpunkt zeigt konventionell ein Wort an, das neue Worte erzeugt (defining word), passt also nicht.
- `?unique` — Das vorangestellte Fragezeichen suggeriert, dass das Wort optional ausgeführt wird, sagt aber nichts über den Rest der Definition dahinter aus. Dabei möchte ich das Gegenteilige sagen, nämlich dass `UNIQUE` immer ausgeführt wird, nur alles danach ist optional.
- `unique?` — Damit würde ein Prädikat angedeutet, das `TRUE` oder `FALSE` sein könnte. Auch das zeigt die bedingte Ausführung der verbleibenden Definition nicht an.
- `only-if-unique` — Zu viele Worte. Ich suche die Ein–Wort–ein–Konzept–Definition. (Ich schätze diese Wortfülle in anderen Sprachen, besonders in C oder Haskell, da es dort flüssiges Codieren verhindert :)
- `if-unique` — Besser, aber immer noch zwei Worte und Bindestrich.
- `ifUnique` – Hier nähme ich statt Bindestrich einen Großbuchstaben; aber immer noch zwei Worte.
- `only` — Das würde bestens ausdrücken, das der Rest der Definition von irgendeiner Bedingung abhängt. Aber es drückt nicht aus, welche Bedingung es sein könnte.

Auch könnte man sich irgendein anderes Wort ausdenken, das einen besseren Namen für die Funktion abgibt.

Auch wäre ein `CASE` für flüssige Codierung denkbar:

```
: blah   unique? only
         ( Rest der Definition hier ) ;
```

wobei `UNIQUE?` einen boolschen Wert abgeben müsste, und `ONLY` den bedingten Ausgang durchführte. Aber das wäre strukturell identisch zum `IF`/`THEN`:

```
: blah   unique? IF ... THEN ;
```

Das alles erschien mir damals viel zu viel Sehwirrwarr zu sein.

mk: Und wozu setzt du die Masken (mask) in dem Programm ein?

sf: Das Wort `DIGITS` wandelt eine Zahl in ihre Zeichenkette um. Die Zeichenkette erlaubt den Zugriff auf die einzelnen Ziffern der Zahl. `RECUR` wandert zu jeder Ziffer und protokolliert zum einen, welche Ziffer in der Zahl vorgefunden wurde, und zum anderen, welche mehrmals dabei gewesen ist.

Die `SEEN-MASK` erzeugt integer Werte. Da in der dezimalen Zahl nur die Ziffern 0..9 vorkommen, reicht für eine `SEEN-MASK` ein 16-Bit Wert hin. Damit wird angezeigt, welche Ziffern in der Zeichenkette tatsächlich vorhanden waren. `SEEN-MASK` ist vielleicht eine Fehlbenennung; indes, zumindest im Englischen, ist es bei Programmierern durchaus üblich, davon zu sprechen, dass ein Element in einer Sammlung *gesehen* worden ist (item seen in a collection).

Die `RECUR-MASK` ist auch ein integer–Wert, ist aber immer nur eine ausgewählte Teilmenge der `SEEN-MASK`. Hierin wird festgehalten, welche Ziffern mehr als einmal in der Zahl waren. Nimm zum Beispiel die Zahl 343, die `SEEN-MASK` enthält dann `{3,4}` - also Bits 3 und 4 sind gesetzt. Die `RECUR-MASK` hingegen enthält nur `{3}`.

Der logische Operator `AND` berechnet den Durchschnitt zweier Datensätze (set), während `OR` die Vereinigung darstellt. Einen leeren Datensatz zu finden, ist dann so einfach wie die, den mask-Wert auf Null (zero) zu testen.

Ich habe eine bit-mask genommen, um die Datensätze zu erzeugen, weil das die trivialste Form in Forth ist, die es gibt: Integer. Auch in den meisten Pascal– und Modula-2–Systemen implementiert man `SET`–Typen als Bitmasken, und Oberon und Oberon-2 geben ihre `SET`–Typen ebenfalls als Bitmasken ausdrücklich dem Programmierer an die Hand. Ich bin in guter Gesellschaft damit. :)

mk: Vielen Dank für die freundliche Unterstützung.

sf: Kein Problem. Vielleicht interessiert es dich auch, dass ich einen Vortrag auf dem SV–FIG Treffen im Februar (2009) halten werde; falls nicht noch ein besseres Thema auftaucht, wird es ein Durchgang durch meine SEAforth-24–VGA–driver–software und video–clock–application werden.

Bootmanager und FAT–Reparatur: Dritter Fort(h)schritt (LBA)

Fred Behringer

Im vorliegenden Artikel werden einige Forth–Worte entwickelt, mit deren Hilfe man bei Bedarf von der CHS–Adressierung zur LBA–Adressierung übergehen kann.

Ein Wort vorweg über Turbo–Forth und ANSI.SYS

Noch unter DOS 6.2 gab es den Geräte–Treiber ANSI.SYS, der in der CONFIG.SYS per DEVICE=ANSI.SYS eingebunden wurde und der System–Attribute über *Escape–Sequenzen* zu steuern gestattete. Im Laufe der Zeit bin ich immer wieder dazu übergegangen, Turbo–Forth auch unter FreeDOS oder DOS 7.0 (Win 95) oder DOS 8.0 (Win 98) oder in der DOS–Box von Windows ME zu verwenden, und ich habe mich immer wieder über die Meldung *Falsche DOS–Version* geärgert und darüber, dass das ANSI.SYS aus DOS 6.2 nicht akzeptiert wurde. Turbo–Forth läuft dann trotzdem, aber die Escape–Sequenzen werden nicht mehr ausgeführt, sondern am Bildschirm in nicht verständlichen Buchstaben direkt angezeigt. Noch im ersten Teil meiner Artikelserie habe ich für solche Fälle empfohlen, die ATTRIBUTS OFF zu schalten. Das wiederum kann ZF nicht vertragen: ZF kennt keine ATTRIBUTS. Also empfahl ich, ATTRIBUTS OFF im Listing per \ auszukommentieren.

In Turbo–Forth wird von ANSI.SYS hauptsächlich das *fettgedruckte* OK oder die *fettgedruckte* Fehlermeldung hinter ABORT" verwendet. An sich kein großes Problem: Man kann ja ohne Weiteres auch darauf verzichten. Aber schade ist es eben doch. Und auch auf reiner DOS–Ebene (außerhalb von Turbo–Forth) verwende ich ANSI.SYS zur Angabe der Uhrzeit oben rechts auf dem Bildschirm und auf Windows–DOS–Box–Ebene zur individuellen Gestaltung des Prompt–Zusatzes der Art von *Zurück zu Windows über EXIT*.

Ich hätte mir immer wieder mal ein ANSI.COM gewünscht, einen Escape–Sequenzen–Treiber, der an beliebiger Stelle (auch nachträglich) aufgerufen werden kann (beispielsweise in der AUTOEXEC.BAT) und der das Turbo–Forth–System, an das ich mich gewöhnt habe, nicht verstümmelt. Ich wurde nie das Gefühl los, irgendwo mal etwas über ein solches ANSI.COM gelesen zu haben. Das ständige Halbwissen, das man mit sich herumschleppt, und der gelegentliche Drang, in den eigenen bisherigen Arbeiten herumzustöbern, hat mir (auch bei diesem Problem wieder) geholfen. (Schade, dass ich dazu im vorliegenden Fall ein geschlagenes Jahr gebraucht habe.) Also:

ANSI.COM von Michael J. Mefford aus dem PC Magazine des Jahres 1988 heißt die Lösung, entwickelt 1988 (als von Turbo–Forth und ZF noch kaum die Rede war)! Sie funktioniert prima! Und damit sind meine Bemerkungen aus dem ersten Artikel dieser Serie über ATTRIBUTS OFF hinfällig: Man google sich zu den ersten 10 Eintragungen (von 13 Millionen!) durch und beschaffe sich ANSI.COM! (Googlen ist eine schöne Sache, wenn man erst einmal genau weiß, was man eigentlich sucht.)

Ganz nebenbei wird mit ANSI.COM noch ein weiteres Problem gelöst: Wie will man denn beispielsweise unter Windows ME einen DOS–Geräte–Treiber einbauen? In der CONFIG.SYS? Darauf reagiert doch ME gar nicht! Aber ANSI.COM, als ganz gewöhnliches DOS–Programm nachträglich eingebracht (von XP rede ich ja nicht), wird in der DOS–Box von ME sauber ausgeführt. Allerdings muss man ANSI.COM bei jedem Abstecher zur DOS–Box erneut aufrufen!

Von XP habe ich hier nicht gesprochen! Aber die Welt besteht ja nicht nur aus XP. Und die Forth–Welt schon gar nicht.

Eine ASM–Fassung von ANSI.SYS oder ANSI.COM, also den Quelltext, habe ich bei meiner Übung im Googlen auch gefunden! Mal sehen, vielleicht mache ich mir eine Forth–Fassung daraus. Übung macht den Meister!

Eigentliche Einleitung

Im ersten und zweiten Teil dieser Artikelserie habe ich über die Hauptpartitionstabelle (den Master–Boot–Record: MBR) nachgedacht, über die Partitionstabellen der logischen Laufwerke der erweiterten Partition und über die zu den primären Partitionen und den logischen Laufwerken gehörigen Bootsektoren. (Jedes logische Laufwerk hat neben dem *Bootsektor* auch eine eigene *Partitionstabelle*. Das System hangelt sich zum gewünschten logischen Laufwerk von Partitionstabelle zu Partitionstabelle durch.) Ich durfte mich darüber wundern, dass die (bei mir) jenseits der DOS–8GB–Grenze liegende primäre Partition mit Windows ME etwas aus der Reihe fiel: Ihre Adresse (man prüfe das nach) ließ sich mit den von mir angebotenen Forth–Worten (getsect) und (putsect) nicht ermitteln.

Im vorliegenden Artikel gehe ich der Frage nach, warum nicht: Mit der bis zu einer Plattenkapazität von 8 GB funktionierenden CHS–Adressierung der (physikalischen) Sektoren lässt sich das nicht machen. Da müssen *logische* Sektoren her, da muss die LBA–Adressierung angreifen. Natürlich muss bei alldem gesichert sein, dass das BIOS die *erweiterten Interrupt–13h–Funktionen* (41h–48h) überhaupt verarbeiten kann. (Die ganz frühen

PCs können das nicht.) Auf meiner hier zu besprechenden Experimentieranlage ist der erweiterte Int13h vorhanden, schon deshalb, weil ich ja sonst die (bei mir) 5 GB große (primäre) ME–Partition beim Installieren von Windows ME gar nicht so weit nach hinten, nämlich ans Ende der (ansonsten für meine Experimente ausreichend großen) 20–GB–Platte, hätte legen können.

Was mich im vorliegenden Teil 3 meiner Serie über Festplattenpartitionen (über das FAT–Dateisystem und dergleichen) interessiert, ist also die LBA–Adressierung. Das ist insbesondere der *erweiterte Interrupt 13h* mit den Funktionen AH=41h bis AH=48h, und da ganz besonders die beiden Funktionen AH=42h (extended reading) und AH=43h (extended writing). (Feinheiten der Absicherung gegen mögliche Eingabe–, Lese– oder Schreibfehler lasse ich weg. Es geht mir um das Gerüst. Alles Fehlende lässt sich in Forth ja leicht nachträglich einbauen.)

Die LBA–Sektoren werden über die gesamte Festplatte hinweg von 0 bis zum Höchstsektor der Platte hochgezählt (*logische* oder *lineare* Block–Adressierung). Ich will versuchen, den Zusammenhang zwischen der (älteren) CHS–Adressierung (über den Interrupt 13h mit vor allen Dingen den Funktionen AH=2 und AH=3) und der LBA–Adressierung plausibel zu machen. Im vorliegenden Artikel soll dementsprechend über den Einbau der Funktionen 42h und 43h des erweiterten Interrupts 13h in eine Forth–Umgebung (ausprobiert mit Turbo–Forth und ZF) gesprochen werden.

Eine Lanze brechen für Turbo–Forth

Alle Entwicklungsarbeiten (in den VD–Artikeln 1 bis 3 zum vorliegenden Thema) habe ich in Turbo–Forth (TF) durchgeführt und mit ZF überprüft. TF und ZF haben den Vorteil, dass man das System nicht erst um Erlaubnis zu bitten braucht, wenn man den vollen Zugriff (*Administrator–Rechte*) auf die eigene Maschine haben möchte. TF und ZF haben aber den Nachteil, dass sie nur in 16–Bit–Breite arbeiten. Bei der LBA–Adressierung reicht das nicht, so dass mit *doppelter Genauigkeit* gearbeitet werden muss.

Konvertierung von CHS nach LBA (Logical Block Addressing)

Für den Fall, dass es interessieren sollte, von welcher Art von *Festplatte* ich rede: Meine Betrachtungen beziehen sich auf IDE–Platten. Eine vereinfachte Darstellung derer Wirkungsweise läuft wie folgt:

Eine Festplatte besteht aus einer oder mehreren Scheiben. Jede Scheibe hat zwei Seiten. Jeder Scheibenseite (die zu 0/1 der ersten Scheibe, 0/1 der zweiten Scheibe usw. durchnummeriert werden) ist genau ein Schreib/Lese–Kopf (head) zugeordnet. Bei der Festplattenverwaltung zur eindeutigen Adressierung der einzelnen Sektoren (siehe gleich) wird nur von *Kopf* gesprochen, auch wenn in der bildlichen Vorstellung eigentlich *Scheibe* (oder zumindest *Scheibenseite*) gemeint ist. Die Köpfe (also eigentlich die Scheibenseiten) werden vom BIOS in konzentrisch angeordnete *Spuren* (tracks) unterteilt. Jede Spur enthält eine bestimmte Anzahl von *Sektoren* (sectors), eine Zahl, die von Spur zu Spur gleich bleibt. Jeder Sektor enthält eine bestimmte Zahl von Bytes. Üblich sind 512 Bytes pro Sektor. Die Daten werden sektorweise gelesen oder geschrieben.

