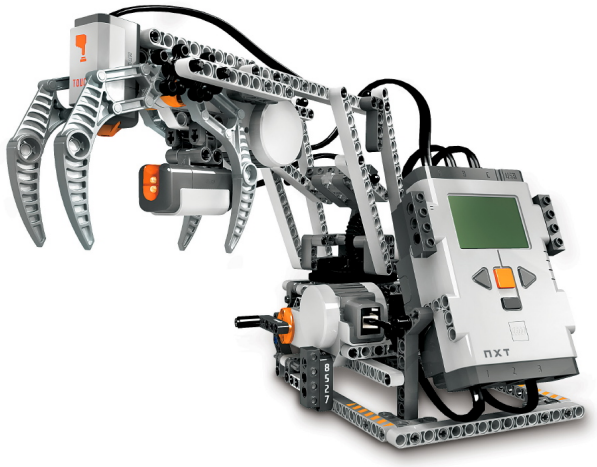
Vierte Dimension

1/2007

Das Forth–Magazin

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



In dieser Ausgabe:

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische Informatik

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

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

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

LEGO RCX–Verleih

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

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

Inhalt:

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

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

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

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

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

KIMA Echtzeitsysteme GmbH

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

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

FORTecH Software
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

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

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

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

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00€ + Porto u. Verpackung

Manuskripte und Rechte

Berücksichtigt werden alle eingesandten Manuskripte. Leserbriefe können ohne Rücksprache wiedergegeben werden. Für die mit dem Namen des Verfassers gekennzeichneten Beiträge übernimmt die Redaktion lediglich die presserechtliche Verantwortung. Die in diesem Magazin veröffentlichten Beiträge sind urheberrechtlich geschützt. Übersetzung, Vervielfältigung, sowie Speicherung auf beliebigen Medien, ganz oder auszugsweise nur mit genauer Quellenangabe erlaubt. Die eingereichten Beiträge müssen frei von Ansprüchen Dritter sein. Veröffentlichte Programme gehen — soweit nichts anderes vermerkt ist — in die Public Domain über. Für Text, Schaltbilder oder Aufbauskizzen, die zum Nichtfunktionieren oder eventuellem Schadhaftwerden von Bauelementen führen, kann keine Haftung übernommen werden. Sämtliche Veröffentlichungen erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes. Warennamen werden ohne Gewährleistung einer freien Verwendung benutzt.

Liebe Leser,

ich begrüße Euch zur ersten Vierten Dimension des **23.** Jahrgangs!

Immer wieder, wenn ich von Forth berichte, werde ich gefragt, ob es *das* denn tatsächlich noch gäbe und ob noch irgendwer damit arbeiten würde. Und ja — ich kann versichern, dass die Forth–Gemeinde immer noch aktiv ist und sich immer noch spannende Dinge entwickeln. Klar, Forth liegt nicht im Mainstream und wir müssen uns auch darum kümmern, junge Leute von Forth zu begeistern. Aber, unsere Aktivitäten unter anderem auf dem Linuxtag sind ja auch nicht ohne Wirkung: Und so möchte ich recht herzlich die neuen Mitglieder der Forth–Gesellschaft begrüßen. Ich hoffe, das vorliegende Heft enthält auch für Euch interessante und nützliche Informationen.

"Fortran is Dead! Long live Forth!", Julian V. Noble

Mit Trauer mussten wir im März erfahren, dass Julian V. Noble, Professor für Physik an der Universität von Virginia/USA, verstorben ist. Julian hat sich ganz besonders für die Verbreitung von Forth insbesondere im Bereich der Numerik eingesetzt. Sein FORmular TRAnslator übersetzt Formeln nach Forth und erlaubt so das bequeme Formulieren numerischer Algorithmen. Auch Julians Buch *Scientific FORTH* widmet sich diesem Thema. Noch in diesem Jahr kommentierte James Hague auf dem Web–Log Lambda-The-Ultimate Julians Artikel *Finite State Machines in Forth*:

> No matter what your opinion of Forth, this is an enticing paper. J.V. Noble takes an interesting problem from rough implementation through an elegant solution which takes advantage of the compile-time features of Forth. There aren't many easily accessible papers that show how beautiful Forth can be, so this is worth taking some time to understand, especially if you've a fondness for stack-based languages.

Ich habe Julian V. Noble nie persönlich kennen lernen können, aber seine Arbeiten und seine Artikel in comp.lang.forth habe ich immer wieder gerne gelesen. Möge Julian in Frieden ruhen. Ich werde ihn sicher in guter Erinnerung behalten.

Zusammen mit dieser Vierten Dimension erscheint das Sonderheft AVR. In ihm haben wir Nützliches über Forth und Atmel–AVR–Prozessoren zusammengetragen.

Enden möchte ich mit einem Hilferuf. Wir brauchen Artikel, Kurzmeldungen, Leserbriefe für die kommenden Ausgaben Vierte Dimension. Themen gibt es genug. Und: Die Redaktion beißt nicht!

Ulrich Hoffmann

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

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

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

Zur Meldung von Ulrich Hoffmann über den Propeller–Chip (VD–Heft 4/2006)

... ganz einfach, Ulli, war das bei den Transputern: Hardware und Software gingen ineinander über. Jeder Transputer (zumindest der T800) konnte mit vier weiteren Transputern direkt verbunden werden. Pro Verbindung in beiden Richtungen je ein Kanal. Zweidraht. Für die Zeitzuteilung (auf zwei Prioritätsebenen) sorgte die Hardware (Timeslicing mit Bereitschaftsmeldungen). Übertragen wurden (Occam–II–)Prozesse. Die Prozesse konnten auch innerhalb ein und desselben Transputers (quasiparallel) arbeiten und übertragen werden. Den „Prozessen“ war es egal, ob sie in ein und demselben Transputer oder über zwei oder mehrere Transputer liefen. Eines der Links war für die Anbindung an den Host–Computer vorgesehen. Rolf Schöne hat erst kürzlich auf der Nostalgie–Computer–Ausstellung in München über die Transputer–Zeiten vorgetragen und kann mich in dem einen oder anderen Punkt sicher verbessern.

Die auf den Prozessgedanken abgestimmte Sprache Occam war eine Marotte von INMOS (1988). Selbstverständlich konnte man auch auf Assembler–Ebene arbeiten. Eine PEEK– und eine POKE–Operation luden direkt dazu ein, „Forth von der Pike auf“ hochzuziehen. Und die Meldung, dass der oder jene Maschinenbefehle für den Anwender keinen Sinn habe und im Occam–Compiler daher nicht zugelassen werde, gab (mir) den Rest: Das Ergebnis liegt immer noch auf dem amerikanischen FTP–Server taygeta.com (`ftp://taygeta.com/pub/forth/compilers/native /dos/transputer/` ...) . Ich habe es damals (ab 1993) großspurig F–TP 1.00 genannt. Große Erwartungen — Great Expectations!. Dass keiner auf den Gedanken kam, damit etwas zu machen, hat mich enttäuscht, zeitweise sogar an Forth zweifeln lassen. Dass in gewissen Abständen in de.comp.lang.forth immer wieder mal einer auftaucht, der nach einem Forth–System für Transputer sucht und „keines finden kann“, macht die Situation, in welcher man sich mit Forth befindet, nur noch absurder. Übrigens kann man sich in Sachen Transputer–Forth auch zu Marcel Hendrix durchgoogeln. Und bei taygeta.com liegt auch Turbo–Forth, ein weiteres Kapitel zum Thema „Forth und die mangelnde Akzeptanz“. Dass jeder einzelne Transputer einen ausgewachsenen Gleitkomma–Teil hatte, bleibe nur so dahinerwähnt.

Fred Behringer

Hallo Fred,

meine Suche im Internet hat mich schnell zum PropellerForth gebracht, das Cliff Biffle von Google im Herbst 2006 begonnen hat. Leider hat Cliff, auch auf Nachfrage, entgegen seiner Ankündigung den Quellcode von PropellerForth noch nicht veröffentlicht, so dass wir bisher mit der Version `2006-11-14 release` und dem darin enthaltenen binärem Image Vorlieb nehmen müssen. Wer mag, kann ja mal auf der PropellerForth–Homepage vorbeischauen: `http://www.cliff.biffle.org/software/propeller/forth/index.php`. Außerdem gibt's Lesenswertes im Forum von Parallax `http://forums.parallax.com/`

Ulrich Hoffmann

Zum Erscheinungsbild der Ausgabe 4/2006

Dear Fred,

mein glanzvolles VD–Heft 4/2006 kam gestern an und ich kann euch erfreut mitteilen, dass es den Münchener Poststempel vom 30. Dezember 2006 trug. Damit sind also die vier Ausgaben des zweiundzwanzigsten Veröffentlichungsjahres termingerecht zusammengekommen. Ich darf allen gratulieren, die zum Erfolg beigetragen haben, und ich bin davon überzeugt, dass ihr es auch 2007 wieder schaffen werdet.

Als Allererstes, bevor ich zum eigentlichen Lesen kam, bewunderte ich natürlich den *New Look* des Heftes. Von denjenigen E–Mails, die in Kopie auch mich erreichten, war ich schon darüber *vorgewarnt*, dass viele von euch das schöne Erscheinungsbild auch bemerkt und darüber Meinungen ausgetauscht haben.

Ihr habt das Heft analysiert. Lasst mich zur Ergänzung ein paar eigene Worte hinzufügen. Ich zog die vorausgegangenen drei VD–Hefte aus meiner Motorola–Tasche, mit welcher ich normalerweise zum SVFIG–Treffen gehe, und legte sie allesamt vor mich hin, um sie mit dem neuen Heft zu vergleichen. Zunächst legte ich die VD 3/2006 beiseite. Sie hat vier Seiten mehr und ist um 10 Gramm schwerer. Die anderen drei Hefte haben alle 36 Seiten, wenn man die Deckblätter mitzählt, und wiegen je etwa 100 Gramm. Im neuen Heft wurde also kein schwereres Papier verwendet. Das Deckblatt des letzten Heftes macht einen weißeren, glanzvolleren und glatteren Eindruck als in den vorausgegangenen Heften. Dass das letzte Heft glatter oder rutschiger ist, wurde von einem *wissenschaftlichen* Test bestätigt, den ich mit meinem Tischtennis–Schläger veranstaltete. Wenn ich die Hefte nacheinander drauflege und den Schläger nach unten neige, gleitet das neue Heft leichter ab als die anderen. Im Inneren des Heftes konnte ich diesen Unterschied nicht feststellen, obwohl bei genauerem Hinsehen auch die inneren Seiten einen weißeren Eindruck machen.

Andere Unterschiede kann ich selbst mit einem Vergrößerungsglas nicht feststellen, da meine Augen inzwischen genauso trüb geworden sind wie die Scheinwerfer meiner alten Volvo–Maschine. Im Übrigen ist eine meiner Leidenschaften Schach und ich rufe beinahe täglich jene Web–Site `http://www.chessbase.com` auf, die auch Bernd erwähnt. Ich habe mir die unglaublichen optischen Täuschungen, auf die das Link von dort führt, angesehen. Meistens sehen wir ja nur das, was unser Gehirn uns vorgaukelt, ob es nun da ist oder nicht.

Fest steht jedenfalls, dass das Heft 4/2006 schöner aussieht als die Hefte davor. Über diesen Punkt sind sich meine Frau und ich einig.

Das nächste SVFIG–Treffen ist für den 27. Januar vorgesehen. Dr. Ting hat geschrieben, dass er gern einen Vortrag über eine verbesserte Version seines digitalen

Speicher–Oszillografen halten würde. Bis jetzt hat sich kein anderer Vortragender freiwillig gemeldet. Wie es aussieht, wird sich die Gruppe für den Rest des Tages wohl in geselligem Beisammensein üben oder sich DVDs vom Computergeschichtlichen Museum ansehen.

Zurück zur VD 4/2006. Ich habe das Heft gelesen und ich hoffe, dass eure Gruppe es weiterhin schafft, die Aufmerksamkeit von Leuten jenes Schlages auf sich zu ziehen, die wie ein gewisser VD–Autor agieren, welcher sich *Anfänger* nennt und doch zu den letzten beiden VD–Heften 21 Seiten beigetragen hat. Alle Achtung! Ein Hoch dem Erich Wälde!

With greetings to all, Henry

Übersetzt von Fred Behringer

Lebenszeichen

Bericht aus der FIG Silicon Valley: *Henry Vinerts*

Dear Fred,

lang lebe Forth! Ich freue mich wirklich über deine erfreulichen Zeilen. Ich war auf dem ersten Treffen der Silicon–Valley–Forth–Interest–Group in diesem Jahr und konnte mich davon überzeugen, dass es da noch mindestens 10 bis 12 Unermüdliche gibt, die dafür sorgen, dass die monatlichen Treffen nicht einschlafen. Voraussetzung dafür ist natürlich, dass uns das Cogswell College in seinem Gebäude weiterhin einen Vortragsraum zur Verfügung stellt.

Was das *immer noch lebende Forth* betrifft, würde ich sagen, dass die SVFIG–Treffen sicherlich auf einen halben Tag zusammenschrumpfen würden und sich kein Forth hören ließe, wenn es nicht Tings unerschöpflichen Ideenreichtum mit Forth in seinen Projekten und Hobbies gäbe, der nur noch von seiner Ausdauer beim Mitteilen und Teilen seines Wissens mit anderen erreicht wird.

Andererseits tut es gut zu sehen, dass nahezu 40 Jahre, nachdem Chuck Moores Sprache auf einer IBM 1130 das Licht der Welt erblickte, diese Sprache immer noch für viele Leute in aller Welt einen guten Grund abgibt, sich in Frieden zu versammeln, um Geselligkeit zu genießen und Neuigkeiten über gemeinsame Interessen auszutauschen.

Ich nehme an, dass die Zahl der Forth–Dialekte inzwischen die Zahl der Sprachen von Papua–Neuguinea übertrifft. Aber ich hoffe, dass der Atlantik und der Pazifik die Forth–Sprecher auf dieser Erde nicht genauso trennen werden, wie es die Bismarck– und die Solomon–See in Bezug auf die Papuas getan haben.

Fred, Dave Jaffe hat sicher wieder Notizen vom Januar–Treffen unter `http://www.forth.org` bereitgestellt und mir fällt nichts ein, was ich als besonders interessant hinzufügen könnte. Was ich dir jedoch noch sagen möchte: Auch Dr. Haydon ist der Meinung, dass das VD–Heft 4/2006 einen glänzenderen Eindruck macht als die vorhergehenden Ausgaben. Und alle waren davon begeistert, das Bild der SVFIG–Gruppe auf Seite 21 zu sehen.

With kind regards,
Henry

Übersetzt von Fred Behringer

Logos der schottischen Radiosender
Forth One (97,3 MHz, `http://www.forthone.com/`) und
Forth2 (1548kHz, `http://www.forth2.com/`)

CSD–Zahlendarstellung

Rafael Deliano

Die Multiplikation von festen Koeffizienten, wie sie besonders bei digitalen Filtern benötigt wird, kann im 2er–Komplementformat durch Shift&Add erfolgen. Dabei sind offensichtlich Koeffizienten günstig, bei denen nur eine geringe Zahl von Bits gesetzt ist. Ein Zahlenformat, das die Zahl der gesetzten Bits minimiert, indem es zusätzlich zu Addition auch Subtraktion verwendet, wird also die Multiplikation beschleunigen.

Das allgemeine SD–Format

(„Signed Digit“) ist ternär und hat als Grundelemente $-1, 0, +1$. In der Literatur wird für die negative Eins eine Überstreichung bevorzugt (Abbildung 1). Da das aber viele DTP–Systeme nicht vertragen, ist ein Kompromiss die Unterstreichung. ASCII–fähig, aber in Zusammenhang mit negativem Vorzeichen problematisch, ist das Verwenden nur der Vorzeichen. Hier wird die Variante mit „T“ bevorzugt.

```
-(-1,0,+1,0,0,-1 )
-T0100T     bevorzugt
-101001
--0+00-     ASCII-fähig
-T0100T
```

Abbildung 1: Schreibweisen

Es sind im SD–Format unterschiedliche Varianten in der Zahlendarstellung möglich. Der Spezialfall CSD („Canonical Signed Digit“) [2] hat die Eigenschaften: eine minimale Anzahl von 1 bzw. -1 und es sind immer Nullen zwischen diesen Ziffern (Abbildung 2).

```
                                    2er-Komplement
1DEh = 0   1    1    1   0   1   1   1   1   0
       *   *    *    *   *   *   *   *   *   *
       2^9 2^8  2^7  2^6 2^5 2^4 2^3 2^2 2^1 2^0
       =   =    =    =   =   =   =   =   =   =
       0 +256 +128 + 64 + 0 +16 + 8 + 4 + 2 + 0 = 478d

                                    CSD-Code
       1   0    0    0   -1  0   0   0  -1   0
       *                *                *
       2^9              2^5              2^1
       =                =                =
       512      +       -32      +       -2       = 478d
```

Abbildung 2: Zahlendarstellung

Interesse für derartige Rechenwerke war schon zu Zeiten der Röhrencomputer vorhanden. Damals wurden ternäre Codes noch als mögliche Alternative zur heute durchwegs üblichen binären Logik angesehen. Bald aber aus Mainframes verdrängt. In den 70ern für Signalverarbeitung wiederentdeckt [3] und dann auch 1979 im allerersten DSP, dem Intel 2920 verwendet. Danach speziell in bitseriellen VLSI–Schaltungen, heute zusätzlich vereinzelt auch in FPGAs eingesetzt.

Controller

Für variable Koeffizienten im SD–format gäbe es zwar „Booth's modified algorithm“, eine Variante des gängigen Booth–Multiplizierers [1], aber besonders auf Controllern ist nur die Multiplikation fester Koeffizienten im CSD–Format von Interesse. In VLSI–Schaltungen werden die Shifts durch geeignete Verdrahtung ausgeführt, nur die Additionen bzw. Subtraktion benötigen Schaltungen. Auf Controllern sind die Shifts explizit nötig (Abbildung 3). Man kann aus einer Shiftkette mehrere Multiplikationen ableiten (Abbildung 4). Die transponierte Form von kaskadierten Biquads (Abbildung 5) oder FIRs (Abbildung 6) sind hier günstig.

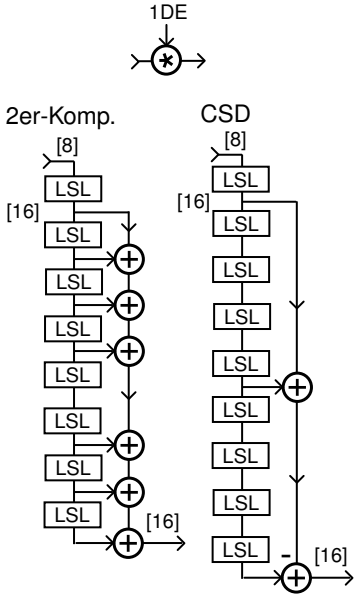


Abbildung 3: Implementierung auf Controller

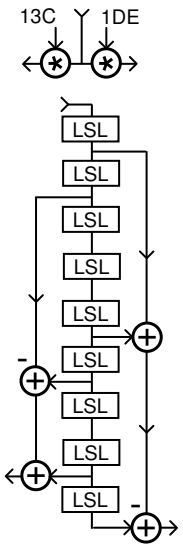


Abbildung 4: Mehrere Multiplikationen aus einer Shiftkette

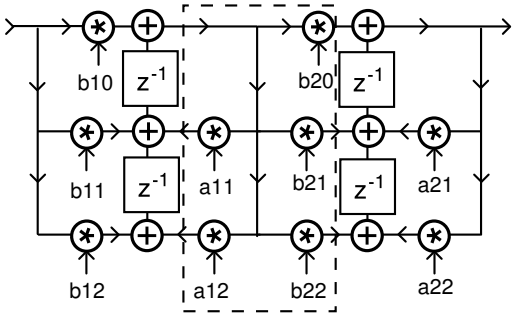


Abbildung 5: Kaskadierte transportierte Biquads

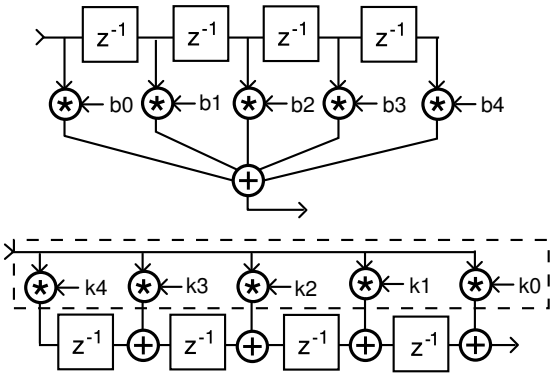


Abbildung 6: Transponiertes FIR–Filter

Programm

Der ursprüngliche Reitwiesner–Algorithmus (Abbildung 7) wandelt eine vorzeichenlose 2er–Komplementzahl x seriell auf CSD–Format y um. Ein Carrybit c wird als Zwischenspeicher verwendet. Die Benutzung der Tabelle ist anhand eines Rechenbeispiels (Abbildung 8) leicht nachvollziehbar. Das entsprechende Programm (Listing 1) ist trivial, druckt dann aber als Schönheitsfehler LSB links aus.

in			out	
x_{n+1}	x_n	c_n	y	c_{n+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	T	1
1	1	0	T	1
1	1	1	0	1

Abbildung 7: Tabelle für Umwandlung

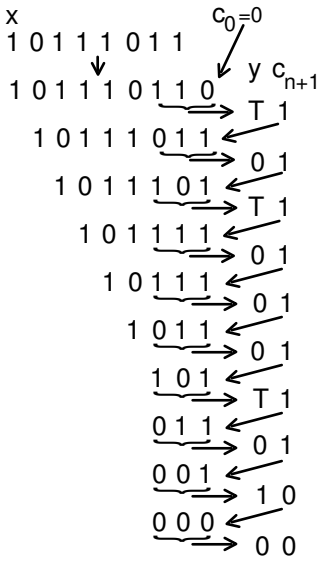


Abbildung 8: Beispiel für Verwendung von Tabelle

Literatur

[1] emb (2)

[2] Reitwiesner „Binary Arithmetic“ in: Advances on Computers Vol 1. Academic Press 1960

[3] Peled „An architecture for digital signal processors“ IEEE Electronics and Aerospace Systems Convention IEEE EASCON 1975

```
1   Listing 1:
2   <| \ CSD
3
4   TABLE SYM-TAB
5   30 C, 31 C, 31 C, 30 C,
6   30 C, 54 C, 54 C, 30 C,
7
8   TABLE Y+1-TAB
9   00 C, 00 C, 00 C, 01 C,
10  00 C, 01 C, 01 C, 01 C,
11
12  : CSD      \ ( UN1 --  )
13  CR ." LSB" CR
14  1<SHIFT    \  -- UN1' )
15  F 0 DO
16  DUP 7 AND DUP
17  SYM-TAB + C@ EMIT SPACE
18  Y+1-TAB + C@ SWAP 1SHIFT>
19  FFFE AND OR
20  LOOP DROP CR ;
21
22  |>
23
24  Testlauf:
25
26  --- --- --- | BB CSD
27  LSB
28  T 0 T 0 0 0 T 0 1 0 0 0 0 0 0 0
29  --- --- --- |
```

Forth auf dem PC — Ein Gedankenexperiment

André Elgeti

In diesem Gedankenexperiment geht es um die Stellung von Forth in den Software–Generationen:

Es sei diese Einteilung gegeben:

1. Generation – Hex–Code
2. Generation – Maschinenorientierte Programmierung
3. Generation – problemorientierte Programmierung
4. Generation – nutzerorientierte Programmierung — Forth
5. Generation – wissensorientierte Programmierung

Was kann das bedeuten? Stellen Sie sich einmal vor, der PC wäre ihr Assistent, statt ihr Werkzeug. Sie würden ihm Aufträge erteilen, statt die Arbeit selbst zu machen. In der Praxis sähe das beispielsweise dann so aus:

Sie starten ihren PC und der Forth–Kern startet die Systemsoftware. Nach dem Abschluss steht ein `ok` und ein blinkender Cursor auf dem ansonsten leeren Bildschirm.