Die auftretenden Zahlen für Köpfe, Spuren und Sektoren geben nicht unbedingt die wirklichen Verhältnisse wieder. Sie sind bei der Festplatte nur für die Verwaltung wichtig. Die eigentliche *Geometrie* der Festplatte kann (vom Hersteller) durch (interne) Umrechnungs–Vorrichtungen angepasst worden sein. (Man braucht es also nicht wörtlich zu nehmen, wenn man vom BIOS–Setup erfährt, dass die Platte 255 Köpfe hat — und man braucht die Platte auf gar keinen Fall auseinanderzunehmen, um nachzuprüfen, ob das wirklich stimmt ;-)

Man hätte sich auch vorstellen können, dass bei der Zählung der Sektoren erst alle Spuren einer Scheibenseite durchlaufen werden, dann die nächste Seite, dann die nächste Scheibe usw. Das ist nicht der Fall. Es wird eine neue Einteilungs–Einheit eingeführt, die des *Zylinders* (cylinder): Alle (übereinanderliegenden) Spuren derselben (auf die jeweilige Scheibenseite bezogenen) Nummer gehören zu ein und demselben Zylinder. Bei der Zählung (dem Durchnummerieren) der Sektoren werden erst die Sektoren der ersten Spur der ersten Scheibenseite drangenommen, dann die der ersten Spur der zweiten Scheibenseite, dann die der ersten Spur der ersten Seite der zweiten Scheibe und so weiter, bis der erste Zylinder durch ist. Dann dasselbe Zählverfahren in Bezug auf den zweiten Zylinder. Und so weiter. (Wenn man liest, dass ein *Kopf* 63 Sektoren hat, dann ist das entsprechend zu verstehen: Ich komme in meiner Vorstellung gut durch, wenn ich bei Ordinalzahlen (bei Nummern) von *Kopf* rede, bei Kardinalzahlen (bei Anzahlen) aber von *Spur*: Sektor 5 bei Kopf 4, aber 63 Sektoren pro Spur. Warum aber, so frage ich, hätte man nicht gleich von Spuren statt von Köpfen reden können? *Spur 4* nimmt sich bestimmt nicht schlechter aus als *Kopf 4*. Aber über *63 Sektoren pro Kopf* stolpert natürlich jeder. Die *Spuren* müssten eben von oben nach unten gezählt werden, und nicht in konzentrischer Reihenfolge. Und wäre das schlimm? Bei den *Köpfen* kommt ja noch hinzu, dass sie alle gleichzeitig auf die Festplatte zugreifen (können), in der Zählung aber sequentiell eingeordnet werden müssen.

Lange Rede, kurzer Sinn: Es ist letztendlich völlig egal, wie man sich den Aufbau einer Festplatte vorstellt, wichtig sind die Zahlen, die den fest eingebürgerten Begriffen *Zylinder*, *Kopf* und *Sektor* (engl. (C)ylinder, (H)ead, (S)ector) zugeordnet sind. Und wenn man nun mal beim Begriff *Kopf* (head) bleibt, braucht man nicht mehr über den Begriff *Spur* zu philosophieren. Der ist (für die Plattenverwaltung) überflüssig. Dann hat eben beispielsweise *jeder Kopf 63 Sektoren* — und beim Begriff SPT (sector per track) im Wikipedia wäre eben nicht, wie dort geschehen, *Track* gleich *Zylinder* zu setzen, sondern gleich *Kopf*.

(Mir geht es bei der Entwicklung eines Forth–Wortes zur Umrechnung von CHS nach LBA (siehe Listing) um ein Verstehen der zugehörigen Formel — und da ist die Angabe *SPT : Sektoren/Track (Track ^= Zylinder)* im Artikel *Cylinder Head Sector* `http://de.wikipedia.org/wiki/Cylinder_Head_Sector` aus der freien Enzyklopädie Wikipedia nicht sehr hilfreich. Eben nicht SPT = Sektoren pro Zylinder! SPT = Sektoren pro Spur! Und das heißt SPT = Sektoren pro Kopf!)

Zu beachten ist bei der CHS–Zählweise (und das ist sehr wichtig!), dass Zylinder und Kopf bei 0 beginnen, die Sektor–Zählung aber bei 1. Der Master–Boot–Record (MBR) hat also die Adresse C–H–S = 0–0–1 (und nicht etwa 0–0–0). In der LBA–Zählweise dagegen (siehe gleich) hat der MBR tatsächlich die Nummer 0. (Man versuche, mit dem ehrwürdigen DEBUG aus DOS an den MBR heranzukommen, nicht durch Nachbilden der im Vorliegenden entwickelten Assembler–Befehle, sondern über den DEBUG–Load–Befehl `l` (kleines l)!)

Bei den frühen Festplatten stieß man mit der CHS–Adressierung bald an Grenzen: Unter Beibehaltung der Kompatibilität musste etwas Neues eingeführt werden, ohne das *Alte* deshalb gleich aufgeben zu müssen. Dieses Neue heißt LBA (Logical oder Linear Block Addressing). Dabei gibt es nur noch *Sektoren* (keine Köpfe und Zylinder mehr), und die werden die ganze Platte hindurch durchnummeriert, auch über eventuell bestehende Platten–Partitionen hinweg. Zur besseren Unterscheidung werden Sektoren der CHS–Betrachtungsweise auch *physikalische* Sektoren genannt, Sektoren der LBA–Betrachtungsweise *logische* Sektoren.

Die Zählung der logischen Sektoren beginnt bei 0 ! Der MBR hat also die LBA–Nummer 0.

Zum besseren Verständnis möchte ich die folgenden Bezeichnungen einführen:

C	=	Zylindernummer
H	=	Kopfnummer
S	=	Sektornummer (phys. Sektor)
s	=	Sektornummer (log. Sektor)
S/K	=	Sektoren pro Kopf
K/Z	=	Köpfe pro Zylinder

Bei gegebener Zylindernummer C, Kopfnummer H und Nummer S des physikalischen Sektors berechnet sich die Nummer s des zugeordneten logischen Sektors zu:

$$s = ((C * K/Z + H) * S/K) + S - 1$$

Dabei liefert

Zylinder	C	:	$C * K/Z * S/K$	logische Sektoren,
Kopf	H	:	$H * S/K$	logische Sektoren und
Phys. Sektor	$S-1$	:	s	logische Sektoren.

(Bei s zählt 0 mit, bei S nicht.)

Der zugrundegelegte Computer als Beispiel

Das BIOS meiner (hier besprochenen) Anlage gibt beim Aufrufen des BIOS–Setups für die in Frage stehende Festplatte 2491 Zylinder, 255 Köpfe und 63 Sektoren aus. Das weiter unten erwähnte Programm *Partition Table Doctor* liefert als Anzahl logischer (also LBA–) Sektoren den Wert 40017915 (alle Angaben dezimal). Geht man von der üblichen Zahl von 512 Bytes pro Sektor aus, dann entspricht der Wert 40017915 * 512 in etwa einer Festplatten–Kapazität von 20 Gigabyte. Das BIOS–Setup liefert für *size* den Wert 20490 MB . Geht man die Nummerierung der LBA–Sektoren durch, dann erhält man:

Cyl.	Head	Sector	LBA–Sector
0–2490	0–254	1–63	0–40017914
0	0	1	0
0	0	2	1
.	.	..	..
0	0	63	62
0	1	1	63
0	1	2	64
.	.	..	...
0	1	63	125
0	2	1	126
0	2	2	127
.	.	..	...
0	2	63	188
0	3	1	189
.	.	..	.....
0	254	1	16002
0	254	2	16003
.	...	..	.....
0	254	63	16064
1	0	1	16065
1	0	2	16066
.	.	..	.....
1	0	63	16127
.	.	..	.....

Der *erste* logische Sektor hat die Nummer 0 und entspricht dem CHS–Wert 0–0–1. Das geht dann die Reihe der physikalischen Sektoren entlang so durch bis zum logischen Sektor 62, der dem CHS–Wert 0–0–63 entspricht. Dann wird (bei gleichem Zylinder) zum nächsten Kopf übergegangen: Der logische Sektor 63 (also der 64. LBA–Sektor) entspricht dem CHS–Wert 0–1–1 (Der CHS–Sektor beginnt wieder bei 1, und nicht etwa bei 0). Und wieder alle physikalischen Sektoren durch bis zum logischen Sektor 125, der dem CHS–Wert 0–1–63 entspricht. Der logische Sektor 126 entspricht dem CHS–Wert 0–2–1. Und so weiter und so fort. Wenn alle Kopfnummern ausgeschöpft sind, wird zum nächsten Zylinder übergegangen. Der logische Sektor 16127 entspricht dem CHS–Wert 1–0–63, usw.

Ich möchte nicht verschweigen, dass mir bei der Ermittlung und Überprüfung der hier wiedergegebenen Werte das Programm *Partition Table Doctor* der PTDD Group at `http://www.ptdd.com` (selbstgebrannte Live–CD aus einer ganz kleinen ISO–Datei der Begleit–DVD von c't 24/2008) eine große Hilfe war. Allerdings habe ich auch hier wieder für meine FAT–Reparatur–Absichten einige Dinge schmerzlich vermisst. Aber gerade dieser Umstand zeigt mir erneut, eine wie sehr viel noch größere

Hilfe mir ein Forth–Werkzeugkasten voller entsprechender Hilfsmittel zur eigenen Gestaltung wäre. Ich befinde mich auf dem besten Wege, mir einen solchen selbst zusammenzustellen, und habe vor, diesen Gedanken zügig weiterzuverfolgen.

Nebenbei bemerkt: Im VD–Heft 4/2008 wurde in einem der dortigen Artikel über die Suche nach einer geeigneten DOS–Emulation für ein DOS–freies System nachgedacht. Die ebenerwähnte Live–CD baut auf einer abgespeckten FreeDOS–Version auf. FreeDOS kann man sich beispielsweise von `http://www.freedos.org` besorgen. Es liefert dem PC eine vollwertige, der MS–DOS–Version 6.2 mehr als ebenbürtige DOS–Umgebung mit `a:` als Laufwerk. Ein eventuell schon vorhandenes reales (Disketten–)Laufwerk `a:` wird dann nach `b:` verbannt.

Noch nebenbeier bemerkt: Traue nie einem Programm, das du nicht selbst vermurkst hast! Das Laufwerk b: (5,25") lasse ich normalerweise als im BIOS abgemeldet mitlaufen. Es lag auf der Hand, dass ich mal schnell wissen wollte, was passiert, wenn ich b: im BIOS wieder einschalte. Wird dann `a:` nach `b:` und daraufhin `b:` nach — ja wohin eigentlich — verschoben? Nichts passiert! Die FreeDOS–basierte Partition–Table–Doctor–Live–CD bootet dann einfach nicht. Und wenn ich beide Disketten–Laufwerke im BIOS abhänge? Dann erhält die DOS–Emulation wie gehabt den Laufwerkbuchstaben `a:`. Man lernt nie aus! Schöner wäre es gewesen, und es hätte mir Arbeit erspart, wenn ich diese Informationen gleich zu Anfang hätte nachlesen können.

Man stelle mir nicht die Frage: *Wozu willst du das eigentlich wissen?* Ich will es halt wissen! Und mit Forth als schnell anpassbares Analyse–Werkzeug darf ich mir eine solche Haltung auch erlauben.

Konvertierung von LBA nach CHS

Natürlich kann man die ganze Angelegenheit auch umkehren: Von LBA nach CHS. Damit könnte man — aus welchem Grund auch immer — ein mit den INT13h–Funktionen 42h und 43h in Assembler erstelltes Sektoren–Lese/Schreib–Programm über die Funktionen 2h und 3h von INT13h neu fassen — soweit die CHS–Grenze hier nicht eine Schranke setzt. Wie man (durch Vergleich mit der obigen Formel für die Berechnung der logischen Sektornummer LBA aus den CHS–Angaben heraus) leicht nachprüft, lautet die *Umkehrformel*:

$$\begin{aligned} (s+1)/(S/K*K/Z) &= C + Rest1 \\ Rest1/(S/K) &= H + Rest2 \\ Rest2 &= S \end{aligned}$$

Links

`http://de.wikipedia.org/wiki/Cylinder_Head_Sector`
`http://www.ptdd.com`
`http://www.freedos.org`

Listing

```
1   \ *******************************************************
2   \ *                                                     *
3   \ *  BOOTMA-3.FTH                                       *
4   \ *                                                     *
5   \ *  Zutaten fuer FAT-Reparatur und Bootmanager unter   *
6   \ *  Turbo-FORTH-83 und ZF                              *
7   \ *                                                     *
8   \ *  Fred Behringer - Forth-Gesellschaft - 01.03.2009   *
9   \ *                                                     *
10  \ *******************************************************
11
12  \ Fuer Turbo-Forth:
13  \ FORTH INCLUDE BOOTMA-1.FTH INCLUDE BOOTMA-2.FTH INCLUDE BOOTMA-3.FTH [ret]
14  \ Hierbei ist die erste Include-Datei die aus Heft 3/2008, die zweite die aus
15  \ Heft 4/2008 und die dritte die hier zu besprechende. Man beachte, dass man
16  \ hier - bis zum allerletzten INCLUDE (man darf natuerlich auch alles
17  \ kleinschreiben) - alle Eingaben 'schachteln' kann, ohne zwischendurch per
18  \ [ret] 'absetzen' zu muessen.
19
20  \ Fuer ZF:
21  \ ZF [ret] [ret]
22  \ FLOAD BOOTMA-1.FTH [ret] FLOAD BOOTMA-2.FTH [ret] FLOAD BOOTMA-3.FTH [ret].
23  \ In BOOTMA-1.FTH muss vorher attributs off per '\' auskommentiert werden (in ZF
24  \ gibt es kein attributs). Bei ZF, zumindest in der erweiterten Fassung der
25  \ Lokalen Forth-Gruppe Moers, geht die Schachtelung der Kommandozeilen-Eingaben
26  \ nicht, ohne dass man zwischendurch per [ret] absetzt. Die Eingabe von ZF muss
27  \ sogar zweimal quittiert werden.
```

```
28
29
30  \ Glossar
31
32  \ chs>lba ( c h/c h s/h sp -- sl ) \ aus CHS-Sektor-Angaben LBA-Wert ermitteln
33  \ lba>chs ( sl h/c s/h -- c h sp ) \ aus LBA-Sektor-Wert CHS-Angaben ermitteln
34  \ getLBAsect      ( sl -- fl )     \ LBA-Sektor in den Puffer sectbuf holen
35  \ putLBAsect      ( sl -- fl )     \ LBA-Sektor aus sectbuf zurueckschreiben
36  \ sect>bak           ( -- )        \ sectbuf in Datei SECT.BAK schreiben
37  \ bak>sect           ( -- )        \ Sektor aus SECT.BAK nach sectbuf holen
38  \ mbr>bak            ( -- )        \ sect>bak fuer mbr, Holen schon von HD
39  \ bak>mbr            ( -- )        \ bak>sect fuer mbr, Schreiben nach HD
40
41  \ Will man sect>bak zum Sichern eines Sektors (z.B. des MBRs) verwenden, und
42  \ bak>sect zum Rueckschreiben nach sectbuf, dann muss man den Vorgang noch durch
43  \ getLBAsect oder putLBAsect (bei LBA-Betrachtungsweise) oder (getsect) oder
44  \ (putsect) (bei CHS-Betrachtungsweise) vervollstaendigen, damit er auf die
45  \ Festplatte einwirkt.
46
47  \ c    : Zylindernummer
48  \ h    : Kopfnummer
49  \ sp   : Sektornummer (phys., also nach CHS)
50  \ sl   : Sektornummer (log.,  also nach LBA)
51  \ h/c  : Koepfe pro Zylinder
52  \ s/h  : Sektoren pro Kopf
53
54  \ Die Nummer des zugeordneten logischen Sektors berechnet sich (vergleiche
55  \ Artikeltext) zu:
56
57  \ sl   = ((c * h/c + h) * s/h) + sp - 1
58
59  \ Das Forth-Wort chs>lba liefert aus den CHS-Eingaben heraus den logischen
60  \ Sektor sl. Der Einsatz von chs>lba hat natuerlich nur dann einen Sinn, wenn
61  \ das BIOS der Anlage den erweiterten Interrupt 13h (insb. den Platten-Zugriff
62  \ auf mehr als 8 GB) ueberhaupt verarbeiten kann.
63
64  \ Es ist nicht gerade leicht, solche Aufgaben wie die vorliegende mit einem
65  \ 16-Bit-Forth-System bewaeltigen zu wollen. Ich mache daher die folgenden
66  \ Einschraenkungen: Eingaben seien einfachgenau. Nicht unueblich sind
67  \ h/c = 255 und s/h = 63. Wenn man beides malnimmt (vergleiche Formel), bekommt
68  \ man wieder eine Zahl, die mit 16 Bit auskommt. Koepfe und Sektoren bereiten
69  \ also keine Schwierigkeiten. Bei c (Zylinder) macht sich die Beschraenkung (bei
70  \ den heutigen Festplattenkapazitaeten) am staerksten bemerkbar: Uebergrosse
71  \ Platten sind nicht mehr so einfach behandelbar. Man koennte das Forth-Wort
72  \ chs>lba als Code-Definition fassen, um der 16-Bit-Beschraenkung elegant zu
73  \ entfliehen. Ich will darauf verzichten, da ja die angestrebte Umrechnung
74  \ sowieso nur dann sinnvoll ist, wenn die Bestandteile der CHS-Adressierung,
75  \ C-H-S, in Groessenordnungen liegen, die ohnehin ins Schema passen.
76
77  : chs>lba ( c h/c h s/h sp -- sl ) \ sl doppeltgenau, alle anderen einfach
78      dup 1 < >r 2dup < r>    \ Nicht 1 <= sp <= s/h ?
79      or abort" Fehler!"      \ Ja, dann Fehler und Abbruch.
80      1 - s>d >r >r           \ Sonst: Sektoranteil doppeltgenau nach r:
81      over ff00 and           \ Oberes Nibble <> 0 ?
82      abort" Fehler!"         \ Ja, dann Fehler und Abbruch.
83      tuck um* >r >r          \ Sonst: h*s/h doppeltgenau nach r:
84      * um*                   \ s/h * h/c * c
85      r> r> r> r> d+ d+ ;
86
87  \ Das folgende Forth-Wort lba>chs liefert bei gegebenem LBA-Sektor die
```

```
\ zugehoerigen CHS-Werte. Die Hoechstzahl an Koepfen (pro Zylinder) und die
\ Hoechstzahl an Sektoren (pro Kopf) muessen ebenfalls bekannt sein. Leitbild
\ ist die folgende Formel (siehe Artikeltext):

\    (sl+1)/(h/c * s/h) = c + Rest1
\    Rest1 /(      s/h) = h + Rest2
\    Rest2              = sp

: lba>chs ( sl h/c s/h -- c h sp ) \ sl doppeltgenau, alle anderen einfach
     >r r@ * >r              \ erst s/h nach r: , dann h/c * s/h nach r:
     1. d+                   \ ab jetzt Sektorzaehlung beginnend bei 1
     r> um/mod               \ (sl+1)/(h/c * s/h) verarbeiten
     swap r>                 \ c Rest1 s/h
     /mod swap ;             \ c h Rest2 (= c h sp)

\ Ich verzichte hier auf die Absicherung gegen Eingabefehler bei h/c und s/h.
\ Die Technik dazu kann aus dem Forth-Wort chs>lba uebernommen werden.

\ Was im vorliegenden Artikel eigentlich interessiert, ist die Entwicklung eines
\ Forth-Wortes, mit dessen Hilfe man sich Sektoren auch von groesseren
\ Festplatten ansehen kann, Sektoren also, bei denen die CHS-Adressierung
\ versagt und bei denen man auf LBA-Adressierung zurueckgreifen muss. Dass man
\ solche Sektoren dann auch wieder 'zurueckschreiben' kann, versteht sich von
\ selbst. Zunaechst benoetige ich einen Puffer zur Parameter-Uebergabe:

28 allot here  \ Platz fuer mindestens 18h Bytes
here 0f and -  \ dap (disk address packet) an den Anfang des Paragraphen
18 -           \ Anfangsadresse des dap
constant dap   \ Liefert bei Aufruf Anfang des dap-Puffers

code csegment ( -- cseg ) cs push next end-code

\ In Turbo-Forth gibt es die Konstante dsegment. Da man bei Definition des
\ Sektorpuffers sectbuf die Segmente ds und cs als wertegleich behandeln kann,
\ haette im gleich folgenden Forth-Wort getLBAsect der Aufruf der Konstanten
\ dsegment gereicht, um fuer das dap an das sectbuf-Segment zu gelangen. In ZF
\ gibt es aber keine Konstante dsegment und es war mir zu umstaendlich
\ herauszusuchen, was man in solchen Faellen in ZF zu machen hat. Daher der
\ Zusatzaufwand mit der Pseudokonstanten csegment. (In ZF ausprobiert! Geht
\ prima!)

\ LBA-Sektor sl (32 Bit) vom Boot-Laufwerk (hier immer 80) nach Puffer sectbuf
\ holen. fl <> 0 bedeutet Fehler (den ich hier nicht naeher spezifizieren
\ moechte). Ein paar leicht einfuegbare Zusatzzeilen wuerden es auch gestatten,
\ gleich mehrere Sektoren in einem Zug einzulesen (oder abzuspeichern). In
\ voller Allgemeinheit waere das unnoetig schwierig. Eine brauchbare
\ Organisation des dann noetig werdenden groesseren Transfer-Puffers wuerde
\ zusaetzliche Ueberlegungen kosten. Ich verzichte hier darauf. Theoretisch
\ koennte ich ja mit drei Forth-Zeilen eine Schleife auch schon mit der
\ Code-Definition getLBAsect aufbauen. Dass mit solch groben Mitteln dann jedoch
\ beispielsweise das Klonen einer ganzen Festplatte unertraeglich lange dauern
\ wuerde, ist leicht vorauszusehen.

code getLBAsect ( sl -- fl ) \ sl doppeltgenau; Fehler, wenn fl <> 0
      18 # dap      #) mov   \ Laenge des dap   nach dap
       1 # dap 02 + #) mov   \ Sektoranzahl (1) nach dap
 sectbuf # dap 04 + #) mov   \ Adresse sectbuf  nach dap
csegment # dap 06 + #) mov   \ Segment sectbuf  nach dap
      80 #          dl mov   \ Boot-Laufwerk    nach dl
                    ax pop   \ sl hi
```

```
148           ax dap 0a + #) mov   \                  nach dap
149                          ax pop   \ sl lo
150           ax dap 08 + #) mov   \                  nach dap
151          0 # dap 0c + #) mov   \ uebergrosse Anteile auf 0 legen
152          0 # dap 0e + #) mov   \ dito
153                          si push  \ si (=ip) sichern
154        dap #             si mov   \ ds:si mit ds:dap laden
155       4200 #             ax mov   \ Funktion 'erweitertes Lesen' (LBA)
156                          13 int   \ Interrupt aufrufen
157                          si pop   \ si (=ip) wiederherstellen
158                     ah al mov   \ Jetzt ax = 0 bei Erfolg
159                          ax push  \ ax <> 0  => Fehler
160                  next end-code
161
162   \ Achtung: Wenn man mit getLBAsect beim ersten Versuch Schiffbruch erleiden
163   \ sollte, dann ist es so gut wie sicher, dass man sl einfachgenau statt
164   \ doppeltgenau angesetzt hat! sl muss als 'Punktzahl' eingegeben werden!
165
166   \ In Offset 'dap 04 +' von dap wird die 'Lang-'Adresse des Uebertragungspuffers
167   \ in der Form segment:sectbuf (bei mir beispielsweise 2860:6750) festgehalten,
168   \ und zwar in der Little-Endian-Form 50 67 60 28 . Steht hier der Wert
169   \ ffff:ffff, dann soll laut 'Browns Liste' (ich hatte keine Veranlassung, das
170   \ nachzupruefen) ab Offset 'dap 10 +' die lineare 64-Bit-Adresse des
171   \ Uebertragungspuffers abgelegt werden.
172
173   \ Und hier das Schreib-Pendant zu getLBAsect.
174   \ Man huete sich vor unueberlegtem Aufruf!
175
176   code putLBAsect ( sl -- fl ) \ sl doppeltgenau; Fehler, wenn fl <> 0
177         18 # dap      #) mov   \ Laenge des dap   nach dap
178          1 # dap 02 + #) mov   \ Sektoranzahl (1) nach dap
179    sectbuf # dap 04 + #) mov   \ Adresse sectbuf  nach dap
180   csegment # dap 06 + #) mov   \ Segment sectbuf  nach dap
181         80 #             dl mov   \ Boot-Laufwerk    nach dl
182                          ax pop   \ sl hi
183           ax dap 0a + #) mov   \                  nach dap
184                          ax pop   \ sl lo
185           ax dap 08 + #) mov   \                  nach dap
186          0 # dap 0c + #) mov   \ uebergrosse Anteile auf 0 legen
187          0 # dap 0e + #) mov   \ dito
188                          si push  \ si (=ip) sichern
189        dap #             si mov   \ ds:si mit ds:dap laden
190       4300 #             ax mov   \ Funktion 'erweitertes Schreiben' (LBA)
191                          13 int   \ Interrupt aufrufen
192                          si pop   \ si (=ip) wiederherstellen
193                     ah al mov   \ Jetzt ax = 0 bei Erfolg
194                          ax push  \ ax <> 0  => Fehler
195                  next end-code
196
197
198   \ Und wozu das Ganze?
199   \ --------------------
200   \ Wenn man nach vielem Experimentieren endlich solche Forth-Worte wie die
201   \ obigen vier 'auf die Beine gestellt' hat, haelt man die Frage nach
202   \ Anwendungsmoeglichkeiten fuer selbstverstaendlich und daher fuer
203   \ ueberfluessig. Man kann es nicht verstehen, wenn man trotzdem gefragt wird:
204   \ "Und warum hast du dich jetzt damit beschaeftigt? Was kann man denn damit
205   \ anfangen?" - Man will ueberhaupt gar nichts damit anfangen! Man macht es, weil
206   \ man es (endlich) verstanden hat und es jetzt kann. Was treibt denn ein Kind
207   \ dazu, eine Sandburg zu bauen? Und einen Maler, ein Bild zu malen? Und einen
```

```
208  \ Wissenschaftler, zwanzig Jahre seines Lebens in die Suche nach einer Loesung
209  \ eines (meist sogar selbsterfundenen) Problems zu stecken?
210
211  \ Na ja, ich darf beispielsweise auf das grosse Problem der Datensicherung, des
212  \ 'Backups', hinweisen: Es gibt Betriebssysteme, die bei ihrer Installation den
213  \ Master-Boot-Record (den MBR) eines anderen, schon installierten
214  \ Betriebssystems ueberschreiben. Es gibt 'Anwender', die sich dann nicht mehr
215  \ zu helfen wissen. Wie leicht ist es aber, wenn man sich fuer solche Faelle ein
216  \ Forth-Tool-Belt geschmiedet hat, mit dessen Hilfe MBRs in eine Datei
217  \ geschrieben (gesichert) und von dort auch wieder zurueckgeholt werden koennen.
218  \ Dazu braucht man wirklich kein 'Paragon' oder 'Acronis' oder sonstwie
219  \ geartetes 180-MB-Paket zu kaufen - und auch noch zwangsregistrieren zu lassen!
220  \ Unter Linux laesst sich der MBR mit einer einzigen dd-Zeile in eine Datei
221  \ kopieren. Mit DEBUG aus DOS geht es nicht, jedenfalls nicht 'auf Anhieb'!
222  \ (Man mache sich klar, warum nicht.) Aber mit Forth! In Forth geht alles! Und
223  \ man kann es sich genau so einrichten, wie man es in seiner
224  \ hoechstpersoenlichen Programmierumgebung gerade braucht. Selbstverstaendlich
225  \ kann man mit wenigen Forth-Handgriffen (siehe Teil 1 und 2 der vorliegenden
226  \ Serie) auch die 'Bootsektoren' der einzelnen Partitionen und sogar die
227  \ 'Partitionstabellen' der logischen Laufwerke der erweiterten Partition in
228  \ Dateien sichern.
229
230  \ Ein weiteres Vorhaben waere der Umzug der zu eng gewordenen Festplatte auf
231  \ eine inzwischen preisguenstigere groessere Platte, das 'Klonen'. Was fuer ein
232  \ Aufwand, wenn man herausbekommen will, welches kostenlose Klon-Programm
233  \ unter welchen Bedingungen funktioniert! Geht das eine fuer XP konzipierte auch
234  \ unter ME? Und so weiter. Mit den in der vorliegenden Serie entwickelten
235  \ wenigen Forth-Worten koennte man ein solches Klon-Vorhaben bei genuegend
236  \ Geduld, was die Ausfuehrzeiten betrifft, schon jetzt in Angriff nehmen. Und
237  \ wem es dabei zu lange dauert: Die ganze Forth-Welt steht einem ja fuer den
238  \ Einbau in ein eigenes automatisierendes Programm offen!
239
240  \ Hier also (als Alibi fuer die Verwendungsmoeglichkeit) zum Abschluss zwei
241  \ Forth-Worte (mbr>bak und bak>mbr), mit deren Hilfe der MBR (der vom BIOS als
242  \ erste Boot-Platte eingestuften Festplatte) in die Datei MBR.BAK geschrieben
243  \ und von dort auch wieder zurueckgeholt werden kann. Die Datei MBR.BAK braucht
244  \ nicht schon zu bestehen. Ich fasse das Vorhaben zunaechst gleich etwas
245  \ allgemeiner (fuer beliebige Sektoren, nicht nur fuer den MBR) auf:
246
247  \ Achtung: Fuer ZF sind die jetzt folgenden Worte sect>bak und mbr>bak und
248  \ bak>sect und bak>mbr herauszunehmen (siehe gleichfolgende Bemerkungen)!
249
250  : sect>bak ( -- ) " sectbuf dup 200 + save sect.bak " $execute ;
251
252  \ Das reicht an sich. Aber hier noch eine auf den MBR spezialisierte Fassung:
253
254  : mbr>bak  ( -- ) getmbr " sectbuf dup 200 + save mbr.bak " $execute ;
255
256  \ Einfacher kann es wirklich nicht gehen! Trotzdem, ZF kann mit 'save' nichts
257  \ anfangen. Da wurde nun damals, fast gleichzeitig, an verschiedenen Enden der
258  \ Welt enorme Energie in die Umwandlung von F83 in die Abkehr vom Blocksystem
259  \ hin zur DOS-Dateiverwaltung gesteckt - und dabei hat man vergessen, auf die
260  \ Muttersprache des jeweils anderen Entwicklers einzugehen, ja vom anderen
261  \ Entwickler ueberhaupt Notiz zu nehmen! Es gab keine Kommunikation. Alles,
262  \ was mit der Dateiverwaltung zu tun hat, wurde (beim Uebergang von F83 zu
263  \ Turbo-Forth oder ZF) auf zwei grundverschiedene Arten programmiert. Schade!
264
265  \ Fuer Turbo-Forth-Fans: Man beachte den Einsatz von $execute ! Das ist eine
266  \ einleuchtende Moeglichkeit, das ganze Hin-und-Her (von ANS-Forth) um compile
267  \ und [compile] und postpone zu umgehen!
```

```
268
269  \ Und schliesslich - ich will ja beispielsweise den MBR backuppen, ohne dazu das
270  \ 180 Megabyte grosse Acronis-TrueImage kaufen, registrieren und ankurbeln zu
271  \ muessen - das Pendant zu mbr>bak zum Zurueckschreiben des MBRs (von Datei
272  \ wieder auf Festplatte). Wie gross ist das Gezeter, wenn sich beispielsweise
273  \ GRUB beim Installieren von Linux (aus Versehen) im MBR eingenistet hat, dort,
274  \ wo eigentlich der Dual-Boot-Manager von XP haette liegenbleiben sollen - oder
275  \ umgekehrt, von Linux zu XP! Also versuche ich mich abschliessend noch an einem
276  \ Forth-Wort bak>mbr, mit welchem ich den zuvor in eine Datei namens mbr.bak
277  \ gespeicherten MBR wieder zurueckschreiben kann.
278
279  \ Auf Anhieb findet man dazu auch in Turbo-Forth nichts. Nichts von der Art
280  \ eines inversen 'save's, ein auf Dateien bezogenes 'load'. Warum eigentlich
281  \ nicht? Aber gemach! Wenn man in den dateibezogenen Worten von Turbo-Forth
282  \ herumstoebert (ueber HELP und SEE und DUMP), findet man, dass das Wort (get)
283  \ schon fast alles bereitstellt:
284
285  : bak>sect ( -- )
286      " open sect.bak >r dsegment sectbuf 200 r> (get) close " $execute
287      nip abort" Fehler!" ;
288
289  : bak>mbr  ( -- )
290      " open mbr.bak >r dsegment sectbuf 200 r> (get) close " $execute
291      nip abort" Fehler!" putmbr ;
292
293  \ Vorsicht auch wieder bei Aufruf dieser Worte! Enthaelt die Datei mbr.bak
294  \ tatsaechlich den MBR? Oder/und hat man den MBR sicherheitshalber wenigstens
295  \ noch woanders aufbewahrt?
296
297  \ Sicherer ist es mit sect>bak und bak>sect (statt mit mbr>bak und bak>mbr).
298  \ sect>bak und bak>sect stellen nur den Datenverkehr vom Puffer sectbuf zur
299  \ Sicherungsdatei und zurueck dar. Gefaehrlich wird es dann erst beim dann
300  \ noetig werdenden Wort (putsect) oder/und putmbr (siehe Teil 1).
301
302  \ In ZF funktioniert bak>sect und bak>mbr so einfach 'natuerlich' auch wieder
303  \ nicht. Es werden ja einige Turbo-Forth-eigene Spezialitaeten verwendet. Das
304  \ ist aber genau das Dilemma, vor dem man steht, wenn es (wie z.B. auch bei
305  \ ANS-Forth) um Standardisierungen geht: Gerade solche Feinheiten wie die eben
306  \ erwaehnten, von denen die praktikable Einsetzbarkeit von Forth-Programmen in
307  \ erheblichem Masse abhaengen kann, werden von den Standardisierungsempfehlungen
308  \ kaum beachtet.
309
310  \ ZF-Fans werden sich vielleicht aufgerufen fuehlen, eine ZF-Fassung fuer
311  \ mbr>bak und bak>mbr (oder sect>bak und bak>sect) zu schreiben!
```