Das `ok` sei das Kürzel für „Was kann ich für Sie tun?“

Sie geben ein `Möglichkeiten anzeigen` und ein Menubildschirm zeigt Ihnen die eingestellten Möglichkeiten. Weil Sie gern einen Brief an Ihre Oma schreiben möchten, geben sie ein: `Brief briefanoma`, und danach `briefanoma anzeigen`. Auf dem Bildschirm erscheint:

```
Absender:
Adressat:
Betreff:
Anrede:
Text:
Schluss:
Anlagen:
```

Jeder dieser Einträge stellt einen Absatz dar, der vom Anwender ansprechbar ist. Mit der Eingabe `Oma Adressat ändern`, ändern sie den Absender mit den unter Oma gespeicherten Daten. `Ich Absender ändern` trägt ihre persönlichen Daten ein.

So ändern Sie Absatz für Absatz. Mit `Text eingeben` können sie z. B. den Brieftext von der Tastatur abfragen, und mit `Text bearbeiten` korrigieren. Mit `briefanoma anzeigen` können Sie die Ergebnisse ablesen. Zuletzt bringt `briefanoma drucken` den Text zu Papier.

Anschließend geben sie `Brief vergessen` ein und das System ist für die nächste Arbeit bereit. Ein `bye` beendet die Sitzung und fährt das System herunter.

Die praktische Umsetzung dieser Idee birgt einen riesigen Berg Arbeit, denn es müßte das gesamte Multimedia– und Internet–System erzeugt werden. Das Wichtigste aber, was sowohl Gefahr als auch Nutzen sein kann, ist, dass sie dem gängigen Menu–System widerspricht, weil ein Forth–System eine Meta– oder Fachsprache darstellen soll.

Die Systemmeldung `Was kann ich für Sie tun?` ergibt die Möglichkeit, eine quasinatürliche Kommunikation des Menschen mit dem System zu führen , z. B. `Datei öffnen`. Um das zu ermöglichen, sind die Namen der Forth–Wörter im obigen Beispiel Tätigkeitswörter (`anzeigen`, `drucken`, `ändern` usw.).

Forth ermöglicht es, Daten im Wörterbuch abzulegen, wo sie abgerufen und bearbeitet werden können. Könnte man das Fachwissen einschließlich Abbildungen so ablegen, wäre man weniger vom Internet abhängig, und die Internetarbeit wie eine Tankstelle.

Das wäre doch eine schöne Möglichkeit für die Zukunft.

Für den Rätselfreund

```
create xxx ' create @ here 2- ! ' dup , ' + , ' dup , ' + , ' exit ,
```

Ausführbar (zumindest) in ZF und Turbo-Forth.

Was würde man statt xxx sinnvollerweise schreiben? Was soll mit diesem *Rätsel* gezeigt werden?

```
variable yyy ' variable @ ' dup yyy ! ' yyy ! ' + , ' dup , ' + , ' exit ,
```

Was würde man hier für `yyy` einleuchtenderweise schreiben? Was haben `xxx` und `yyy` gemeinsam? Was ist wesentlich anders? Wie sehen Systeme aus, in denen `yyy`, aber nicht `xxx` funktioniert? Wie kann man `xxx` so ändern, dass es auch dann noch geht?

Fred Behringer

Adventures in Forth 3

Erich Wälde

Teil 1: Ausgaben auf `uart0`

Der R8C13-Controller hat eine weitere serielle Schnittstelle (`uart0`), die beim Programmieren nicht benötigt wird. Über diese sollen Daten verschickt werden. Die Daten sollen als ASCII-Zeichenkette verschickt werden, abgeschlossen mit CR LF. Auf ein Protokoll zum Absichern des Datentransports soll an dieser Stelle verzichtet werden. Es sind drei Aufgaben zu lösen:

1. Initialisieren der Schnittstelle
2. Formatieren der zu verschickenden Zeichenkette
3. Ausgabe der Zeichenkette auf die Schnittstelle

Um die für `uart0` wichtigen Register und Einstellungen zu finden, habe ich das Datenblatt konsultiert. Die Schnittstelle liegt auf den Pins 4 und 5 von Port 1.

```
$E1 Constant port1    \ port 1
$E3 Constant pddr1    \ port direction reg. 1

&4  Constant PinTxD0 \ P1.4 is TxD0
&5  Constant PinRxD0 \ P1.5 is RxD0

                      \ uart0
$A0 Constant U0MR     \ mode register
$A4 Constant U0C0     \ control register 0
$A5 Constant U0C1     \ control register 1
&0  Constant TE       \ transmit enable bit
&1  Constant TI       \ transmit buffer empty?
$B0 Constant UCON     \ control register 2
&2  Constant U0RRM    \ receive mode bit
$A1 Constant U0BRG    \ bit rate
$A2 Constant U0TBRL   \ transmit buffer, low
```

Die Schnittstelle soll mit 38400 baud, 8 Datenbits, kein Paritätsbit, 1 Stopbit und ohne handshake betrieben werden. Die zugehörigen Einstellungen finden sich ebenfalls im Datenblatt. Schwierig ist das alles nicht, aber ein wenig Geduld hilft.

```
: uart0-init ( -- )
  PinTxD0 port1 bset \ TxD0 is 1
  PinTxD0 pddr1 bset \ TxD0 is output
  PinRxD0 pddr1 bclr \ RxD0 is input
  $05 U0MR c!        \ 8N1, internal clock
  $00 U0C0 c!        \ clock f1, TxD0 CMOS out,
                     \ . data on falling edge,
                     \ . lsb first
  U0RRM UCON bclr    \ continuous receive OFF
  &32 U0BRG c!       \ 38400 @ 20 MHz
;
```

Als Nächstes definieren wir ein Wort, welches ein Zeichen auf `uart0` ausgibt. Zuerst warten wir, bis das letzte Zeichen übertragen ist, d.h. bis das Bit `TI` gelöscht wurde. Danach schreiben wir das Zeichen in das Senderegister `U0TBRL`, und sagen dem Controller, dass ein neues Zeichen zu verschicken ist, durch Setzen des Bits `TE`.

```
: uart0emit ( n -- )
                     \ transmit buffer empty?
  BEGIN TI U0C1 btst UNTIL
  U0TBRL c!          \ n to transmit buffer
  TE U0C1 bset       \ enable transfer
;
```

Dieses Wort lässt sich jetzt schon einsetzen, um die serielle Verbindung zu prüfen. `uart0`, auf der Elektorplatine mit *COM port* bezeichnet, wird mit einem Rechner verbunden (es gibt jetzt 2 Kabel vom Rechner zur Platine). Ein Programm wie *minicom* oder *Hyperterminal* liest dann die Zeichen, die der Kontroller verschickt. Wer die ASCII-Tabelle auswendig kann, ist klar im Vorteil:

```
Gforth terminal
Press ENTER to get ok from connected device.
Leave with BYE
include adv3_uart0.fs  ok
uart0-init  ok

$48 uart0emit $61 uart0emit $6c uart0emit ok
$6c uart0emit $6f uart0emit $21 uart0emit ok
```

Das produziert am *anderen Ende* ein freundliches

```
Hallo!_
```

wobei _ den Cursor bezeichnet. Die größte Hürde wäre damit geschafft. Allerdings ist es schon ziemlich unpraktisch, eine Zeichenkette so umständlich auszugeben. Das Wort `type` ist in allen Forths vorhanden und gibt eine Zeichenkette aus. Wenn man ein wenig die Dokumentation studiert, dann findet man, dass `type` zwei Argumente auf dem Stapel erwartet: `(addr n - )`. `type` erwartet dann eine Zeichenkette der Länge `n` an der Adresse `addr`. Ein Wort von der Art wie `type` können wir uns selbst schreiben:

```
\ send n bytes starting at addr
: uart0type ( addr n -- )
  0 DO               \ do-loop consumes n
    dup I +          \ { consumes copy of addr
    c@ uart0emit     \   get+send byte
  LOOP               \ }
  drop               \ consumes addr
;
```

Mit Hilfe des Wortes `s"` wird die Welt schön. `s"` produziert aus dem nachfolgenden Text (abgeschlossen mit `"`) eine Zeichenkette, und legt die Adresse und die Länge auf den Stapel. `uart0type` benutzt diese Information und

```
s" Hallo!" uart0type
```

produziert das gleiche freundliche `Hallo!` wie die Übung vorher, nur viel bequemer. Ein zusätzliches Wort `uart0cr` vereinfacht die Sache weiter:

```
: uart0cr ( -- )
  $0d uart0emit $0a uart0emit
;
```

Teil 2: Ausgabeumleitung

Wenn man auf das Wort `type` gestoßen ist und ein wenig in der Dokumentation von gforth (auch das Wort `see` wurde mein Freund) und/oder im Buch gelesen hat, dann ist es nicht sehr weit zu sagen: Wäre doch schön, wenn man die gleiche Ausgabe am gforth-terminal, auf `uart0` und auf das LCDisplay benutzen könnte. Etwa so:

```
: msg s" blabla" ;
: run
  to-lcd msg
  to-uart0 msg
  to-stdout msg
;
```

Die kurze Antwort ist: *Man kann.*

Um dem gleichen Wort verschiedene Aufgaben unterzuschieben benutzt man so etwas wie *Zeiger auf Funktionen*. Das haben wir bei `timeup` in der zweiten Folge schon einmal gesehen: Man kann die *Einsprungadresse* eines Wortes in einer Variablen speichern und dann mit `execute` die Ausführung erreichen. Zur Erinnerung:

```
create Jobs ' job.tick , job.sec , ...
...
... DO
  ...
  I cells Jobs + @ execute
  ...
LOOP
```

Die Frage heißt also: kann man mit den Worten `type` (gib eine Zeichenkette aus) und `emit` (gib ein Zeichen aus) so eine Akrobatik veranstalten? Also eigentlich müsste man ja die Adresse von `type` in einer Variablen speichern und dann alle Worte, die eine Ausgabe machen, so umdefinieren, dass sie über die Variable auf die Worte `type` und `emit` zugreifen — nee, das ist nicht schön, denn es gibt einige solcher Worte (`. u. d. d.r ."` usw.). Da spart man nichts, wenn man die alle umschreiben muss.

Jetzt stellt sich aber günstigerweise heraus, dass `type` selbst nur auf ein Wort namens `(type)` verweist. Das lässt hoffen.

Wir basteln uns also drei Variablen, die Zeiger auf die gerade benötigten Versionen von `type`, `emit` und der Vollständigkeit halber `cr` beinhalten.

```
Variable 'type  \ pointer to correct "type"
Variable 'emit  \ pointer to correct "emit"
Variable 'cr    \ pointer to correct "cr"
```

Dann brauchen wir drei Worte, die diesen Variablen die korrekten Werte verpassen, falls wir die Ausgaben auf `lcd`, `uart0` oder das gforth terminal (`stdout`) umbiegen wollen. Dabei gibt es noch eine kleine Schwierigkeit. Normalerweise liefert uns das Wort `'` (*tick*) die Einsprungadresse des nachfolgenden Wortes — außer in einer `:`-Definition. Dort muss man stattdessen `[']` benutzen. Das liegt daran, dass innerhalb einer `:`-Definition Forth im compile-Modus läuft. Wir wollen aber nicht `'` einkompiliert haben, sondern das Ergebnis von `' wort`. Das Wort `[']` schaltet daher den compile-Modus vorübergehend aus.

```
: to-stdout ( -- )
  ['] (type)    'type !
  ['] (emit)    'emit !
  ['] cr        'cr   !
;
: to-lcd ( -- )
  ['] lcdtype   'type !
  ['] lcdemit   'emit !
  ['] lcdcr     'cr   !
;
: to-uart0 ( -- )
  ['] uart0type 'type !
  ['] uart0emit 'emit !
  ['] uart0cr   'cr   !
;
```

Jetzt fehlt noch die Verbindung zwischen den alten Worten und der Variablen, die auf die umgeleiteten Worte zeigt.

```
: >type ( -- ) 'type @ execute ;
: >emit ( -- ) 'emit @ execute ;
: >cr   ( -- ) 'cr   @ execute ;
```

Die Umleitung lässt sich jetzt aktivieren, indem man `type` auf `>type` zeigen lässt. Das Wort `is` schiebt dem nachfolgenden `type` das Ergebnis von `[']` `>type` als Adresse der neuen Anweisungen unter.

```
: +redir ( -- )
 ['] >type is type \ "type" output redirection
 ['] >emit is emit \ "emit" output redirection
;
: -redir ( -- )
  ['] (type) is type
  ['] (emit) is emit
;
to-stdout   \ default
```

Und wie funktioniert das jetzt? Wir schreiben einen kleinen Test. In `init` wird die serielle Schnittstelle `uart0` initialisiert, einmal CR LF verschickt und die Anzeige auf dem LCDisplay gelöscht. Dann wird die Umleitung eingeschaltet.