nach einer Idee von Jessica Ledbetter (jledbetter@acm.org)

Forth im FPGA — Teil 1, der Einstieg

Ulrich Hoffmann

Einen Forth–Prozessor in Hardware zu realisieren, ist ein Wunsch, den wohl jeder Forth–Programmierer schon einmal gehabt hat. Musste dieser Wunsch in der Vergangenheit häufig unerfüllt bleiben — Halbleiterentwurf ist teuer und aufwändig — so bietet heute die Verfügbarkeit von FPGAs und kostengünstigen Entwicklungsboards die Möglichkeit, dass jeder sich an die Erfüllung dieses Wunsches machen kann. Aber es gilt, die Enstiegsschwelle zu überwinden. Zwar haben Bernd Paysan und Klaus Schleisiek, die beruflich mit FPGAs arbeiten, mit ihren Forth–Prozessoren b16 [6] und MicroCore [7] zwei FPGA–fähige Forth–Prozessoren vorgestellt, aber diese Prozessoren selber auf einem FPGA–Board zum Laufen zu bekommen, ist immer noch eine Herausforderung.

Dieser Herausforderung stellt sich die Artikelserie *Forth im FPGA*. Im vorliegenden ersten Teil wird eine geeignete Zielplattform, das DE1–Entwicklungsboard von Altera, vorgestellt und die darauf befindlichen Schalter und LEDs angesprochen, um eine Idee davon zu bekommen, wie Hardware–Programmierung mit FPGAs abläuft.

Das Ansprechen weiterer Hardware des Boards und die Portierungen von b16 und Microcore sind für die weiteren Teile von *Forth im FPGA* vorgesehen.

Einleitung

Eine der Besonderheiten von Forth ist, dass es auf einer virtuellen Maschine beruht, deren zwei Stacks direkt in der Programmiersprache sichtbar sind. Diese virtuelle Maschine ist so einfach, dass sie sich geradezu anbietet, direkt in Hardware realisiert zu werden und Forth so zur Maschinensprache eines Mikroprozessors zu machen. Zahlreiche Ansätze [1][2][3] für Forth–orientierte Mikroprozessoren, so genannte *Forth–Maschinen*, nicht zuletzt durch Charles Moore selbst, verfolgen diese Idee.

Klassischerweise werden Mikroprozessoren in einem kundenspezifischen und damit kostspieligen Entwurfsprozess hergestellt und ihr Entwurf ist damit für kleine Firmen oder versierte Laien unerschwinglich. Aber die Zeiten haben sich geändert: Prozessoren lassen sich heute in FPGAs realisieren.

FPGAs

Die so genannten *Field Programmable Gate Arrays* [8], kurz FPGA, sind allgemeine digitale Schaltungen, die durch eine gezielte Konfiguration spezielle Schaltungen realisieren können. Dazu sind allgemeine, vielseitige *Logikelemente* mit einem vorgegebenen Vorrat an digitalen Elementarfunktionen (UND, ODER, NICHT, usw., oder mit kleinen Funktionstabellen) in einer Matrix angeordnet. Die Konfiguration legt die Funktionalität jedes Logikelements und auch die Verbindungen zwischen den Elementen fest. Auf diese Weise entstehen Schaltnetze und Schaltwerke (rückgekoppelte Schaltnetze). Nach dem Einschalten lädt ein FPGA typischerweise die Konfiguration in Form eines Bitstroms aus einem seriellen Flash–Speicher. Das Definieren jedes einzelnen Bits der Konfiguration ist extrem mühsam, deswegen werden heute *Hardware–Beschreibungssprachen* (etwa VHDL oder Verilog) verwendet, um die Schaltungen — den *Entwurf* (Design) — auf abstrakter Ebene zu beschreiben. Die *Hardware–Synthese* gewinnt dann die Konfiguration aus dem Entwurf.

Moderne FPGAs besitzen eine Vielzahl von Logikelementen — das von uns verwendete Altera–FPGA Cyclone II 2C20 besitzt 20000 Logikelemente, genug um einen Mikroprozessor aufzunehmen — und außerdem auch spezielle Einheiten etwa zur Datenspeicherung (Block RAM) oder zur schnellen Multiplikation. Während der Synthese wird auf die speziellen Fähigkeiten des FPGAs Rücksicht genommen, der Entwurf selbst muss hingegen auf die Eigenheiten des jeweiligen FPGAs nur wenig Einfluss nehmen.

Das Altera–DE1-Entwicklungsboard

Mit dem Board DE1 von Altera/Terasic [4][5][9] steht ein kostengünstiges FPGA–Entwicklungssystem zur Verfügung, das jedermann Experimente mit FGPAs ermöglicht. Quartus, die Entwicklungsumgebung mit Projektverwaltung und Synthese für Altera–FPGAs, kann von der Altera–Webseite geladen werden und liegt dem System auf DVD bei.

Das DE1–Board (siehe Abbildung 1 auf der nächsten Seite) enthält ein Cyclone II 2C20 FPGA mit zugehörigem Konfigurations–Flash EPCS4. Ebenfalls auf dem Board vorhanden ist Speicher: 8 MB SDRAM, 512 KB SRAM und 4 MB Flash. Für den Mikrocontroller–Board–Interessierten ist das nichts Besonderes. Auf einem FPGA–Board muss man aber geeignete Ansteuerungsschaltungen in seinem Entwurf selbst definieren, bevor man den Speicher überhaupt ansprechen kann — sicher ein Thema, mit dem wir uns in Zukunft auseinandersetzen werden. Zur Taktung von Schaltungen können 3 Oszillatoren mit 24, 27 bzw. 50 Mhz verwendet werden.

Als Verbindungen zur Außenwelt stehen zum einen Schiebeschalter, Taster, rote/grüne LEDs und eine vierstellige 7–Segment–Anzeige zur Verfügung, die sich besonders gut für eigene Experimente eignen. Zum anderen gibt es Schnittstellen, die sich an typischen PC–Schnittstellen orientieren: VGA zur Videoausgabe, RS232 zur seriellen Kommunikation, PS/2 zum Anschluss von PC–Tastaturen, Klinkenbuchsen zur Audio–Ein– und Ausgabe (Line/Mikrofon, 24–Bit–Codec) und

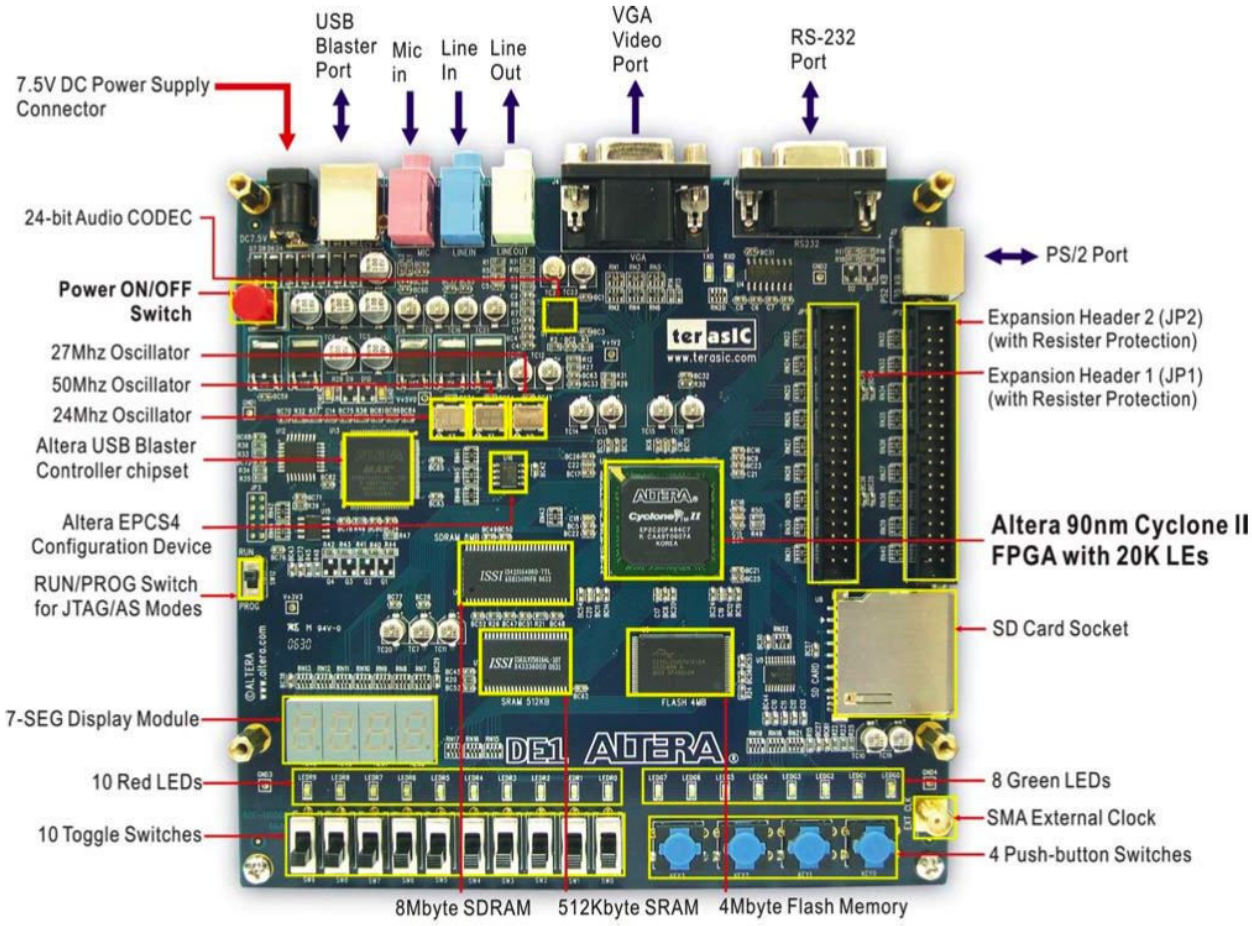


Abbildung 1: Anschlüsse des Altera–DE1–Entwicklungsboards [13]

ein Steckplatz für SD–Karten als Massenspeicher. Auch hier sind die Anschlüsse zwar vorhanden, aber die Schnittstellen benötigen passende Ansteuerungsschaltungen, um sie zum Leben zu erwecken.

Eigene Elektronik lässt sich über zwei Steckverbinder anschließen.

Abbildung 2 auf der gegenüberliegenden Seite zeigt den Lieferumfang des DE1–Systems. Neben dem Board selbst findet sich ein Steckernetzteil, ein USB–Verbindungskabel zur Kommunikation und zum Programmieren des FPGAs, eine DVD mit Dokumentation, Software und Beispiel–Entwürfen und ein kleines Anleitungsheftchen (nicht auf dem Bild zu sehen). Das Steckernetzteil unterstützt 110V und auch 220V, hat aber einen amerikanischen Stecker. Das Netzteil und damit ein Adapter für deutsche Steckdosen wird aber erst benötigt, wenn man Erweiterungen an das Board anschließt: Das Board alleine vesorgt sich auch problemlos über das USB–Kabel vom angeschlossenen PC. Sehr nett.