`msg` produziert zwei Zeilen Ausgabe. Man beachte, dass hier nicht `cr`, sondern `>cr` verwendet wird. Das ist der Preis dafür, dass `cr` eben kein Zeiger auf ein anderes Wort ist. Wenn man jetzt dem Wort `cr` mit `' >cr is cr` etwas anderes zu tun unterschiebt, dann verliert man

den Zugriff auf die Originalanweisungen. Das kann man dann auch nicht mehr zurücksetzen. Es sieht allerdings so aus, als würde `cr` nur vom Programmierer benutzt — damit bleibt alles unter unserer Kontrolle. Wenn alles klappt, dann sieht die Ausgabe so aus:

```
cr run-test
String 1  32
String 2   x
```

`run-test` schickt diese Zeichenketten an alle drei Ausgabegeräte: auf das LCDisplay, auf die serielle Schnittstelle und auf die Ausgabe im gforth terminal.

```
include adv3_uart0.fs
include adv3_redir.fs
: init-test
  uart0-init uart0cr
  lcdpage
  +redir
;
: msg
  ." String 1" $20 s>d4 d.r >cr
  ." String" space ." 2" 3 spaces $78 emit
;
: run-test
  init-test
  to-lcd    msg
  to-uart0  msg
  to-stdout msg
;
```

Wirklich schön. Man hat Zeiger-auf-Funktionen in der Gegend herumgebogen. Aber das geht doch mit $MEINE_LIEBLINGS_SPRACHE auch, oddrrr? Vielleicht. Aber in diesem Fall haben wir immerhin zwei Funktionen aus der Sprache (`emit`, `type`) verbogen — auch wenn das netterweise schon vorbereitet war. Das führt dazu, dass wir die Worte `.`, `."`, `d.r` etc. für jede Ausgabe einfach weiterverwenden können, wie wenn das immer so gegangen wäre. Zurückbiegen geht übrigens auch ohne Verluste. Ob das alles in $MEINE_LIEBLINGS_SPRACHE immer noch so einfach ist?

Wenn man vor dem nächsten `empty` noch `-redir` eingibt, dann kann man sich auch den Reset sparen. Ergänzungen, Kommentare, Korrekturen sind ausdrücklich erwünscht. Sie erreichen mich unter `ew.forth@nassur.net`

Referenzen

1. E. Wälde, Adventures in Forth, Die 4. Dimension 3/2006, Jahrgang 22
2. E. Wälde, Adventures in Forth 2, Die 4. Dimension 4/2006, Jahrgang 22
3. Renesas R8C/13 Datenblatt, siehe `www.renesas.com`

Listings

```
\ 2006-07-09 EW
\ Bytes auf uart0 verschicken

\ uart0 is on port 1
$E1 Constant port1   \ port 1
$E3 Constant pddr1   \ port direction reg. 1

&4  Constant PinTxD0 \ P1.4 is TxD0
&5  Constant PinRxD0 \ P1.5 is RxD0
                     \ uart0
$A0 Constant U0MR    \ mode register
$A4 Constant U0C0    \ control register 0
$A5 Constant U0C1    \ control register 1
&0  Constant TE      \ transmit enable bit
&1  Constant TI      \ transmit buffer empty?
$B0 Constant UCON    \ control register 2
&2  Constant U0RRM   \ receive mode bit
$A1 Constant U0BRG   \ bit rate
$A2 Constant U0TBRL  \ transmit buffer, low

: uart0-init ( -- )
  PinTxD0 port1 bset \ TxD0 is 1
  PinTxD0 pddr1 bset \ TxD0 is output
  PinRxD0 pddr1 bclr \ RxD0 is input
  $05 U0MR c!        \ 8N1, internal clock
  $00 U0C0 c!        \ clock f1, TxD0 CMOS out,
                     \ . data on falling edge,
                     \ . lsb first
  U0RRM UCON bclr    \ continuous receive OFF
  &32 U0BRG c!       \ 38400 @ 20 MHz
;

\ send one byte
: uart0emit ( n -- )
                     \ transmit buffer empty?
  BEGIN TI U0C1 btst UNTIL
  U0TBRL c!          \ n to transmit buffer
  TE U0C1 bset       \ enable transfer
;

\ send n bytes starting at addr
: uart0type ( addr n -- )
  0 DO               \ do-loop consumes n
    dup I +          \ { consumes copy of addr
    c@ uart0emit     \   get+send byte
  LOOP               \ }
  drop               \ consumes addr
;

\ send CR LF
: uart0cr ( -- )
  $0d uart0emit
  $0a uart0emit
;
```

```
\ 2006-12-03  EW  adv3_redir.fs
\ poor man's redirection
\ (stdout, lcd, uart0)

Variable 'type  \ pointer to correct "type"
Variable 'emit  \ pointer to correct "emit"
Variable 'cr    \ pointer to correct "cr"

```

```
: to-stdout ( -- )
  ['] (type)    'type !
  ['] (emit)    'emit !
  ['] cr        'cr   !
;
: to-lcd ( -- )
  ['] lcdtype   'type !
  ['] lcdemit   'emit !
  ['] lcdcr     'cr   !
;
: to-uart0 ( -- )
  ['] uart0type 'type !
  ['] uart0emit 'emit !
  ['] uart0cr   'cr   !
;
: >type ( -- ) 'type @ execute ;
: >emit ( -- ) 'emit @ execute ;
: >cr   ( -- ) 'cr   @ execute ;

: +redir ( -- )
  ['] >type is type
  ['] >emit is emit
;
: -redir ( -- )
  ['] (type) is type
  ['] (emit) is emit
;
to-stdout   \ default

```

```
rom

include adv3_uart0.fs
include adv3_redir.fs

: init-test
  uart0-init
  $0d uart0emit $0a uart0emit
  lcdpage
  +redir
;

: msg
  ." String 1" $20 s>d 4 d.r >cr
  ." String" space ." 2" 3 spaces $78 emit
;

: run-test
  init-test
  to-lcd    msg
  to-uart0  msg
  to-stdout msg
;
\ main

\ ." call run-test"


ram
```

Adventures in Forth 4

Erich Wälde

Eine Funkuhr

Im zweiten Teil (2) dieser kleinen Artikelserie wurde eine Uhr für Steuerungsaufgaben realisiert (`timeup`). Diese Uhr wollte ich nun mit einem DCF-77 Empfänger ausrüsten. Allerdings zeigte sich beim Programmieren, dass diese Aufgabe mehr zu einer *Studie für Designentscheidungen* geriet, als ich gutheißen kann.

Ich hatte noch einen arbeitslosen DCF-77 Empfänger herumliegen. Den invertierten Signalpin des Empfängers verband ich mit dem Pin 5 von Port D. Dieser Pin des Renesas R8C/13–Kontrollers kann auch Interrupts auslösen.

Das DCF-Signal ist recht einfach aufgebaut. Jede Sekunde wird ein Bit übertragen. Der Wert des Bits wird durch die Dauer kodiert: ein Signal von 100 ms wird als Null, ein Signal von 200 ms wird als Eins interpretiert. In der 59. Sekunde wird kein Bit übertragen. Die führende Flanke des nächsten Bits markiert den Beginn der neuen Minute. Die Daten sind als *binary coded decimal* (BCD) Werte kodiert und das niedrigstwertige Bit wird zuerst übertragen. BCD bedeutet, dass in jeder Hälfte des Bytes eine Ziffer übertragen wird, also etwa `0x16` für 16 und nicht etwa `0x10`. Die Bedeutung der DCF-Bits sei hier aufgelistet, soweit für das Programm von Bedeutung (5).

Bit Nr.	
0	Beginn der Minute
17	Sommerzeit
21–27	Minute
28	Parität Minute
29–34	Stunde
35	Parität Stunde
36–41	Tag
42–44	Wochentag
45–49	Monat
50–57	Jahr−2000
58	Parität
59	kein Signal

Entscheidungen

Eine oder zwei Uhren?

Ich hatte so eine DCF-Uhr vor ein paar Jahren schon einmal in 8051-Assembler geschrieben. Damals fand ich heraus, dass das DCF-Signal durchaus ohne Vorwarnung *ausbleibt*, und das für mehrere Minuten. Das mag am Wetter, der Geographie oder den Unzulänglichkeiten des Empfängers liegen. Bei Gewittern ist das Signal außerdem stark gestört. Ich will aber eine Uhr, die korrekt weiterläuft, wenn das Signal ausbleibt. Also nehme ich `timeup` als Grundlage. Ganz grob will ich zwei Funktionen in `timeup` einhängen. Eine Funktion soll *oft* das Signal der DCF-Uhr abtasten und auswerten, die andere soll *selten* die `timeup`-Uhr mit der DCF-Uhr synchronisieren.

Entscheidung 1: Die `timeup`-Uhr läuft unabhängig; sie wird mit der DCF-Uhr gelegentlich synchronisiert.

Abtasten oder Interrupts?

Es gibt zwei Möglichkeiten, das DCF-Signal auszuwerten. Entweder man tastet das Signal regelmäßig ab oder man lässt jede Flanke einen Interrupt erzeugen, welcher die Auswertung des Signals auslöst. Die zweite Variante hat den Vorteil, dass das Programm nur dann das Signal auswertet, wenn es wirklich was zu tun gibt. Die erste Variante hat den Vorteil, dass sie einfach zu durchschauen ist. Erschwerend kam hinzu, dass ich bislang nicht herausgefunden habe, wie ich einem Interrupt aus Forth heraus eine *interrupt service routine* verpasse — die Götter und die Gurus mögen mir verzeihen.

Entscheidung 2: Das Signal wird abgetastet.

Daran schließt sich sofort die Frage an, wie oft das Signal abgetastet wird. Für eine Null dauert das Signal 1/10 s. In dieser Zeit möchte ich das Signal vielleicht 10–mal abtasten. Dann erhält man normalerweise Zählwerte von 8 oder 9. Wenn außerdem eine Abtastung durch Rauschen falsch ist, dann ist das immer noch von Eins (16-19 Zähler) oder Kein-Signal (0-1 Zähler) gut zu unterscheiden.

Entscheidung 3: Die Abtastung erfolgt mit 100 Hz.

Das lässt sich sehr einfach realisieren, indem man der `timeup`-Uhr die entsprechenden Konstanten verpasst.

```
10  Constant cycles.tick \ timerC cycles/tick
100 Constant ticks.sec   \ ticks/second
```

Die Abtastung des Signals ruft man dann in `job.tick` auf.

Synchronisieren auf die Sekunde

In Sekunde 59 bleibt das Signal aus, die neue Minute fängt an der Flanke des nächsten Bitsignals an. Man könnte versucht sein, auf diese Flanke zu warten. Was, aber wenn diese Flanke dann ausbleibt oder durch Rauschen zu früh vorgetäuscht wird? Das hat mir auch nicht so recht gefallen. Also habe ich beschlossen, die `timeup`-Uhr anders an das Signal anzugleichen.

Das Signal wird also 100–mal in der Sekunde gemessen. Ist der Pegel 0 V (aktiv), dann wird der Pulszähler erhöht, andernfalls der Pausenzähler:

```
: dcf.readPin ( -- t/f )
  \ pin is on "active low" connection -> invert
  pinDCF portDCF btst invert
;
: dcf.tick ( -- )
```

```
  dcf.readPin

  \ count up Pulse/Pause counters
  IF
    1 dcfPulse +!
  ELSE
    1 dcfPause +!
  ENDIF
  ...
;
```

Zum Erkennen von Null, Eins und der 59. Sekunde reicht das aus. Allerdings könnte es sein, dass der Bitpuls so spät in der `timeup`-Sekunde losgeht, dass sein Ende bereits in der nächsten Sekunde liegt. Das kann dazu führen, dass über lange Zeit kein gültiges DCF-Telegram empfangen wird, vor allem dann, wenn eine Null noch innerhalb der `timeup`-Sekunde liegt, eine Eins aber in die nächste Sekunde hineinreicht.

Um das zu vermeiden, will ich feststellen, ob der Puls des Bits komplett im ersten Viertel der `timeup`-Sekunde liegt. Wenn nicht, dann kann ich das Auswerten des Bits über ein Offset etwas verschieben. Damit wird die Verschiebung der beiden Uhren zumindest gemessen. Dieses Verfahren ist auch einigermaßen robust gegen Störungen des Signals.

Entscheidung 4: Die `timeup`-Uhr auf wenige ticks (10 ms/tick) mit der DCF-Uhr synchronisiert.

Auswertung fortlaufend oder am Ende?

Eine weitere Frage ist die, ob man zuerst alle Bits einsammelt und die Information dann am Ende der Minute am Stück auswertet oder ob man nach jedem Bit die Information auswertet. Auch das ist eher Geschmacksache.

Entscheidung 5: Die Bits werden fortlaufend ausgewertet und nicht erst am Ende der Minute.

Code 1: DCF-Signal abtasten

Als Grundlage dient der Code aus dem 2. Teil: eine `timeup`–Uhr, bei der alle Teile, die mit dem i2c–Bus zu tun haben, herausgenommen wurden. Die Worte für die DCF-Uhr befinden sich in der Datei `adv4_dcf.fs`. Diese erwartet ein paar definierte Konstanten und die Worte `led2_0` und `led2_1`, um das DCF-Signal direkt auf LED2 anzuzeigen.

```
$e8   Constant portDCF \ port-D
$ea   Constant pddrDCF \ pddr-D
5     Constant pinDCF
: led2_0 ( -- ) 2 port1 bclr ;
: led2_1 ( -- ) 2 port1 bset ;
include adv4_dcf.fs
```

`adv4_dcf.fs` enthält zunächst drei simple Worte:

```
: dcf.readPin
  \ pin is on "active low" connection
  \ --> invert
  pinDCF portDCF btst invert
;
: dcf.init
  pinDCF pddrDCF bclr      \ set pinDCF input
;
: dcf.tick
  dcf.readPin
  \ show DCF-Signal "active" on led2
  IF led2_1 ELSE led2_0 ENDIF
;
```

Zwei dieser Worte werden in den Innereien der `timeup`–Uhr aufgerufen: `dcf.init` in `loop.init` und `dcf.tick` in `job.tick`.

Wenn man das Programm lädt und mit `run` die Uhr startet, dann blinken die beiden LEDs Nr. 2 und 3. LED Nr. 3 blinkt im Sekundentakt, Puls und Pause sind gleich lang. LED Nr. 2 blinkt wie das DCF–Signal. Man kann recht bald Null und Eins (leuchtet länger) durch bloßes Hinsehen unterscheiden. Ebenso ist die Pause bei Sekunde 59 (kein Signal) bald gefunden.

Code 2: Bits vermessen

Im nächsten Schritt werden zwei Variablen `dcfPulse` und `dcfPause` definiert. Mit diesen werden die Längen von Puls (aktives DCF-Signal) und Pause vermessen. Außerdem verwende ich die Ausgabeumleitung aus dem dritten Teil (**??**) für die Ausgaben.

```
Variable dcfPulse
Variable dcfPause
: dcf.tick
  dcf.readPin
  \ show pin "active" on led2
  dup IF led2_1 ELSE led2_0 ENDIF

  \ count up Pulse/Pause counters
  IF
    1 dcfPulse +!
  ELSE
    1 dcfPause +!
  ENDIF

  \ reload counters if tick == 0
  tick @ 0 = IF
    to-stdout cr
    dcfPulse @ s>d 3 d.r space
    dcfPause @ s>d 3 d.r
    dcf.reload
  ENDIF
;
```

Wenn `tick` aus `adv4_timeup.fs` gleich null ist, dann werden die Zählerstände ordentlich formatiert ausgegeben und die Zähler zurückgesetzt. Diese Aufgaben würde man eigentlich in einem Wort `dcf.sec` erwarten, welches von `job.sec` aufgerufen wird. Allerdings habe ich in `dcf.tick` die Möglichkeit, die Auswertung des DCF-Signals innerhalb der Minute der `timeup`-Uhr zu verschieben, falls das nötig wird. `dcf.tick` produziert etwa folgende Ausgabe:

```
include adv4_main.fs  ok
run  ok
...
  8  92
  9  91
  8  92   <== 0 bit
 18  82   <== 1 bit
  0 100   <== sync!
...
```

Die Bitfolge des Zeitsignals ist mit dieser Ausgabe leicht zu sehen. Der gezeigte Abschnitt zeigt ein makelloses DCF–Signal. Aber ich brauche auf gestörte Signale nie sehr lange zu warten:

```
 10  90
  9  91
 26  74   <== Störungen!
 21  79   .
 43  57   .
 17  83   .
  7  93   .
  0 100   .
  0 100   .
 10  90   .
  1  99   .
```

Code 3: Bits auswerten

Unter der Annahme, dass `dcf.tick` tatsächlich regelmäßig aufgerufen wird, kann man aus dem Wert in `dcfPulse` auf den Wert des Bits schließen. Außer den Werten 0 und 1 für die Bits sollen noch -1 für Fehler und -2 für die Erkennung der 59. Sekunde verwendet werden. Bei dieser Gelegenheit soll auch ein Bit eingeführt werden, welches das Auftreten eines Fehlers innerhalb eines Telegramms (eine Minute) anzeigt. Außerdem wird in der neuen Variablen `dcfPos` die Position innerhalb des Datentelegramms gespeichert.

```
Variable dcfFlags
$00 Constant dcfError
Variable dcfPos
: dcf.error.set  dcfError dcfFlags bset ;
: dcf.error.clr  dcfError dcfFlags bclr ;
: dcf.error?     dcfError dcfFlags btst ;
: dcf.init
  ...
  dcf.error.set \ error unless proven ok.
;
: dcf.bit ( pulse -- bit/error )
  dup  2 < IF ( sync59      ) -2 ELSE
  dup  6 < IF dcf.error.set   -1 ELSE
  dup 11 < IF ( bit:0       )  0 ELSE
  dup 16 < IF dcf.error.set   -1 ELSE
  dup 21 < IF ( bit:1       )  1 ELSE
    dcf.error.set             -1
  ENDIF ENDIF ENDIF ENDIF ENDIF
  swap drop
;
: .3r  s>d 3 d.r space ;
: dcf.tick
  ...
  tick @ 0 = IF
    dcfPulse @ dcf.bit
    to-stdout cr
    dcfPulse @ .3r
    dcfPause @ .3r
    dcfPos   @ .3r
    dup        .3r \ bit value
    dcf.error? .3r
    dcf.reload

    -2 = IF \ sync detected
      dcf.error.clr
      0 dcfPos !
    ELSE
      1 dcfPos +!
    ENDIF
  ENDIF
;
```

Die Funktion `dcf.bit` bewertet den Zählerstand aus `dcfPulse`. Wurde die Pause in der 59. Sekunde gefunden, dann wird das Fehlerbit gelöscht und `dcfPos` zurückgesetzt. Die Ausgabe sieht dann etwa so aus.

```
  8  92   0   0  -1
 18  82   1   1  -1
 18  82   2   1  -1
...
  8  92  17   0  -1
  9  91  18   0  -1
  0 100  19  -2  -1  <== 1. sync
 10  90   0   0   0
 19  81   1   1   0
 19  81   2   1   0
...
  8  92  56   0   0
  9  91  57   0   0
  9  91  58   0   0
  0 100  59  -2   0  <== sync
  9  91   0   0   0
  9  91   1   0   0
 19  81   2   1   0
...
```

Code 4: Bits zu Zahlen schmieden

Soweit habe ich erreicht, dass das DCF-Signal abgetastet und jede Sekunde ein Bit ausgewertet wird. Die Sekunde 59 wird ebenfalls erkannt und `dcfPos` zählt die Bitpositionen des Daten-Telegramms. Im nächsten Schritt geht es darum, den Bitstrom in Zahlen umzuwandeln.

Dafür könnte man einen einfachen, aber dafür großen `case`-Block erstellen, der das erledigt:

```
dcf.bit \ bit: 0 or 1
dcfPos @ CASE
   0 OF 0 minute ! ...                  ENDOF
  21 OF IF  1 minute +! Ptoggle ENDIF ENDOF
  22 OF IF  2 minute +! Ptoggle ENDIF ENDOF
  23 OF IF  4 minute +! Ptoggle ENDIF ENDOF
```

```
  24 OF IF  8 minute +! Ptoggle ENDIF ENDOF
  25 OF IF 10 minute +! Ptoggle ENDIF ENDOF
  26 OF IF 20 minute +! Ptoggle ENDIF ENDOF
  27 OF IF 40 minute +! Ptoggle ENDIF ENDOF
  28 OF Ptoggle
        Pset? IF dcf.error.set ENDIF
     ENDOF
  ...
  ( default: tue nichts )
ENDCASE
```

Allerdings sieht man an diesem kurzen Stück schon, dass sich die Anweisungen ziemlich eintönig wiederholen. Das hat mir nicht gefallen. Eigentlich wird jedes Bit in eine Variable *rotiert*, und wenn man für eine Angabe im Telegramm genügend Bits zusammen hat, dann wird der Wert abgespeichert und das Spielchen beginnt von vorne.

Um die Bitfolge korrekt auszuwerten, schreibe ich eine Tabelle. Diese enthält die Bitpositionen im DCF-Telegramm, an denen ein neues Feld beginnt: etwa den Wert 21 für den Beginn der Minute.

```
create   dcfFields
0  c, \  .start
8  c, \
16 c, \  leading flags
17 c, \  daylight savings time
18 c, \
21 c, \  Minute
28 c, \  MinuteParity
29 c, \  Hour
35 c, \  HourParity
36 c, \  Day
42 c, \  DayOfWeek [1: Mo ... 7:So]
45 c, \  Month
50 c, \  Year-2000
58 c, \  Parity
59 c, \  .fin

16 Constant dcfFieldsN
Variable dcfFieldsI
Variable dcfCurr
ram create dcfValues dcfFieldsN allot rom
: dcf.NextField? ( dcfPos -- t/f )
  dcfFieldsI @ dcfFields + c@ 1- =
;
```

Die Variable `dcfFieldsI` enthält die Nummer des momentan aktiven Felds (beginnt mit 1), also etwa 6 am Beginn des Minutenfelds, und dient als Index. Das Wort `dcf.FieldComplete?` berechnet, ob das aktuelle Feld vollständig ist. Wenn ja, dann kann ich den bislang aufgesammelten Wert speichern, und zwar im Feld `dcfValues` unter dem gleichen Index. Das verplempert am Anfang ein paar Byte, aber die will man vielleicht später mal auswerten. Die Felder 0 und 1 mit den Werten 8 und 16 dienen lediglich dem Schutz von `dcfValues`, welches als Bytearray definiert wurde, und nicht als Array von Worten (`cells`).

```
6 dcfFieldsI !  ok
dcfFields dcfFieldsI @ + c@ . 28  ok
25 dcf.FieldComplete? . 0  ok
26 dcf.FieldComplete? . 0  ok
27 dcf.FieldComplete? . -1  ok
7 dcfFieldsI !  ok
28 dcf.FieldComplete? . -1  ok
```

`dcfFields` enthält die Anfangspositionen der Felder, weil ich dann die relativen Bitposition in diesem Feld einfach ausrechnen kann (beim Setzen der Bits in `dcfCurr`). Außerdem wird auf das Feld mit dem Index `dcfFieldsI-1` zugegriffen, daher setze ich konsequenterweise den Startwert auf 1.