Um ein FPGA zu programmieren, wird in der Regel ein JTAG–Programmieradapter benötigt. Beim DE1 ist dieser Adapter, Altera nennt ihn *USB-Blaster*, jedoch bereits auf dem Board integriert, so dass das Board direkt per USB an einen PC angeschlossen werden kann und dann direkt über dieses Kabel programmiert wird. Auch sehr nett.

Inbetriebnahme

Schließt man das DE1 also über das mitgelieferte USB–Kabel an einen PC an und schaltet es ein, so wird die im Konfigurations–Flash befindliche vordefinierte Standard–Schaltung (`CII_Starter_Default`) in das FPGA geladen und aktiviert: Die 7–Segment–Anzeige zählt, die LEDs zeigen Lauflichter und auf VGA und Lineout werden Testsignale produziert. Einige Taster führen einen Reset durch. So kann man schnell feststellen, ob das Board im Prinzip funktioniert.

Alteras FPGA–Entwicklungssystem Quartus

Lange Zeit war FPGA–Entwurfssoftware — ihre Hauptaufgaben sind eben die Umsetzung des Hardware–Entwurfs in die Konfigurationsinformation des FPGAs und die Simulation des Entwurfs — teure Spezialwerkzeuge für den Hardware–Ingenieur. Erfreulicherweise hat sich dieses Bild geändert: Alle gängigen FPGA–Anbieter, wie Xilinx, Altera oder Lattice, bieten neben den Profi–Versionen ihrer Entwurfswerkzeuge auch freie Einstiegsversionen an. Häufig ist es nötig, dennoch eine (kostenfreie) Lizenz beim Anbieter anzufordern — Accounts auf den jeweiligen Hersteller–Web–Seiten sind unabdingbar. Hier nimmt Altera, deren Entwicklungswerkzeug *Quartus II* heißt, derzeit (März 2009) eine Vorreiterrolle ein: Dem DE1–Board liegt, wie oben beschrieben eine DVD

Abbildung 2: Lieferumfang des Altera–DE1–Entwicklungssystems [12]

mit der kostenfreien Quartus Variante *Quartus II Web Edition* [10] in der Version 7.2 bei. Zur Benutzung ist, wie eben erwähnt, eine kostenfreie Lizenz von Altera erforderlich, die alle 150 Tage erneuert werden muss. Aber Altera entwickelt Quartus nicht nur technisch weiter. Mittlerweile gibt es die Version 9 zum Download auf der Altera–Homepage und für ihren Betrieb ist keine Lizenz mehr nötig. Nett.

Quartus Web Edition läuft unter Windows XP oder Vista (die Profi–Version auch unter Linux). Es stellt eine integrierte Entwicklungsumgebung klassischen Zuschnitts mit Editor, Projekt– und Versionsverwaltung und zahlreichen Konfigurationsdialogen bereit und integriert die Entwurfswerkzeuge unter einer einheitlichen Oberfläche (siehe Abbildung 3 auf der nächsten Seite).

Quartus verarbeitet Hardware–Entwürfe in Verilog und in VHDL. Wir werden im Verlauf der Artikelserie beide näher kennenlernen. Doch zunächst wollen wir uns auf Verilog konzentrieren. Zur Vertiefung bietet sich das Buch *Digital Design* [11] von Morris Mano an, das eine gute Einführung in den Entwurf digitaler Systeme und zugehörige Beispiele in Verilog enthält.

Nach der Quartus–Installation, eine einfache Click-Click-Click–Installation, schließen wir das Board an den PC an. Es muss der USB–Treiber für den USB–Blaster installiert werden. Er findet sich im Verzeichnis ...`\quartus\drivers\usb-blaster`.

Ansteuern der Schalter und LEDs

Unser allererster Verilog–Entwurf soll die Schiebeschalter auf dem Board auslesen und ihren Zustand auf den darüber befindlichen roten Leuchtdioden ausgeben. Außerdem lassen wir die grünen Leuchtdioden binär zählen. Zunächst gilt es, ein passendes Quartus–Projekt anzulegen. Der *New Project Wizard* führt uns durch die notwendigen Schritte: Als Projektnamen wählen wir `DE1LEDs` und auch unsere Schaltung trägt diesen Namen. Als anzusteuerndes FPGA wählen wir unter den zahlreichen FPGA–Varianten den auf dem DE1 befindlichen `Cyclone II EP2C20F484C7` als *Device*. Als Synthese–Werkzeug verwenden wir den *Design Compiler*, einen Simulator und ein Werkzeug zur Analyse der Signallaufzeiten benötigen wir im Moment noch nicht.

Das FPGA ist an die unterschiedlichen Schnittstellen des Boards abgeschlossen. Welcher Pin des FPGAs mit welcher Schnittstelle verbunden ist, kann je nach verwendetem Board natürlich variieren. Wie die Zuordnung im konkreten Fall aussieht, wird durch das *PIN–Assignment* festgelegt. Für das DE1–Board importieren wir das Pin–Assignment `CII_Starter_Default.qsf` aus den Beispiel–Entwürfen.

Ein Verilog–Entwurf besteht aus einer Hierarchie von Modulen, die jeweils eine Schnittstelle mit Eingangs– und Ausgangs–Signalen besitzen. Das Innere der Module legt die Verknüpfung der Schnittstellensignale fest. Ein ausgezeichnetes Haupt–Modul besitzt schließlich die Schnittstelle zur Board–Peripherie. Unsere erste Schaltung besteht nur aus diesem Hauptmodul. Wir definieren die Schnittstelle, versetzen die als `inout` gekennzeichneten Signale in den Tri–State–Zustand, dann definieren wir unsere eigentliche Schaltung:

```
reg [27:0] Cont;
always@(posedge CLOCK_50) Cont <= Cont+1'b1;
assign LEDG = Cont[27:20];
assign LEDR = SW;
```

Wir definieren ein 28 Bit breites Register `Cont`. Immer wenn das Eingangs–Signal `CLOCK_50`, das mit dem 50 Mhz Oszillator verbunden ist, eine positive Flanke hat, wird dieses Register um 1 erhöht. (Die Konstante `1'b1` beschreibt eine 1 Bit breite Binärzahl mit dem Wert 1.)

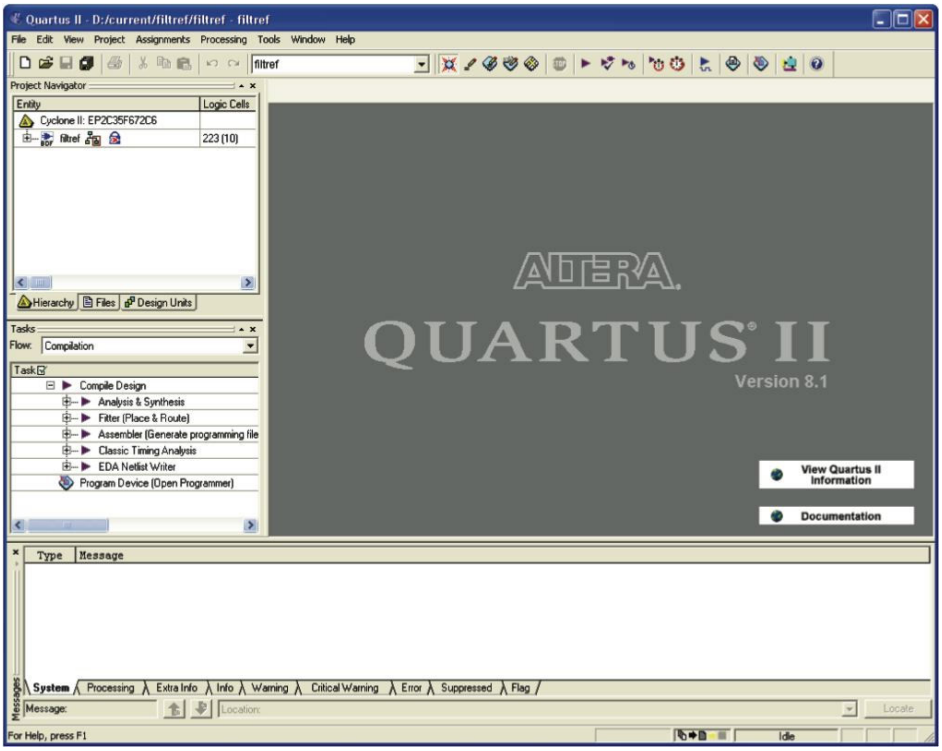


Abbildung 3: Ein Blick auf den Hauptbildschirm der Altera–Hardware–Entwicklungsumgebung Quartus [10]

Dann verbinden wir die 8 grünen LEDs (`LEDG`) mit den Bits 27 bis 20 des `Cont`–Registers. Die grünen LEDs werden passend binär zählen. Schließlich verbinden wir die Schiebeschalter (`SW`) mit den roten Leuchtdioden (`LEDR`). Der vollständige erste Verilog–Entwurf ist im Listing auf der gegenüberliegenden Seite abgedruckt.

Dieses Listing nehmen wir in das oben angelegte Projekt auf und starten mit *Start Compilation* die Synthese. Haben wir alles richtig gemacht, entsteht dabei die FPGA–Konfiguration in Form des Files `DE1LEDs.sof`. Dieses File können wir jetzt auf das DE1–Board laden. Das geschieht über den Menüpunkt *Programmer*. Wir wählen beim ersten Mal den USB–Blaster als unser Programmiergerät (*Hardware Setup*) und dann kann mit *Start* der Download beginnen. Die Standard–Anwendung auf dem Board verschwindet und nach kurzer Zeit fangen die grünen Leuchtdioden an zu zählen. Ein bisschen Schieben der Schalter zeigt uns, dass auch die LEDs richtig mit den Schiebeschaltern verbunden sind. Es ist geschafft. Unser erster Verilog–Entwurf läuft im FPGA!

Ausblick

Nun wollen wir ja nicht nur Leuchtdioden blinken lassen. Die auf dem Board integrierte 7–Segment–Anzeige können wir gut als Debugging–Werkzeug für die Suche nach Fehlern in unseren Schaltungen brauchen. Sie soll unsere Lupe in das FPGA werden. Wenn wir Zweifel über den Wert bestimmter Signalleitungen haben, so können wir versuchen, diese auf der 7–Segment–Anzeige darzustellen und so unseren Entwurf bestätigen oder Ungereimtheiten finden. Auch unsere Forth–Prozessoren werden sich über diese Anzeige bemerkbar machen können. Die Ansteuerung der 7–Segment–Anzeige soll unsere nächste Aufgabe sein, die wir im kommenden Teil 2 bearbeiten.