Das Resultat des Wortes `dcf.bit` wird in einem neuen Wort `dcf.usebit` verwendet: der Bitwert wird an die richtige Bitposition in `dcfCurr` kopiert. Wenn ein Feld vollständig ist, wird der Wert von BCD in dezimal gewandelt und abgespeichert. Diese beiden Werte werden auch an der Stelle ausgegeben, bevor `dcfCurr` gelöscht wird.

```
: dcf.usebit ( b -- )
  CASE
    -2 OF ( sync59, reset Fields Index )
      1 dcfFieldsI !
    ENDOF
    -1 OF dcf.error.set       ( error ) ENDOF
    0 OF    ( bit value 0, do nothing ) ENDOF
    1 OF    ( bit value 1, use it     )
      \ get position of current bit in dcfCurr
      \ B: minute ones
      \ dcfPos = 21..24, dcfFieldsI = 6
      \ dcfFields[5] = 21
      \ bit pos in dcfCurr = 0..3
      dcfPos @
      dcfFields  dcfFieldsI @ 1-  +  c@
      -
      dcfCurr bset
    ENDOF
    ( default: ) dcf.error.set
  ENDCASE

  to-stdout space space
  dcfCurr @ dup .3r bcd>dec .3r

  dcfPos @ dcf.FieldComplete? IF
    dcfCurr @ bcd>dec
    dcfValues dcfFieldsI @ 1- + c!
    0 dcfCurr !
    1 dcfFieldsI +!
  ENDIF
;
```

Zur Belohnung erhalten wir ein komplettes Datentelegramm, hier noch von Hand kommentiert. Alle Felder wurden richtig ausgewertet. Seit 2003 wird in den ersten 15 Bits des Datentelegramms zusätzliche Information übertragen, davor waren dort alle Bits auf null gesetzt.

```
 0 100  59 -2  0  15  0   0  0   Sync
 9  91   0  0  0   1  0   0  0
18  82   1  1  0   1  0   2  2
18  82   2  1  0   1  0   6  6
```

```
 17  83   3  1  0   1  0  14 14
  8  92   4  0  0   1  0  14 14
  9  91   5  0  0   1  0  14 14
  8  92   6  0  0   1  0  14 14
  9  91   7  0  0   1 -1  14 14
 18  82   8  1  0   2  0   1  1
 18  82   9  1  0   2  0   3  3
 18  82  10  1  0   2  0   7  7
 17  83  11  1  0   2  0  15 15
  8  92  12  0  0   2  0  15 15
  9  91  13  0  0   2  0  15 15
  9  91  14  0  0   2  0  15 15
  9  91  15  0  0   2 -1  15 15
  9  91  16  0  0   3 -1   0  0
  9  91  17  0  0   4 -1   0  0
 18  82  18  1  0   5  0   1  1
  8  92  19  0  0   5  0   1  1
 18  82  20  1  0   5 -1   5  5
  8  92  21  0  0   6  0   0  0
  9  91  22  0  0   6  0   0  0
 18  82  23  1  0   6  0   4  4
  8  92  24  0  0   6  0   4  4
  9  91  25  0  0   6  0   4  4
 18  82  26  1  0   6  0  36 24
  9  91  27  0  0   6 -1  36 24   24 Minuten
  9  91  28  0  0   7 -1   0  0   P.ok
 18  82  29  1  0   8  0   1  1
 19  81  30  1  0   8  0   3  3
  8  92  31  0  0   8  0   3  3
  9  91  32  0  0   8  0   3  3
  8  92  33  0  0   8  0   3  3
 19  81  34  1  0   8 -1  35 23   23 Stunden
 19  81  35  1  0   9 -1   1  1   P.ok
  8  92  36  0  0  10  0   0  0
  8  92  37  0  0  10  0   0  0
  8  92  38  0  0  10  0   0  0
  8  92  39  0  0  10  0   0  0
 19  81  40  1  0  10  0  16 10
 17  83  41  1  0  10 -1  48 30   30 Tag
  8  92  42  0  0  11  0   0  0
 19  81  43  1  0  11  0   2  2
  8  92  44  0  0  11 -1   2  2   2 Dienstag
 19  81  45  1  0  12  0   1  1
  8  92  46  0  0  12  0   1  1
  9  91  47  0  0  12  0   1  1
  8  92  48  0  0  12  0   1  1
  8  92  49  0  0  12 -1   1  1   1 Monat
 19  81  50  1  0  13  0   1  1
 18  82  51  1  0  13  0   3  3
 17  83  52  1  0  13  0   7  7
  8  92  53  0  0  13  0   7  7
  8  92  54  0  0  13  0   7  7
  9  91  55  0  0  13  0   7  7
  8  92  56  0  0  13  0   7  7
  8  92  57  0  0  13 -1   7  7   07 Jahr
 19  81  58  1  0  14 -1   1  1   P.ok
  0 100  59 -2  0  15  0   0  0   Sync
```

Ergebnis: Dienstag 2007-01-30 23:24 h.

Das war jetzt ein ordentlicher Brocken, und daran habe ich auch einige Zeit gegrübelt. Und beim Schreiben des Artikels nochmal. Eine Konsequenz des Schreibens ist, dass man sich nochmal richtig Gedanken machen muss: Warum habe ich das jetzt so gelöst und nicht anders? Und gelegentlich stolpert man so auch über eine bessere Lösung.

Code 5: Paritätsbits auswerten

Um die Paritätsbits auszuwerten, brauchen wir ein paar Worte, die das Bit setzen, löschen, abfragen und umschalten. Bei jedem Datenbit, welches den Wert 1 hat, wird das interne Paritätsbit umgeschaltet. Wenn das zugehörige Paritätsbit aus dem Telegramm 1 ist, dann wird das interne Paritätsbit ebenfalls umgeschaltet. Danach muss das interne Paritätsbit zwangsläufig den Wert 0 haben.

Die Positionen der Paritätsbits im Datentelegramm habe ich im nachfolgenden Block hart kodiert. Wenn das interne Paritätsbit zu den angegebenen Stellen nicht 0 ist, wird `dcfError` gesetzt.

```
Variable dcfFlags
...
$01 Constant dcfParity

: dcf.par.set dcfParity dcfFlags bset ;
: dcf.par.clr dcfParity dcfFlags bclr ;
: dcf.par.tgl dcfParity dcfFlags 2dup
  btst IF bclr ELSE bset ENDIF ;
: dcf.par?    dcfParity dcfFlags btst ;

\ set dcf.error bit if parity bit is set
: dcf.err.if.par
  dcf.par? IF dcf.error.set ENDIF
  dcf.par.clr
;
: dcf.usebit
  ...
  CASE
    1 OF
      ...
      dcf.par.tgl
    ENDOF
    ( default: ) dcf.error.set
  ENDCASE

  dcfPos @
  CASE
    20 OF    dcf.par.clr ENDOF
    28 OF dcf.err.if.par ENDOF
    35 OF dcf.err.if.par ENDOF
    58 OF dcf.err.if.par ENDOF
  ENDCASE
  ...
;
```

Jetzt ist es an der Zeit, die empfangenen Werte anzuzeigen.

```
: get.DCF ( -- S M H d m Y )
  0                          \ fake second
  dcfValues  5  + c@         \ minute
  dcfValues  7  + c@         \ hour
  dcfValues  9  + c@         \ day
```

```
  dcfValues 11  + c@          \ month
  dcfValues 12  + c@ 2000 +   \ year
;
```

Das Wort `get.DCF` liest die Werte aus dem `dcfValues`-array und legt sie auf den Stapel. Zur Ausgabe benutze ich das schon vorhandene Wort `show.DT`. Die Ausgabe wird in dcf.tick gemacht, in dem Block `tick @ 0 = IF ... ENDIF`. Ein fehlerfreies Telegramm vorausgesetzt, erscheint jetzt auch ein ordentlich formatierter Zeitstempel in den Ausgaben:

```
...
  9  91  57   0   0    13  -1     7   7   0
  9  91  58   0   0    14  -1     0   0   0
  0 100  59  -2   0    15   0     0   0   0
20070201-224700
  9  91   0   0   0     1   0     0   0   0
  9  91   1   0   0     1   0     0   0   0
...
```

Danach kann man sich von den länglichen Debugausgaben verabschieden. Die habe ich mit einem weiteren Bitflag verziert. So kann man das bequem am laufenden Programm ein- und ausschalten.

```
Variable dcfFlags
...
$0f Constant dcfDebug
: dcf.D? dcfDebug dcfFlags btst ;

...
  dcf.D? IF
    to-stdout
    ...
  ENDIF
...
```

```
dcf error
dcf error
20070201-225100
20070201-225300
20070201-225400
20070201-225500
20070201-225600
...
```

Code 6: timeup-Uhr stellen

Jetzt sind alle Zutaten bereit, um die vom DCF-Signal gewonnene Zeit auf die `timeup`-Uhr zu übertragen. Dazu führe ich ein weiteres Bit namens `dcfCommit` ein. Dieses wird von `dcf.init` und danach jede Stunde von `job.hour` gesetzt. Am Ende von `dcf.tick` wird die Zeit aus dem DCF-Signal auf die Zähler von `timeup` kopiert, wenn Sekunde 59 erkannt wurde und wenn `dcfCommit` gesetzt ist und wenn das Telegramm fehlerfrei war. Dabei muss man bedenken, dass die `timeup`-Uhr die Zähler für Tag und Monat um Eins erniedrigt benutzt. Danach wird `dcfCommit` gelöscht.

```
: dcf.commit ( -- )
  get.DCF
  year     !
  1- month !
  1- day   !
  hour     !
  min      !
  sec      ! \ zero
;

: dcf.init
  ...
  dcfCommit dcfFlags bset \ commit requested
;
: dcf.tick
  ...
  tick @ 0 = IF
    ...
    -2 = IF \ sync detected
      to-stdout cr
      dcf.error? invert IF
        get.DCF
        show.DT
        dcfCommit dcfFlags btst IF
          dcfCommit dcfFlags bclr
          dcf.commit
        ENDIF
      ELSE
        ." dcf error "
      ENDIF
      dcf.error.clr
      0 dcfPos !
      0 dcfCurr !
    ELSE
      1 dcfPos +!
    ENDIF
  ENDIF
;
...
: job.hour
  dcfCommit dcfFlags bset
;
```

Damit haben wir eine einigermaßen funktionierende DCF-Uhr. Der Code belegt etwa 3520 Bytes (nach savesystem). *Einigermaßen* bedeutet, dass unter ungünstigen Bedingungen das DCF-Signal nicht korrekt lesen kann und die Uhr lange Zeit braucht, bis sie sich gestellt hat.

Code 7: dcfOffset

Wenn man zu lang das Blinken der LEDs und die Anzeige der Uhrzeit anstarrt, dann findet man, dass die `timeup`-Uhr zu langsam (oder zu schnell, abhängig vom Quarz) läuft. Die Fabrikationstoleranz der Quarze zeigt sich hier deutlich. Daher habe ich ein Offset eingeführt, welches die Verschiebung des DCF-Signals gegen die `timeup`-Uhr beinhaltet (maximal 100 ticks, also 1 Sekunde). Die Verschiebung beträgt bei mir ca. 0.15 Sekunden in der Stunde. Beim Übertragen der DCF-Zeit wird diese Verschiebung ausgeglichen, siehe Entscheidung 4.

Der Aufwand ist beträchtlich. Zunächst sind 2 Variablen zu definieren: `dcfOffset` enthält die Verschiebung der DCF-Sekunde gegen die `timeup`-Sekunde. Die erlaubten Werte sind `0` bis `ticks.sec-1`. Alle Abfragen von der Bauart `tick @ [Zahl] = IF ...` werden geändert in `tick @ [Zahl] dcfOffset @ + ticks.sec mod =` Wichtig ist dabei die Modulo–Operation, denn negative oder zu große Werte erreicht `tick` nun mal nicht.

In `dcf.tick` wird nach dem Hochzählen der Wert von `dcfPulse` in `dcfPulse1` gespeichert, falls `dcfOffset1` ticks verstrichen sind (Aufruf von `dcfCountPP`). Damit kann ich feststellen, ob der DCF-Puls am Anfang der mit `dcfOffset` festgelegten Sekunde liegt. Falls nicht, wird `dcfOffset` angepasst (Aufruf von `dcf.check.offset`).