Literatur

[1] `http://www.colorforth.com/` Homepage von Charles Moore, Forth–Prozessor 25x
[2] `http://www.intersil.com/cda/deviceinfo/0,1477,HS-RTX2010RH,0.html` Produktseite des RTX2010 bei Intersil
[3] *Stack Computers: The New Wave*, Phillip Koopman, Ellis Horwood, 1989,
auch online unter `http://www.ece.cmu.edu/~koopman/stack_computers/index.html`
[4] `http://www.altera.com` Homepage von Altera
[5] `http://www.terasic.com` Homepage von Terasic, dem DE1-Board Hersteller
[6] `http://www.jwdt.com/~paysan/b16.html` Bernd Paysans Forth–Prozessor b16
[7] `http://www.microcore.org` Klaus Schleisieks Forth–Prozessor MicroCore
[8] `http://http://de.wikipedia.org/wiki/Field_Programmable_Gate_Array` Wikipedia–Eintrag zu FPGAs
[9] `http://www.altera.com/products/devkits/altera/kit-cyc2-2C20N.html` Beschreibung/Resourcen für das DE1–Board
[10] `http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html` Quartus II Web–Edition
[11] *Digital Design, 3rd edition*, Morris Mano, Prentice Hall, 2001
[12] *DE1 Development and Education Board — Introduction*, DE1–Doku im File `DE1_introduction_box.pdf` siehe [9]
[13] *DE1 Development and Education Board — User Manual*, DE1–Doku im File `DE1_UserManual_v1018.pdf` siehe [9]

```
module DE1LEDs
    (
        ////////////////////    Clock Input     ////////////////////
        CLOCK_24,                       //  24 MHz
        CLOCK_27,                       //  27 MHz
        CLOCK_50,                       //  50 MHz
        EXT_CLOCK,                      //  External Clock
        ////////////////////    Push Button     ////////////////////
        KEY,                            //  Pushbutton[3:0]
        ////////////////////    DPDT Switch     ////////////////////
        SW,                             //  Toggle Switch[9:0]
        ////////////////////    7-SEG Dispaly   ////////////////////
        HEX0,                           //  Seven Segment Digit 0
        HEX1,                           //  Seven Segment Digit 1
        HEX2,                           //  Seven Segment Digit 2
        HEX3,                           //  Seven Segment Digit 3
        //////////////////////////    LED     //////////////////////////
        LEDG,                           //  LED Green[7:0]
        LEDR,                           //  LED Red[9:0]
        //////////////////////////    UART    //////////////////////////
        UART_TXD,                       //  UART Transmitter
        UART_RXD,                       //  UART Receiver
        ////////////////////    SDRAM Interface     ////////////////
        DRAM_DQ,                        //  SDRAM Data bus 16 Bits
        DRAM_ADDR,                      //  SDRAM Address bus 12 Bits
        DRAM_LDQM,                      //  SDRAM Low-byte Data Mask
        DRAM_UDQM,                      //  SDRAM High-byte Data Mask
        DRAM_WE_N,                      //  SDRAM Write Enable
        DRAM_CAS_N,                     //  SDRAM Column Address Strobe
        DRAM_RAS_N,                     //  SDRAM Row Address Strobe
        DRAM_CS_N,                      //  SDRAM Chip Select
        DRAM_BA_0,                      //  SDRAM Bank Address 0
        DRAM_BA_1,                      //  SDRAM Bank Address 0
        DRAM_CLK,                       //  SDRAM Clock
        DRAM_CKE,                       //  SDRAM Clock Enable
        ////////////////////    Flash Interface     ////////////////
        FL_DQ,                          //  FLASH Data bus 8 Bits
        FL_ADDR,                        //  FLASH Address bus 22 Bits
        FL_WE_N,                        //  FLASH Write Enable
        FL_RST_N,                       //  FLASH Reset
        FL_OE_N,                        //  FLASH Output Enable
        FL_CE_N,                        //  FLASH Chip Enable
        ////////////////////    SRAM Interface      ////////////////
        SRAM_DQ,                        //  SRAM Data bus 16 Bits
        SRAM_ADDR,                      //  SRAM Address bus 18 Bits
        SRAM_UB_N,                      //  SRAM High-byte Data Mask
        SRAM_LB_N,                      //  SRAM Low-byte Data Mask
        SRAM_WE_N,                      //  SRAM Write Enable
        SRAM_CE_N,                      //  SRAM Chip Enable
        SRAM_OE_N,                      //  SRAM Output Enable
        ////////////////////    SD_Card Interface   ////////////////
        SD_DAT,                         //  SD Card Data
        SD_DAT3,                        //  SD Card Data 3
        SD_CMD,                         //  SD Card Command Signal
        SD_CLK,                         //  SD Card Clock
        ////////////////////    USB JTAG link   ////////////////////
        TDI,                            // CPLD -> FPGA (data in)
        TCK,                            // CPLD -> FPGA (clk)
        TCS,                            // CPLD -> FPGA (CS)
        TDO,                            // FPGA -> CPLD (data out)
        ////////////////////    I2C     ////////////////////////////
        I2C_SDAT,                       //  I2C Data
        I2C_SCLK,                       //  I2C Clock
        ////////////////////    PS2     ////////////////////////////
        PS2_DAT,                        //  PS2 Data
        PS2_CLK,                        //  PS2 Clock
```

```
67          ////////////////////    VGA     ////////////////////////////
68          VGA_HS,                         //  VGA H_SYNC
69          VGA_VS,                         //  VGA V_SYNC
70          VGA_R,                          //  VGA Red[3:0]
71          VGA_G,                          //  VGA Green[3:0]
72          VGA_B,                          //  VGA Blue[3:0]
73          ////////////////    Audio CODEC     ////////////////////////
74          AUD_ADCLRCK,                    //  Audio CODEC ADC LR Clock
75          AUD_ADCDAT,                     //  Audio CODEC ADC Data
76          AUD_DACLRCK,                    //  Audio CODEC DAC LR Clock
77          AUD_DACDAT,                     //  Audio CODEC DAC Data
78          AUD_BCLK,                       //  Audio CODEC Bit-Stream Clock
79          AUD_XCK,                        //  Audio CODEC Chip Clock
80          ////////////////////    GPIO    ////////////////////////////
81          GPIO_0,                         //  GPIO Connection 0
82          GPIO_1                          //  GPIO Connection 1
83      );
84
85  //////////////////////    Clock Input     //////////////////////
86  input   [1:0]   CLOCK_24;               //  24 MHz
87  input   [1:0]   CLOCK_27;               //  27 MHz
88  input           CLOCK_50;               //  50 MHz
89  input           EXT_CLOCK;              //  External Clock
90  //////////////////////    Push Button     //////////////////////
91  input   [3:0]   KEY;                    //  Pushbutton[3:0]
92  //////////////////////    DPDT Switch     //////////////////////
93  input   [9:0]   SW;                     //  Toggle Switch[9:0]
94  //////////////////////    7-SEG Dispaly   //////////////////////
95  output  [6:0]   HEX0;                   //  Seven Segment Digit 0
96  output  [6:0]   HEX1;                   //  Seven Segment Digit 1
97  output  [6:0]   HEX2;                   //  Seven Segment Digit 2
98  output  [6:0]   HEX3;                   //  Seven Segment Digit 3
99  ////////////////////////////    LED     ////////////////////////////
100 output  [7:0]   LEDG;                   //  LED Green[7:0]
101 output  [9:0]   LEDR;                   //  LED Red[9:0]
102 ////////////////////////////    UART    ////////////////////////////
103 output          UART_TXD;               //  UART Transmitter
104 input           UART_RXD;               //  UART Receiver
105 //////////////////////    SDRAM Interface //////////////////////
106 inout   [15:0]  DRAM_DQ;                //  SDRAM Data bus 16 Bits
107 output  [11:0]  DRAM_ADDR;              //  SDRAM Address bus 12 Bits
108 output          DRAM_LDQM;              //  SDRAM Low-byte Data Mask
109 output          DRAM_UDQM;              //  SDRAM High-byte Data Mask
110 output          DRAM_WE_N;              //  SDRAM Write Enable
111 output          DRAM_CAS_N;             //  SDRAM Column Address Strobe
112 output          DRAM_RAS_N;             //  SDRAM Row Address Strobe
113 output          DRAM_CS_N;              //  SDRAM Chip Select
114 output          DRAM_BA_0;              //  SDRAM Bank Address 0
115 output          DRAM_BA_1;              //  SDRAM Bank Address 0
116 output          DRAM_CLK;               //  SDRAM Clock
117 output          DRAM_CKE;               //  SDRAM Clock Enable
118 //////////////////////    Flash Interface //////////////////////
119 inout   [7:0]   FL_DQ;                  //  FLASH Data bus 8 Bits
120 output  [21:0]  FL_ADDR;                //  FLASH Address bus 22 Bits
121 output          FL_WE_N;                //  FLASH Write Enable
122 output          FL_RST_N;               //  FLASH Reset
123 output          FL_OE_N;                //  FLASH Output Enable
124 output          FL_CE_N;                //  FLASH Chip Enable
125 //////////////////////    SRAM Interface  //////////////////////
126 inout   [15:0]  SRAM_DQ;                //  SRAM Data bus 16 Bits
127 output  [17:0]  SRAM_ADDR;              //  SRAM Address bus 18 Bits
128 output          SRAM_UB_N;              //  SRAM High-byte Data Mask
129 output          SRAM_LB_N;              //  SRAM Low-byte Data Mask
130 output          SRAM_WE_N;              //  SRAM Write Enable
131 output          SRAM_CE_N;              //  SRAM Chip Enable
132 output          SRAM_OE_N;              //  SRAM Output Enable
```

```
133  ////////////////////    SD Card Interface   ////////////////////////
134  inout           SD_DAT;                 //  SD Card Data
135  inout           SD_DAT3;                //  SD Card Data 3
136  inout           SD_CMD;                 //  SD Card Command Signal
137  output          SD_CLK;                 //  SD Card Clock
138  ////////////////////////    I2C     ////////////////////////////////
139  inout           I2C_SDAT;               //  I2C Data
140  output          I2C_SCLK;               //  I2C Clock
141  ////////////////////////    PS2     ////////////////////////////////
142  input           PS2_DAT;                //  PS2 Data
143  input           PS2_CLK;                //  PS2 Clock
144  ////////////////////    USB JTAG link   ////////////////////////////
145  input           TDI;                    // CPLD -> FPGA (data in)
146  input           TCK;                    // CPLD -> FPGA (clk)
147  input           TCS;                    // CPLD -> FPGA (CS)
148  output          TDO;                    // FPGA -> CPLD (data out)
149  ////////////////////////    VGA         ////////////////////////////
150  output          VGA_HS;                 //  VGA H_SYNC
151  output          VGA_VS;                 //  VGA V_SYNC
152  output  [3:0]   VGA_R;                  //  VGA Red[3:0]
153  output  [3:0]   VGA_G;                  //  VGA Green[3:0]
154  output  [3:0]   VGA_B;                  //  VGA Blue[3:0]
155  ////////////////////    Audio CODEC     ////////////////////////////
156  output          AUD_ADCLRCK;            //  Audio CODEC ADC LR Clock
157  input           AUD_ADCDAT;             //  Audio CODEC ADC Data
158  output          AUD_DACLRCK;            //  Audio CODEC DAC LR Clock
159  output          AUD_DACDAT;             //  Audio CODEC DAC Data
160  inout           AUD_BCLK;               //  Audio CODEC Bit-Stream Clock
161  output          AUD_XCK;                //  Audio CODEC Chip Clock
162  ////////////////////////    GPIO    ////////////////////////////////
163  inout   [35:0]  GPIO_0;                 //  GPIO Connection 0
164  inout   [35:0]  GPIO_1;                 //  GPIO Connection 1
165  //////////////////////////////////////////////////////////////////////
166
167  //  All inout port turn to tri-state
168  assign  DRAM_DQ     =   16'hzzzz;
169  assign  FL_DQ       =   8'hzz;
170  assign  SRAM_DQ     =   16'hzzzz;
171  assign  SD_DAT      =   1'bz;
172  assign  I2C_SDAT    =   1'bz;
173  assign  GPIO_0      =   36'hzzzzzzzzz;
174  assign  GPIO_1      =   36'hzzzzzzzzz;
175
176
177  //////////////////////////////////////////////////////////////////////
178  // Ansprechen der Leuchtdioden
179  //////////////////////////////////////////////////////////////////////
180
181  // Counter definieren, der mit 50 Mhz hochzaehlt.
182  reg     [27:0]  Cont;
183  always@(posedge CLOCK_50)       Cont    <=  Cont+1'b1;
184
185  // Counter auf den gruenen Leuchtdioden anzeigen.
186  assign LEDG = Cont[27:20];
187
188  // Schalterstatus auf den roten Leuchtdioden anzeigen.
189  assign LEDR = SW;
190
191  endmodule
```

Der Josephspfennig

Dirk Brühl, Heinrich Haußmann, Michael Kalus

Wenn Joseph zu Christi Geburt einen einzigen Pfennig angelegt hätte, auf welchen Betrag wäre dieser heute bei einer jährlichen Verzinsung von 5% angewachsen?

Kein Mensch weiß heute mehr so genau, was neulich der Anlass war, die Geschichte vom Josephspfennig wieder hervorzuholen — war es der Banken–Crash in diesem Jahr oder das Gerede von einer drohenden Wirtschaftskrise oder dass jemand meinte, man könne doch auch *sein Geld arbeiten lassen*? Nun, wie dem auch sei, was passiert, wenn man die Zinseszinsrechnungen konsequent verfolgt, hat Dirk Brühl vor einigen Jahren mit Hilfe von Forth nachgerechnet. Und das kam so.

Vorgeschichte

Nachdem sein Freund Heinrich abgeschätzt hatte, dass so ungefähr die Masse von 250 Milliarden goldener Erdkugeln als Zinseszinsforderung anstehen würde, wollte er aber den Zahlbetrag in DM auch haben. Die unterschiedliche Rechengenauigkeit der verwendeten Programme und Programmversionen erzeugten aber auch Abweichungen des Endergebnisses. Zuletzt bat er Dirk, eine Rechengenauigkeit von 50 Stellen vor und 15 Stellen nach dem Komma zu programmieren. So fand man den genaueren Endbetrag der *Geldforderung*:

Aus 1 Pfennig im Jahre 0 zu 5% mit Zins und Zinseszins ergab sich im Jahr 2003 der rechnerische Betrag von:

27.679.996.896.051.261.677.068.884.476.135.650.875.110,12 DM

Das Ergebnis zum Mitlesen:

27 Sextilliarden
679 Sextillionen
996 Quintilliarden
896 Quintillionen
051 Quadrilliarden
261 Quadrillionen
677 Trilliarden
068 Trillionen
884 Billarden
476 Billionen
135 Milliarden
650 Millionen
875 Tausend
110,12 DM

Und es folgte eine Kopfrechenübung: Was wäre das Ergebnis nach 2000 Jahren mit einfachem Zins ohne den Zinseszinseffekt?

0,01 DM x 0,05 Zinssatz
= 0,0005 Zinsbetrag im Jahr x 2000 Jahre
= 1,00 DM

(Da die Geschichte bereits im Jahre 1989 von den beiden aufgeschrieben worden war, und 2003 und dann heute noch mal Thema wurde, weiß heute natürlich jeder, dass das Ergebnis durch 1,98853 zu teilen ist, will man statt DM den Betrag in Euro haben. Die Umrechnung verkneifen wir uns hier und sehen einer entsprechenden Kritik gelassen entgegen. ;-)

Zur Programmgeschichte

Dirk hat für diesen Beitrag das Programm herausgesucht und etwas aufbereitet. Das Programm hat mit FPC gleich funktioniert, und nachdem die Definitionen für `2+`, `2-` und `Printing` ergänzt worden waren, lief es auch mit Win32Forth. Es wurde noch ein Abbruch in die Schleifen eingebaut, damit man für den Fall, dass 2000 Jahre zu lange sind, die Anzeige beenden kann.

Die Multiplikation wurde durch Additionen ersetzt, das war das Einfachste. Und nachdem erst einmal gezeigt ist, dass so lange Zahlen ganz einfach sind, dürfte es auch kein riesiges Problem sein, die übrigen Grundrechenarten zu machen, oder? Vielleicht eine Herausforderung für die VD–Leser?

Heinrich hat damals einen *Pfennig* als Einheit gewählt, weil sich das gut anhörte. Das macht natürlich banktechnisch keinen Sinn. Für einen Pfennig gibt es IMO keine Zinsen. Deshalb wurden 15 Kommastellen angehängt, damit am Anfang wenigstens etwas dabei herausspringt. Mit den 15 Kommastellen ist das Ergebnis damals in Umlauf gebracht worden, und niemand hat sich bis jetzt dran gestört.