Wenn man `dcfOffset` um `1` erhöht, dann wird der mit `tick @ dcfOffset @ ticks.sec mod = IF ... ENDIF` eingefasste Block beim nächsten tick wieder ausgeführt, obwohl nur ein einziger tick verstrichen ist. Um das zu vermeiden, bekommt `dcf.bit` einen weiteren Rückgabewert, welcher diesen Fall anzeigt. In `dcf.tick` wird alles nach dem Aufruf von `dcf.bit` entsprechend abgesichert: `dup -3 <> IF ... ELSE drop ENDIF`.

```
Variable dcfOffset
Variable dcfPulse1
21 Constant dcfOffset1

: dcf.commit ( -- )
  0 tick !
  0 dcfOffset !
  get.DCF
  ...
  ."  commited "
;

: dcf.reload
  ...
  0 dcfPulse1 !
;

: dcf.bit ( pulse pause -- bit/error )
  \ return values:
  \ -3 interval too short
  \ -2 sync detected
  \ -1 error
  \  0 bit value 0
  \  1 bit value 1
  \ limits hardcoded assuming 10 ms per count
  2dup + 97 > IF
    drop
    dup  2 < IF ( ." sync59 " ) -2 ELSE
    dup  6 < IF dcf.error.set   -1 ELSE
    dup 11 < IF ( ." bit:0 "  )  0 ELSE
    dup 16 < IF dcf.error.set   -1 ELSE
    dup 21 < IF ( ." bit:1 "  )  1 ELSE
    dcf.error.set             -1
    ENDIF ENDIF ENDIF ENDIF ENDIF
    swap drop
  ELSE
    2drop -3
  ENDIF
;

: dcf.usebit ( b -- )
  CASE
    ...
    1 OF    ( bit value 1, use it )
      dcf.error? invert IF
        \ position of current bit in dcfCurr
        ...
        dcf.par.tgl
      ENDIF
    ENDOF
    ...
  ENDCASE

  dcfPos @ dcf.FieldComplete? IF
    dcfCurr @ bcd>dec
    dcfValues dcfFieldsI @ 1- + c!
    0 dcfCurr !

    dcfFieldsI @ 1+
    dup dcfFieldsN < invert IF drop 1 ENDIF
    dcfFieldsI !
  ENDIF
;

: dcfCountPP
  tick @ dcfOffset1 dcfOffset @ + ticks.sec mod =
  IF dcfPulse @ dcfPulse1 ! ENDIF
;

: dcf.tick
  dcf.readPin

  \ show pin "active" on led2
  dup IF led2_1 ELSE led2_0 ENDIF

  \ count up Pulse/Pause counters
  IF    1 dcfPulse +!
  ELSE  1 dcfPause +!
  ENDIF
  dcfCountPP

  \ reload counters if tick == 0
  tick @ ( 0 ) dcfOffset @ ( + ) ticks.sec mod = IF
    dcfPulse @ dcfPause @ dcf.bit

    dup -3 <> IF \ unless interval too short

      dup dcf.dbg.out  \ print counters

      dup dcf.usebit   \ print dcfCurr bcd, dec
      dup 0 >= IF      \ check dcfOffset if valid
        dcf.check.offset
      ENDIF
      dcf.reload       \ clear counters

      -2 = IF \ sync detected
```

```
        to-stdout cr
        dcf.error? invert IF
          get.DCF
          show.DT
          dcfPos @ 59 =
          dcfCommit dcfFlags btst
          and IF
            dcfCommit dcfFlags bclr
            dcf.commit
          ENDIF
        ELSE
          ." dcf error "
        ENDIF
        dcf.error.clr
        0 dcfPos !
        0 dcfCurr !
        1 dcfFieldsI !
      ELSE
        \ 1 dcfPos +!
        \ assert 0 .. 59!
        dcfPos @ 1+ 60 mod dcfPos !
      ENDIF
    ELSE
      drop
    ENDIF
  ENDIF
;
```

Das ist nochmal ein ordentlicher Brocken. Man kann das bestimmt auch anders machen, wie alles. Aber so hat es mir am besten gefallen. Das Programm belegt 3722 Byte und nach `rom ’ run is bootmessage ram savesystem` 4006 Byte. Wenn die Uhr noch irgendetwas anderes können soll, dann ist wohl erst mal Code aufräumen angesagt.

Ausblick

Ein paar Dinge könnte man jetzt noch tun:

- die Debugausgaben aus dem Programm entfernen, um Platz zu schaffen
- man könnte den Zustand von `dcfError` auf dem LC-Display anzeigen
- Geschickt wäre auch ein Zeichen, welches die Sommerzeit anzeigt
- Schaltsekunden werden derzeit nicht berücksichtigt
- dcfOffset auf dem LCDisplay anzeigen `lcdpos 1 0 dcfOffset @ s>d 4 d.r` in `job.sec`

Möglicherweise könnte man dem Kontroller einen externen Uhrenquarz oder Oszillator spendieren, um das Problem mit der Drift zu lösen.

Diese Uhr ist *nur eine Uhr* — keine DCF-synchronisierte Hauszentrale. Dazu reicht leider der Platz nicht so recht, selbst wenn man die nicht mehr benötigten Teile des Programms entfernt. Auch die Geschichte mit dem Sonnenscheinsensor muss noch etwas warten. Ergänzungen, Kommentare, Korrekturen sind ausdrücklich erwünscht. Sie erreichen mich unter `ew.forth@nassur.net`

Referenzen

1. E. Wälde, Adventures in Forth, Die 4. Dimension 3/2006, Jahrgang 22
2. E. Wälde, Adventures in Forth 2, Die 4. Dimension 4/2006, Jahrgang 22
3. E. Wälde, Adventures in Forth 3, Die 4. Dimension 1/2007, Jahrgang 23
4. Renesas R8C/13 datasheet auf `www.renesas.com`
5. `http://de.wikipedia.org/wiki/DCF77` und Verweise darin

Listings

```
\ 2007-01-16 EW adv4_main.fs

rom
include adv4_redir.fs
+redir
include adv4_tasker.fs
include adv4_timeup.fs
include adv4_lcd.fs

: get.DT ( -- S M H d m Y )
  sec @     min   @     hour  @
  day @ 1+  month @ 1+  year  @
;
: show.DT ( S M H d m Y -- )
  p4! type  p2! type  p2! type
  s" -" type
  p2! type  p2! type  p2! type
;
$e8   Constant portDCF \ port-D
$ea   Constant pddrDCF \ pddr-D
5     Constant pinDCF
: led2_0 ( -- ) 2 port1 bclr ;
: led2_1 ( -- ) 2 port1 bset ;
include adv4_dcf.fs

: clock.init
  2000 year !
  1    1- month !
  1    1- day !
  0    hour !
  0    min  !
  0    sec  !
;
: led3_0 ( -- ) 3 port1 bclr ;
: led3_1 ( -- ) 3 port1 bset ;
: job.tick
  tick @ ticks.sec 4 / = IF led3_0 ENDIF
  tick @  0            = IF led3_1 ENDIF
  dcf.tick
;
```

```
: job.sec
  0 13 lcdpos sec @ p2! to-lcd type
  1 0  lcdpos dcfOffset @ s>d 4 d.r
  to-stdout
;

: job.min
  to-lcd
  0 0 lcdpos get.DT show.DT
  to-stdout
;
: job.hour
  dcfCommit dcfFlags bset
;
: job.day
;
: job.month
  year @
  month @ 1+ \ month offset 1 correction
  length_of_month
  4 Limits + c!
;
: job.year
;
: loop.init ( -- )
  clock.init
  0 tick !
  timer @ cycles.tick + newtimer !
  lcdpage get.DT to-lcd show.DT to-stdout
  job.sec
  dcf.init
;
create Jobs   ' job.tick ,
  ' job.sec , ' job.min ,   ' job.hour ,
  ' job.day , ' job.month , ' job.year ,

: run
  task
  loop.init
  BEGIN
    tickover? IF
      timeup    \ Zaehler aktualisieren
    ENDIF
    7 0 DO
      I tuFlags btst IF
        I cells Jobs + @ execute
        I tuFlags bclr
      ENDIF
    LOOP
    pause
  AGAIN
;
ram
```

```
\ 2007-01-16 EW adv4_dcf.fs

Variable dcfFlags
$00 Constant dcfError
$01 Constant dcfParity
$02 Constant dcfCommit
$0f Constant dcfDebug
Variable dcfPulse
Variable dcfPause
Variable dcfPos
Variable dcfOffset
Variable dcfPulse1
21 Constant dcfOffset1
create   dcfFields
0  c, \  1  .start
8  c, \  2
16 c, \  3  leading flags
17 c, \  4  daylight savings time
18 c, \  5
21 c, \  6  Minute
28 c, \  7  MinuteParity
29 c, \  8  Hour
35 c, \  9  HourParity
36 c, \ 10  Day
42 c, \ 11  DayOfWeek [1: Mo ... 7:So]
45 c, \ 12  Month
50 c, \ 13  Year-2000
58 c, \ 14  Parity
59 c, \ 15  .fin

16 Constant dcfFieldsN
Variable dcfFieldsI
Variable dcfCurr
ram create dcfValues dcfFieldsN allot rom
: dcf.FieldComplete? ( dcfPos -- t/f )
  dcfFieldsI @ dcfFields + c@  1-  =
;
: dcf.error.set  dcfError dcfFlags bset ;
: dcf.error.clr  dcfError dcfFlags bclr ;
: dcf.error?     dcfError dcfFlags btst ;
: dcf.par.set dcfParity dcfFlags bset ;
: dcf.par.clr dcfParity dcfFlags bclr ;
: dcf.par.tgl dcfParity dcfFlags 2dup
  btst IF bclr ELSE bset ENDIF ;
: dcf.par?    dcfParity dcfFlags btst ;
: dcf.D? dcfDebug dcfFlags btst ;

: bcd>dec ( n.bcd -- n.dec )
  $ff and
  dup
  4 rshift 10 * \ extract high nibble as 10s
  swap
  $0f and       \ extract low  nibble as 1s
  +             \ add
;
: get.DCF
  0                          \ fake Second
  dcfValues  5  + c@         \ Minute
  dcfValues  7  + c@         \ Hour
  dcfValues  9  + c@         \ Day
  dcfValues 11  + c@         \ Month
  dcfValues 12  + c@ 2000 +  \ Year
;
: dcf.commit ( -- )
  0 tick !
  0 dcfOffset !
  get.DCF
  year     !
  1- month !
  1- day   !
  hour     !
  min      !
  sec      ! \ zero
  ."  commited "
;
: dcf.readPin
  \ pin is on "active low" connection
  \ thus invert
```

```
  pinDCF portDCF btst invert
;
: dcf.reload
  0 dcfPulse  !
  0 dcfPause  !
  0 dcfPulse1       !
;
: dcf.init
  pinDCF pddrDCF bclr     \ set pinDCF input
  0 dcfPos  !
  1 dcfFieldsI !
  dcf.error.set    \ error unless proven ok.
  dcfDebug dcfFlags bset  \ debug off
  dcfCommit dcfFlags bset \ commit requested
;
: dcf.bit ( pulse pause -- bit/error )
  \ return values:
  \ -3 interval too short
  \ -2 sync detected
  \ -1 error
  \  0 bit value 0
  \  1 bit value 1
  \ limits hardcoded assuming 10 ms per count
  2dup + 97 > IF
    drop
    dup  2 < IF ( ." sync59 " ) -2 ELSE
    dup  6 < IF dcf.error.set   -1 ELSE
    dup 11 < IF ( ." bit:0 "  )  0 ELSE
    dup 16 < IF dcf.error.set   -1 ELSE
    dup 21 < IF ( ." bit:1 "  )  1 ELSE
    dcf.error.set             -1
    ENDIF ENDIF ENDIF ENDIF ENDIF
    swap drop
  ELSE
    2drop -3
  ENDIF
;
: .3r  s>d 3 d.r space ;
: dcf.err.if.par
  dcf.par? IF dcf.error.set ENDIF
  dcf.par.clr
;
: dcf.usebit ( b -- )
  CASE
    -2 OF ( sync59, reset Fields Index )
      1 dcfFieldsI !
    ENDOF
    -1 OF dcf.error.set       ( error ) ENDOF
    0 OF  ( bit value 0, do nothing ) ENDOF
    1 OF    ( bit value 1, use it )
      dcf.error? invert IF
        \ position of current bit in dcfCurr
        \ B: minute ones
        \ dcfPos = 21..24, dcfFieldsI = 6
        \ dcfFields[5] = 21
        \ bit pos in dcfCurr = 0..3
        dcfPos @
        dcfFields  dcfFieldsI @ 1-  +  c@
        -
        dcfCurr bset
        dcf.par.tgl
      ENDIF
    ENDOF
    ( default: ) dcf.error.set
  ENDCASE

  dcfPos @
  CASE
    20 OF    dcf.par.clr ENDOF
    28 OF dcf.err.if.par ENDOF
    35 OF dcf.err.if.par ENDOF
    58 OF dcf.err.if.par ENDOF
  ENDCASE

  \ debug output
  dcf.D? IF
    to-stdout
    space space
    dcfCurr @ dup .3r bcd>dec .3r
    dcf.par? .3r
  ENDIF

  dcfPos @ dcf.FieldComplete? IF
    dcfCurr @ bcd>dec
    dcfValues dcfFieldsI @ 1- + c!
    0 dcfCurr !
    \ 1 dcfFieldsI +! \ too simple
    dcfFieldsI @ 1+
    dup dcfFieldsN < invert IF drop 1 ENDIF
    dcfFieldsI !
  ENDIF
;
: dcf.dbg.out ( bit -- )
  dcf.D? IF
    to-stdout cr
    ." ("
    dcfPulse1 @         .3r
    dcfPulse @ dcfPulse1 @ - .3r
    ." ) "
    dcfPulse @ .3r   \ dcfPulse
    dcfPause @ .3r   \ dcfPause
    dcfPos   @ .3r   \ dcfPos
               .3r   \ bit value
    dcf.error? .3r   \ error flag
    space space
    dcfFieldsI @ .3r \ dcfFieldsI
                     \ dcf.FieldComlete?
    dcfPos @ dcf.FieldComplete? .3r
    dcfOffset @ .3r  \ dcfOffset
  ENDIF
;
: dcfCountPP
  tick @ dcfOffset1 dcfOffset @ + ticks.sec mod =
  IF dcfPulse @ dcfPulse1 ! ENDIF
;
: dcf.inc.offset
  dcfOffset @ 1+ ticks.sec mod  dcfOffset !
;
\ : dcf.dec.offset
\   dcfOffset @ 1- ticks.sec mod  dcfOffset !
\ ;
: dcf.check.offset
  dcfPulse1 @ dcfPulse @ < IF
    ."  dcfOffset++"
    dcf.inc.offset
  ENDIF
;
: dcf.tick
  dcf.readPin

  \ show pin "active" on led2
  dup IF led2_1 ELSE led2_0 ENDIF
```

```
211
212    \ count up Pulse/Pause counters
213    IF    1 dcfPulse +!
214    ELSE  1 dcfPause +!
215    ENDIF
216    dcfCountPP
217
218    \ reload counters if tick == 0
219    tick @  dcfOffset @  ticks.sec mod = IF
220      dcfPulse @ dcfPause @ dcf.bit
221
222      dup -3 <> IF \ unless interval too short
223
224        dup dcf.dbg.out  \ print counters
225
226        dup dcf.usebit   \ print dcfCurr bcd
227        dup 0 >= IF      \ check offset
228          dcf.check.offset
229        ENDIF
230        dcf.reload       \ clear counters
231
232        -2 = IF \ sync detected
233          to-stdout cr
234          dcf.error? invert IF
235            get.DCF
236            show.DT
237            dcfPos @ 59 =
238            dcfCommit dcfFlags btst
239            and IF
240              dcfCommit dcfFlags bclr
241              dcf.commit
242            ENDIF
243          ELSE
244            ." dcf error "
245          ENDIF
246          dcf.error.clr
247          0 dcfPos !
248          0 dcfCurr !
249          1 dcfFieldsI !
250        ELSE
251          \ 1 dcfPos +!
252          \ assert 0 .. 59!
253          dcfPos @ 1+ 60 mod dcfPos !
254        ENDIF
255      ELSE
256        drop
257      ENDIF
258    ENDIF
259  ;
```

```
1   \ 2007-01-16 EW adv4_timeup.fs
2
3   Variable tuFlags
4   Variable newtimer
5   Variable lastsec
6   10  Constant cycles.tick \ timerC cycles/tick
7   100 Constant ticks.sec   \ ticks/second
8
9   ram
10  create Counts &7 cells allot
11  create Limits ticks.sec c, 60 c, 60 c,
12                24 c, 31 c, 12 c,
13  rom
14  create MaxDay 31 c, 28 c, 31 c, 30 c,
15                31 c, 30 c, 31 c, 31 c,
16                30 c, 31 c, 30 c, 31 c,
```