Das Programm

```
1  \ Josephspfennig  DB 1998-12-06
2
3  \ Anleitung und Text-Auswahl am 8.3.2002 hinzugefuegt,
4  \ Key?-leave, Vornullen-Unterdrueckung, Mark/Pfennig, Values und Kommentare ergaenzt am 29.12.2008
5
6
7  decimal
8
```

```
 9  65 Constant Stellen             \ Anzahl der maximalen Stellen insgesamt
10  15 Value    Kommastellen        \ Anzahl der maximalen Nachkommastellen
11   1 Value    Starteinlage        \ Start-Einlage
12   5 Value    Prozent             \ Prozent Zinseszins
13
14  \ Der Microsoft-Calculator zeigt bis zu 33 Kommastellen an, solange die Zahl kleiner als 1 ist,
15  \ eine 34. Stelle wird zum Aufrunden berechnet. Danach werden maximal 32 Stellen angezeigt.
16  \ Zum Vergleichen kann hier die Anzahl der Kommastellen auf 34 geaendert werden
17  \ mit "34 to Kommastellen", allerdings rundet dieses Programm hier am Ende nicht auf.
18  \ Dieses Aufrunden oder nicht wurde gelegentlich bei Bank-Software verwendet,
19  \ um die Differenz auf das Konto des Programmierers abzuzweigen.
20  \ Beim Microsoft-Calculator koennen maximal 32 stellige Zahlen eingegeben werden,
21  \ weitere Eingaben werden nicht angenommen.
22  \ Fazit: Berechnungsergebnisse mit Kommastellen koennen nicht bei allen (Taschen-)Rechnern gleich sein.
23
24  vocabulary Josephspfennig
25  Josephspfennig definitions
26
27  Variable Wert Stellen allot     \ Ablage des errechneten Wertes
28  Variable % Stellen allot        \ Ablage des Prozentwertes
29  Variable Vornullen              \ Steuerung der Vornullenunterdrueckung
30  Variable Printer                \ Druckersteuerung
31
32
33  \ Ergaenzungen
34
35  : 2+ 2 + ;  : 2- 2 - ;  \ eingefuegt fuer Win32Forth
36  Variable PRINTING       \ entfernen fuer Druckerausgabe bei FPC
37  \ : ascii     postpone [char] ; immediate \ einfuegen fuer gforth.
38
39  \ Ende Ergaenzungen
40
41  Variable Pfennig
42  Pfennig on                      \ Pfennig off schaltet um auf Mark statt Pfennig als Start-Einlage
43
44  : Waehrung     ( -- )           \ Mark oder Pfennig
45     Pfennig @ if 1- else 1+ then
46  ;
47
48  : "Waehrung     ( -- )           \ Mark oder Pfennig
49     Pfennig @ if ." Pfennig" else ." Mark" then
50  ;
51
52  : Wert>%       ( -- )           \ Uebertrag des letzten Wertes zur %-Berechnung
53     0 % ! Wert % 2+ Stellen 2- cmove
54  ;
55
56  : .Wert        ( -- )           \ Ausgabe des ermittelten Wertes
57     Vornullen on                 \ Vornullen-Unterdrueckung ein
58     Stellen Kommastellen -
59         dup 0 do dup I 2 + < if Vornullen off then
60                  Wert I + c@ 15 and Vornullen @       \ Vorkommastellen
61                  if   ?dup if   0 .r  Vornullen off
62                            else space
63                            then
64                  else 0 .r
65                  then loop drop
66     ascii , emit                                      \ Komma
67     Stellen dup Kommastellen -
68               do Wert I + c@ 15 and 0 .r loop         \ Nachkommastellen
69  ;
70
71  : Clear        ( -- )           \ Nullsetzen aller Werte
72     Wert Stellen erase
73        % Stellen erase
74  ;
```

```
75
76  : setzen          ( Wert -- )      \ Setze Waehrung-Startwert ( 1-9 )
77    Wert Stellen + Kommastellen Waehrung - c! ;
78
79  : Add ( Wert1 Wert2 Carry -- Wert3 Carry )  \ addiere zwei Werte mit Carry
80    + + dup 9 > if 10 - 1 else 0 then
81  ;
82
83  : +1%          ( -- )           \ Addiere %-Wert zu bisherigem Wert
84    0 0 Stellen 1- do Wert i + c@ 15 and % i + c@ 15 and add
85                      swap Wert i + c!  -1 +loop drop
86  ;
87
88  : +%          ( % -- )           \ Addiere % zu bisherigem Wert
89    Wert>% 0 do +1% loop
90  ;
91
92
93  : .Z-Ziffern   ( -- )           \ Zeige Zehnerstellen an
94    cr 7 Spaces
95    Stellen Kommastellen - 0 do Stellen Kommastellen  - i -
96                                10 /  0 .r loop
97    ascii , emit
98    Stellen dup Kommastellen - do i Stellen Kommastellen  - - 1+
99                                  10 / 0 .r loop
100 ;
101
102 : .E-Ziffern   ( -- )           \ Zeige Einerstellen an
103   cr 7 Spaces
104   Stellen Kommastellen - 0 do Stellen Kommastellen  - i -
105                               10 /mod drop  0 .r loop
106   ascii , emit
107   Stellen dup Kommastellen - do i Stellen Kommastellen  - - 1+
108                                 10 /mod drop  0 .r loop
109 ;
110
111 : Jahr         ( Jahr -- )      \ Zeige Jahr und Zinseszins-Ergebnis an
112   cr 5 .r 2 Spaces .Wert
113 ;
114
115 : >Printer     ( Jahr -- )      \ Ausgabe auf Drucker (nur mit FPC)
116   Printer @ if printing on then
117 ;
118
119 : All          ( Jahre -- )     \ Ergebnis fuer alle Jahre bis "Jahre"
120   >Printer
121   Clear Starteinlage setzen 1+
122   cr 7 Spaces
123   ." Josephspfennig - " Starteinlage . "Waehrung ."  im Jahre 0, mit " Prozent 0 .R ." % Zinseszins"
124   cr 7 Spaces
125   ." dargestellt fuer " dup 1- . ." Jahre" cr
126   .Z-Ziffern .E-Ziffern cr
127   0 do key? if leave else i Jahr Prozent +% then loop
128   cr .E-Ziffern .Z-Ziffern cr
129   Printer @ if 12 emit Printing off Printer off then
130 ;
131
132 : Years        ( Jahre -- )     \ Ergebnis fuer jedes hundertste Jahr bis "Jahre"
133   >Printer
134   Clear Starteinlage setzen dup 1+
135   cr 7 Spaces
136   ." Josephspfennig - " Starteinlage . "Waehrung ."  im Jahre 0, mit " Prozent 0 .R ." % Zinseszins"
137   cr 7 Spaces
138   ." dargestellt fuer jedes hundertste Jahr bis " dup 1- . cr
139   .Z-Ziffern .E-Ziffern cr
140    1- 0 do i 100 /mod drop 0= if i Jahr then key? if drop 0 leave else Prozent +% then loop
```

```
141     ?dup if Jahr then
142     cr .E-Ziffern .Z-Ziffern cr
143     Printer @ if 12 emit printing off  Printer off then
144   ;
145
146   : Jahrevonbis  ( von bis -- )
147     >Printer
148     Clear Starteinlage setzen 1+
149     cr 7 Spaces
150     ." Josephspfennig - " Starteinlage . "Waehrung ."  im Jahre 0, mit " Prozent 0 .R ." % Zinseszins"
151     cr 7 Spaces
152     ." dargestellt vom Jahr " over . ." bis zum Jahr " dup 1- . cr
153     .Z-Ziffern .E-Ziffern cr
154     dup
155     0 do key? if leave else 2dup 1+ i rot rot within if i Jahr then  Prozent +% then loop
156     cr .E-Ziffern .Z-Ziffern cr 2drop
157     Printer @ if 12 emit printing off Printer off then
158   ;
159
160   Variable Auswahl
161   Auswahl on
162
163   : Alle 1 Auswahl ! ;
164   : Jahre ;
165   : von 2 Auswahl ! ;
166   : bis Auswahl @ true = if 3 Auswahl ! then ;
167
168   : anzeigen     ( -- )
169     Auswahl @ 1 = if ." :" cr All Auswahl on then
170     Auswahl @ 2 = if ." :" cr Jahrevonbis Auswahl on then
171     Auswahl @ 3 = if ." :" cr Years Auswahl on  then
172   ;
173
174   : A cr All ;
175   : J cr Years ;
176   : VB cr JahreVonBis ;
177   : P Printer on ;
178
179   : Anleitung    ( -- )
180     cr cr
181     ." Das Programm zur Berechnung des Josephspfennig ist betriebsbereit !"
182     cr cr
183     ." Es gibt drei Anzeigemoeglichkeiten: "
184     cr cr
185     ." Alle Jahre bis 2010 anzeigen "
186     cr
187     ." Jahre von 2000 bis 2010 anzeigen "
188     cr
189     ." Jahre bis 2010 anzeigen "
190     cr cr
191     ." Der Text muss wie angezeigt eingegeben werden, die Jahreszahlen"
192     cr
193     ." koennen beliebig sein, danach Eingabetaste betaetigen."
194     cr  cr
195     ." Alternative:"
196     cr  cr
197     ." 2010 A - zeigt alle Jahre bis 2010 an,"
198     cr
199     ." 2010 J - zeigt die Jahre von 0 bis 2010 in hunderter Schritten an,"
200     cr
201     ." 2000 2010 VB - zeigt die Jahre von 2000 bis 2010 an."
202     cr cr
203     ." P - gibt Werte auf Drucker aus (nur mit FPC), z.B.: "
204     cr
205     ." P 2000 2010 VB - gibt die Jahre von 2000 bis 2010 auf Drucker aus!"
206     cr
```

```
207      ." ? - zeigt die Anleitung an . Beenden des Programms: Bye"
208   ;
209
210   : ? Anleitung ;
211
212   \ Folgendes bitte auskommentieren fuer gforth:
213   cls Anleitung
214
215   \S
216
217     Mindestens vier Kommastellen sind noetig, um bei einem Pfennig Einlage Zinsen zu bekommen,
218     das Ergebnis ist etwa 60% vom Ergebnis mit 15 Kommastellen.
219     Mit 6 Kommastellen werden 99,5% des Ergebnisses der Berechnung mit 15 Kommastellen erreicht.
220     Mit 1 Mark als Starteinlage werden nur 2 Kommastellen benoetigt - das Ergebnis ist 171 mal so hoch.
221     Test mit:
222               2 to Kommastellen Pfennig off 2003 j <Enter>
223
```

Anhang: Making of ...

Aus unserer Email–Korrespondenz in 12/2008;

Dirk Brühl schrieb:

> ... (Die Geschichte zum und das Programm) ...
>
> Soweit, so gut. Ich hoffe, dass Du meine Zeilen ein wenig aufbereiten kannst. Gestartet wird das Programm in FPC mit
>
> `fload Joseph.seq <Enter>`
>
> Mit dem Start wird eine Beschreibung angezeigt. Das Programm läuft unverändert sowohl mit FPC als auch mit Win32Forth. Ich habe es mehrmals getestet, würde mich freuen, wenn Du es auch mal ausprobierst.
>
> Ich habe das Programm wahrscheinlich vor fast genau zehn Jahren und drei Wochen geschrieben, an einem Sonntagnachmittag, anlässlich eines Besuches von Heinrich. Deshalb habe ich die ganze Sache so einfach wie möglich gemacht, damit ich nicht lange rumzuprobieren brauche und mir keine komplizierten Gedanken machen muss. Sicherlich gibt es elegantere Arten, Arithmetik zu programmieren, aber vielleicht zeigt dieses Beispiel auch, dass man kompliziert oder aufwändig scheinende Aufgaben mit einfachen Schritten in Forth lösen kann. Das Programm enthält nur einen Teil einer Grundrechenart, aber es lässt sich sicherlich einfach ergänzen, und es läuft sowohl mit 8–Bit– als auch mit 64–Bit–Rechnern, solange ein Character–`C@` mit 8 Bit arbeitet.
>
> Viel Spaß beim Ausprobieren, lasst mich wissen, was draus geworden ist!
>
> Viele herzliche Grüße aus dem stürmischen Pennsylvanien, Dirk

Michael Kalus:

> Gerne bereite ich das für die VD auf. Dir und Heinrich meinen herzlichen Dank für Eure Unterstützung und fürs Mitteilen Eurer Überlegungen.
>
> Viele Grüße nach Pennsylvanien!

Link

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

Forth von der Pike auf — Teil 11

Ron Minke

Die mit freundlicher Genehmigung der HCC–Forth–gebruikersgroep in der Vierten Dimension in einer Übersetzung von Fred Behringer wiedergegebene achtteilige Artikelserie erschien ursprünglich in den Jahren 2004 und 2005 in der Zeitschrift *Vijgeblaadje* unserer niederländischen Forth–Freunde. Der Autor arbeitet fleißig an seinem Projekt weiter: Im Vijgeblaadje waren Teil 9 und 10 der Serie (deutsche Übersetzung in den VD-Heften 3-4/2007 und 2/2008) erschienen. Auch die heutige Übersetzung aus dem Niederländischen (aus dem Vijgeblaadje 70) stammt von Fred Behringer.

Hier kommt nun Teil 11 der Wiedergabe des Versuchs, ein AVR–Forth–System mit der Voraussetzung *from scratch* zu erstellen.

Big Oops

In der vorigen Folge von *Forth von der Pike auf* hatten wir zwei früher beschlossene Festlegungen (aus Teil 2) wieder zurückgenommen. Einige Zeit, nachdem das Vijgeblaadje in Druck gegangen war, stellte sich beim Experimentieren eine sehr unangenehme Eigenschaft der gerade angebrachten Änderung heraus. Was war passiert? Nach dem Umdrehen der Reihenfolge der Bytes auf dem Stack hatten wir die in Abbildung 1 gezeigte Situation (vergleiche das in Teil 10 Gesagte).

In Bezug auf das Forth–Wort AND beispielsweise ist nichts auszusetzen. Zwei 16–Bit–Werte auf dem Stack werden verarbeitet und ein 16–Bit–Wert wird zurückgegeben. Ob das obere Byte eines Wertes oder das untere Byte eines Wertes nun zuoberst oder zuunterst auf dem Stack steht, tut eigentlich nichts zur Sache. Der betreffende Maschinencode–Teil ist genau so codiert, dass das richtige Ergebnis herauskommt. Für High–Level–Worte gilt dasselbe.

Was läuft da schief?

Es läuft eigentlich gar nichts schief. Es ist die Interpretation der Werte auf dem Stack, die dafür verantwortlich ist, dass es nicht so läuft, wie wir das wollen. Schauen wir uns noch einmal zwei Werte auf dem Stack genauer an:

	0		0x100
	1	n2 oberes Byte	0x0FF
	2	n2 unteres Byte	0x0FE
	3	n1 oberes Byte	0x0FD
SP →	4	n1 unteres Byte	0x0FC
	5		0x0FB

Wenn wir diese zwei Werte nicht als zwei einzelne Worte betrachten, sondern als ein einziges *Doppelwort*, dann bleibt das Schema dasselbe, aber die Interpretation ändert sich. Jetzt wird es wichtig zu wissen, welches Wort eines Doppelwortes auf dem Stack zuoberst liegt. Wenn wir auf unserem Weg der *umgedrehten Version* weitergehen, dann erhalten wir als Reihenfolge auf dem Stack:

	0			0x100
	1 oberes Byte	oberes Wort	R19	0x0FF
	2 unteres Byte	oberes Wort	R18	0x0FE
	3 oberes Byte	unteres Wort	R17	0x0FD
SP →	4 unteres Byte	unteres Wort	R16	0x0FC
	5			0x0FB

An dieser Reihenfolge ist nichts verkehrt. Die Maschinencode–Programmteile, die hiermit arbeiten, holen von selbst die richtige Reihenfolge ab und legen sie auch richtig wieder zurück. Aber

Was tun wir mit einer *double*–Zahl?

Hier zeigt sich das eigentliche Problem. Wenn wir auf dem Stack ein *Single*–Wort haben und wenn wir das zu einem *Double*–Wort umformen wollen, dann verwenden wir das Wort `S>D`. In der einfachen Version setzen wir

AND (n2 n1 -- n3)

	Input			Output		tmp. AVR-Reg.	Daten-Stack-Adr.
	0			0		—	0x100
	1	n2 oberes Byte		1	n3 oberes Byte	R19	0x0FF
	2	n2 unteres Byte	SP →	2	n3 unteres Byte	R18	0x0FE
	3	n1 oberes Byte		3		R17	0x0FD
SP →	4	n1 unteres Byte		4		R16	0x0FC
	5			5			0x0FB

Abbildung 1: Verhalten von `AND` bei umgedrehter Reihenfolge der Elemente auf dem Stack

eine Null davor. Das lässt sich am good–old FIG–Forth–System gut ablesen an der Definition:

```
: U.  0 D.  ;
```

Das Anfügen einer Extra–Null sorgt für die Erweiterung einer Single–Zahl zu einer Double–Zahl. Aber in dem oben stehenden Stack–Auszug kommt das nicht gut heraus. Die Extra–Null steht auf dem Platz des unteren Wortes, derweil wir es als das obere Wort haben wollen.

Einzige Lösung: Die Reihenfolge wieder zurückdrehen. Damit sind wir dann wieder bei den urspünglichen Festlegungen 2 und 3 angelangt.

Und wie weiter?

Um die Sache nicht noch komplizierter zu machen, lassen wir die Registerbelegungen paarweise so, wie sie sind: R17:R16 gibt weiterhin das oberste Wort auf dem Stack an. Damit bekommen wir:

	0			0x100
	1 unteres Byte	unteres Wort	R18	0x0FF
	2 oberes Byte	unteres Wort	R19	0x0FE
	3 unteres Byte	oberes Wort	R16	0x0FD
SP →	4 oberes Byte	oberes Wort	R17	0x0FC
	5			0x0FB

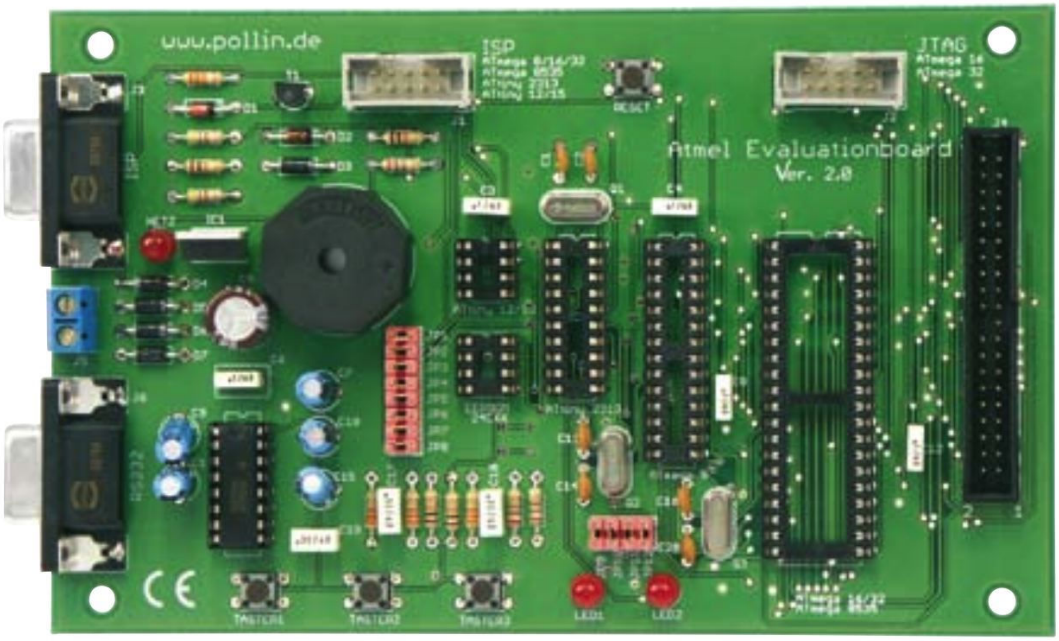
Die ursprünglichen Festlegungen 2 und 3 scheinen eben doch die richtige Wahl gewesen zu sein:

Das untere Byte eines 16–Bit–Wortes wird zuerst auf den Stack gelegt (hier R16), und darunter dann das obere Byte (hier R17).

Der Forth–Datenstack–Pointer SP zeigt bei einem 16–Bit–Wort auf das obere Byte.

Alle anderen Anpassungen aus Teil 10 (wie beispielsweise die Anordnung der Buchstaben in einer Text–Passage im Flash–Speicher) bleiben dieselben.

Reichlich viel Arbeit, dafür aber ein weiterer Schritt in Richtung auf das System *Forth von der Pike auf* zu!



Das Low-Cost Atmel–Evaluationsboard von Pollin (Quelle: `www.pollin.de`)

g4 — Ein Forth–nach–Assembler–Converter

Michael Kalus

Bei der Arbeit mit dem amforth von Matthias Trute entstand der Wunsch, den in Forth verfassten Code auch als Assemblerquelle zu haben. So brauchen ausgetestete Forth–Programmteile nicht mehr über die serielle Schnittstelle in das Board nachgeladen zu werden, sondern können mit dem Kern des amforths zusammen assembliert und dann in den Prozessor hochgeladen werden. Das Tool heißt[1] g4 und ist freie Software und *work in progress* — Fehlerhinweise bitte an mich. Das g4 kann vom Server der FG geladen werden, siehe Links weiter unten.

Wie funktioniert das g4?

Das g4 ist ein amforth–Code–Interpreter und in gforth verfasst. Es wird in gforth geladen und nimmt dann entweder Kommandozeilen oder Textdateien an. Ein Beispiel zeigt, wie es geht. Die Zeile

```
: zoo ." hallo" s" tach" ;
```

wird dann expandiert zu:

```
VE_ZOO:
    .dw $FF03
    .db "zoo",0
    .dw VE_HEAD
    .set VE_HEAD = VE_ZOO
XT_ZOO:
    .dw DO_COLON
PFA_ZOO:
    .dw XT_SLITERAL
    .dw $5
    .db "hallo"
    .dw XT_ITYPE
    .dw XT_SLITERAL
    .dw $4
    .db "tach"
    .dw XT_EXIT
```

In diesem Beispiel ist zu sehen, dass Header und Struktur der amforth–Quellen von dem g4–Interpreter nachgebildet werden. Und sofern alles geklappt hat, kann die resultierende Datei in die eigene amforth–Quelle als Applikation eingebunden werden.

Features des g4 Interpreters

Forth–Definitionen werden in amforth–Quellen expandiert (colon definitions). Das geschieht mit Hilfe eines polyForth–ähnlichen Compilers. Einfache amforth–Worte werden z. B. so konvertiert:

```
_: swap  _cr _."     .dw XT_SWAP "  _;
```

Defining words wie : (colon) VARIABLE CONSTANT USER legen zunächst im Wörterbuch des gforth neue Definitionen an, die dann zur Laufzeit ihren Namen als Macro für den Assembler ergeben. Die defered words RDEFER und EDEFER werden ebenso unterstützt wie CREATE ... DOES>–Konstrukte. Dabei können auch , (comma) ALLOT und CELLS verwendet werden.

In Definitionen funktionieren auch Forthworte wie ." s" und POSTPONE inzwischen. Die Kontrollstrukturen wie DO LOOP oder IF ELSE THEN und BEGIN UNTIL werden in Label aufgelöst, die der amforth–Quellen–Syntax entsprechen.

Da amforth ab der Version 3.1 Strings und Namen im Wörterbuch des Flash mit ganzen 16–Bit–Längeninformation ablegt, statt nur mit 8 Bit wie bis dahin, kann nun zwischen beiden Versionen mit einem Schalter ausgewählt werden.

Um auch Steueranweisungen aus der Forthquelle heraus an den Assembler schicken zu können, wurde ein Sonderbefehl eingefügt. Mit >>> lassen sich nun ganze Textzeilen schicken; z. B.:

```
>>> .set pc = pc + $10
```

Um Adress–Token in den amforth–Quellen angeben zu können, wurde der Befehl _lit: geschaffen.

Ist zum Beispiel irgendwo im amforth eine Adresse definiert, kann diese damit auch im Forth definiert und benutzt werden. Beispiel: Registeradressen wie Timer.

```
_lit: TIMSK1
_lit: TIMSK2

: OSCCAL_CALIBRATE ( -- )
    \ Make sure all clock division
    \ is turned off (8MHz RC clock)
    1 7 lshift 61 c!
    0          61 c!
    \ Disable timer interrupts
    0 TIMSK1 c!
    0 TIMSK2 c!

(usw) ;
```

wird damit passend übersetzt. TIMSK1 expandiert dabei zu

```
...

    .dw XT_DOLITERAL
    .dw TIMSK1

...
```

Auch Forth–Code folgender Art wird verarbeitet:

[1] g4 ist benannt in Anlehnung an den m4–Macro–Prozessor, der aber für diesen Zweck unbrauchbar ist. So war es einfacher, einen eigenen Konverter in gforth zu verfassen, eben den g4.

```
variable x1  10 allot
create y2    10 cells allot
```

Dabei werden die Variablen angelegt und die Assembleranweisung

```
; allot ram
.set heap = heap + nn
```

wird generiert, wobei *nn* die passende Anzahl Bytes angibt.

Um innerhalb einer Forthdefinition auch kalkulieren zu können, wird `[ ]` wie gewöhnlich benutzt.

```
hex  : test  ... [ 11 22 + ] literal ...  ;
```

erzeugt dann

```
...
    .dw XT_DOLITERAL
    .dw $33
...
```

Einschränkung dabei ist, dass `[ ]` in g4 auf eine Zeile Forth–Code begrenzt ist.

Eine Besonderheit sind immediate words. Dazu muss die Anweisung `g4-immediate-on` vor der Forth–Definition angegeben werden, statt wie sonst üblich dahinter.

Am Ende prüft g4, ob irgendetwas auf dem Stack liegen geblieben ist, denn falls das der Fall ist, ist was schiefgegangen bei der Erzeugung der Assemblerquelle. OK wäre so etwas:

```
; Items on stack: <0> mk 2008.11.16 21:40:35
```

Wie wird g4 aufgerufen?

So gehts mit Apple OSX:

```
PowerBook:~/michael$  gforth g4.fs
```

Oder das g4 gleich in die zu übersetzende Datei einbinden:

```
PowerBook:~/michael$ gforth [datei.fs] >[datei.asm] ,
```

wobei [datei.fs] die Forth–Quelle ist, die mit einem darin untergebrachten `include g4` befähigt wird, das Ergebnis sogleich in die [datei.asm] auszugeben — das ist sehr komfortabel.

Mit `gforth` auf Windows XP läuft das g4 ebenfalls. Dort läuft `gforth` in der Unix Shell `sh.exe`. Bitte darauf achten das der Punkt und Schrägstrich auf der Kommandozeile mit angegeben werden und die Dateien in den gleichen Verzeichnissen liegen.

```
sh-3.00$ ./gforth [datei.fs] >[datei.asm]
```

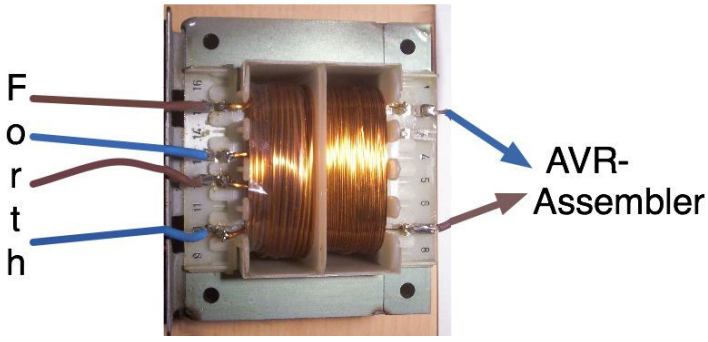
Nutzanwendungen bisher

Bisher habe ich nur eigene `amforth`–Quellen in Assembler–Quellen konvertiert, und so in mein `amforth` 2.9 eingebunden. Es waren dies der Forth–Assembler von Lubos Pekny (`assembler.asm`) , eine Reihe Dump–Utilities (`dump.asm`, `idump.asm`, `edump.asm`), alle Zahlenausgabeworte in der doppeltgenauen Form sowie `key` und `emit` in der pollenden Form. Bisher bin ich mit den Ergebnissen zufrieden.

Ulrich Hoffmann und Matthias Trute gebührt mein Dank für die ermunternden Worte und Hinweise. Ich hoffe, es ist dem einen oder anderen ebenfalls ein nützliches Tool auf dem Wege zu seiner Applikation.

Quellen

```
http://www.forth-ev.de/trac/browser/g4
http://amforth.sourceforge.net/
```



g4 transformiert High-Level–Forth in amForth–Assembler–Text

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) – 79 85 57
bernd.paysan@gmx.de
Treffen: Jeden 4. Mittwoch im Monat um 19:00, im Chilli Asia Dachauer Str. 151, 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

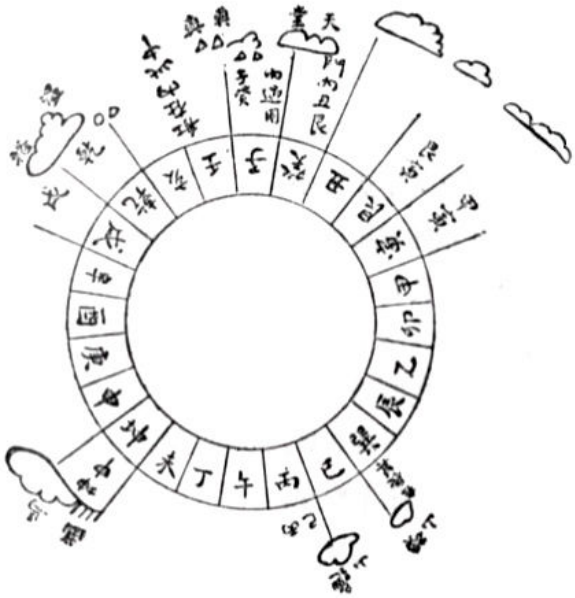
Carsten Strotmann
microcontrollerverleih@forth-ev.de
mcv@forth-ev.de

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 2009

vom 27. bis 29. März 2009
in Wilminks Parkhotel (`www.wilminks-parkhotel.de`)
Wettringer Straße 46, 48485 Neuenkirchen (bei Rheine)

Programm

Donnerstag, 26.03.2009
14:00 Uhr Forth–Schulung

Freitag, 27.03.2009
9:00 Uhr (Rad-)Wanderung
Forth–200x–Standard–Treffen
Forth–Schulung
14:00 Uhr Beginn der Forth–Tagung

Samstag, 28.03.2009
14:00 Uhr Ausflug (z. B. Bagno in Steinfurt)

Sonntag, 29.03.2009
09:00 Uhr Mitgliederversammlung
14:00 Uhr Ende der Tagung

Anreise siehe: `http://www.wilminks-parkhotel.de/de/anfahrt_wegbeschreibung/`

Über das Parkhotel Wilminks:

„Willkommen in Neuenkirchen! Mit diesen Worten werden Sie in rund 18 gleichnamigen Orten in ganz Deutschland begrüßt. Verwechslungen sind da nicht ausgeschlossen. Doch ein Neuenkirchen, gelegen in der ländlichen Idylle Westfalens, ist unverwechselbar. Es hat nämlich etwas ganz Besonderes: ein familiengeführtes Hotel mit so etwas wie »Eigenleben«. Keine Sorge, damit meinen wir nicht etwa Holzwürmer – sondern eine bewegende Atmosphäre, die es schafft, jeden zu überraschen, zu begeistern und das Gastsein neu kennen zu lernen...

Unser Hotel ist so vielseitig wie unsere Gäste. So hat zum Beispiel jedes Zimmer seinen ganz eigenen Charakter. Mal klassisch, mal modern, mal romantisch, mal anders. “

Ideal also auch für unsere Forth–Tagung mit ihren Vorträgen, Workshops und Nachtsitzungen.

Quellen: `http://www.wilminks-parkhotel.de/`