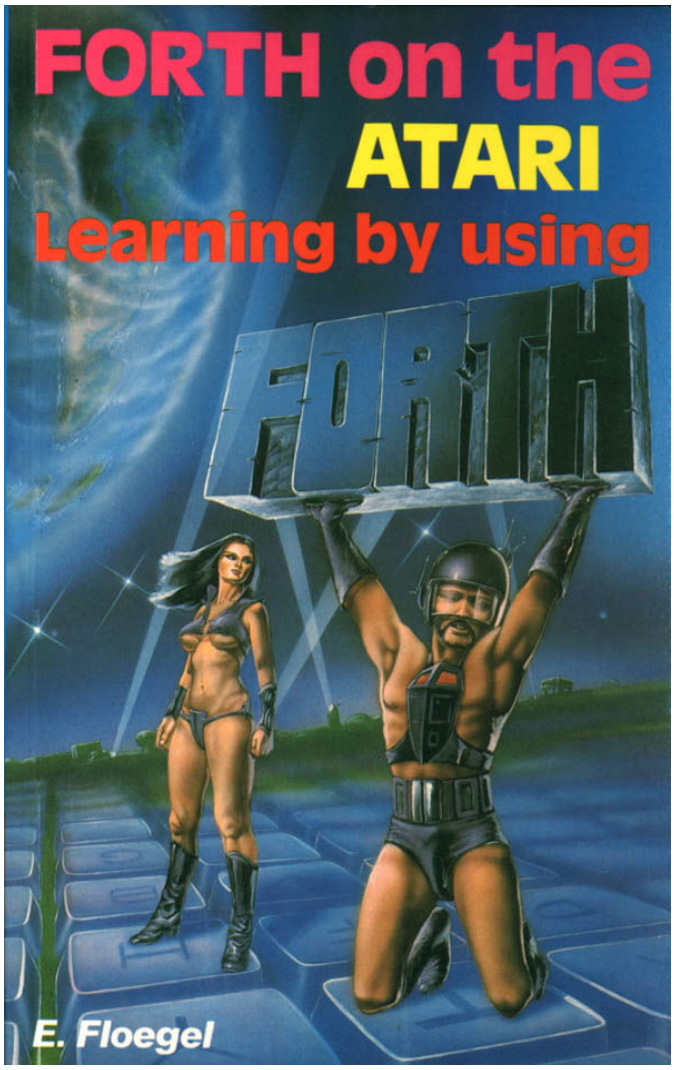
```
17
18  : tick  ( -- addr )         Counts   ;
19  : sec   ( -- addr ) 1 cells Counts + ;
20  : min   ( -- addr ) 2 cells Counts + ;
21  : hour  ( -- addr ) 3 cells Counts + ;
22  : day   ( -- addr ) 4 cells Counts + ;
23  : month ( -- addr ) 5 cells Counts + ;
24  : year  ( -- addr ) 6 cells Counts + ;
25
26  : tickover? ( -- ) newtimer @ timer @ -  0< ;
27
28  \ --> Stephen Pelc, chapter 6
29  : leap_year ( year -- t/f )
30    dup    4 mod 0=
31    over 100 mod 0<> and
32    swap 400 mod 0=  or
33  ;
34  : length_of_month ( year month -- maxday )
35    dup 1-                \ array starts at 0
36    MaxDay + c@
37    swap 2 = IF           \ if month == 2
38      swap leap_year IF   \ and leap_year
39        1+                \   month += 1
40      ENDIF
41    ELSE
42      swap drop           \ remove year
43    ENDIF
44  ;
45  : timeup ( -- )
46    cycles.tick newtimer +!
47    0 tuFlags bset              \ tickflag++
48    1         Counts   +!       \ tick++   <-- INC!
49
50    \ for i in tick sec min hour day month
51    6 0 DO
52      I cells Counts +  @
53      I       Limits + c@ >= IF  \ if C[i] >= L[i] ?
54        0 I cells Counts + !     \   C[i]=0;
55        I 1+ tuFlags bset        \   F[i+1]++;
56        1 I 1+ cells Counts + +! \   C[i+1]++; < INC
57      ENDIF                      \ endif
58    LOOP
59  ;
```

```
1   \ 2007-01-16 EW adv4_lcd.fs
2
3   \ always display a sign
4   : sign! 0 < IF 45 ELSE 43 ENDIF hold ;
5   \ always display 4,2 digits
6   : p4!  s>d <# # # # # #> ;
7   \ : p3!  s>d <# # # # #> ;
8   : p2!  s>d <# # # #> ;
9   \ start next lcdtype at lcdposition
10  \ row (0,1) col (0..15)
11  : lcdpos ( row col -- )
12    swap $40 * + $80 + lcdctrl! &1 ms
13  ;
```

```
1   \ 2007-01-16  EW  adv4_redir.fs
2   \ poor mans redirection
3   \ (stdout, lcd, uart0)
4
5   Variable 'type  \ pointer to correct "type"
6   Variable 'emit  \ pointer to correct "emit"
7   Variable 'cr    \ pointer to correct "cr"
8
```

```
: >type ( -- ) 'type @ execute ;
: >emit ( -- ) 'emit @ execute ;
: >cr   ( -- ) 'cr   @ execute ;

: +redir ( -- )
  ['] >type is type
  ['] >emit is emit
;
: -redir ( -- )
  ['] (type) is type
  ['] (emit) is emit
;
: to-stdout ( -- )
  ['] (type)    'type !
  ['] (emit)    'emit !
  ['] cr        'cr   !
;
to-stdout   \ default

: to-lcd ( -- )
  ['] lcdtype   'type !
  ['] lcdemit   'emit !
  ['] lcdcr     'cr   !
;
\ : to-uart0 ( -- )
\   ['] uart0type 'type !
\   ['] uart0emit 'emit !
\   ['] uart0cr   'cr   !
\ ;
```

```
\ 2007-01-16 EW adv4_tasker.fs

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

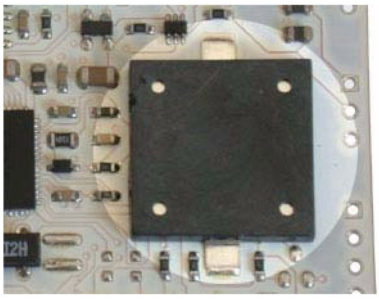


Bucheinband des 1983 bei Blue Cat/Hofacker erschienenen Buchs *FORTH on the Atari* von Ekkehard Floegel (ISBN 978-0936200385)

Der Piezo–Summer des AVR–Butterfly

Ulrich Hoffmann, Michael Kalus

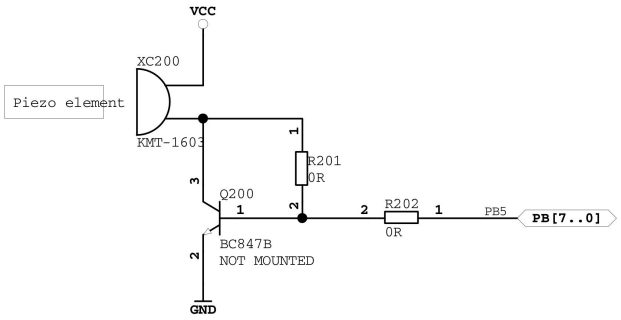
Für die Ausgabe von Tönen hat der AVR–Butterfly einen Piezosummer. Dieser ist an PORTB5 angeschlossen. Benutzt man PWM (Pulsweiten–Modulation), lassen sich verschiedene Frequenzen generieren, um Melodien zu spielen.



Das Piezo–Element auf dem Butterfly–Board

Dieser Port–Pin 5 dient neben der normalen I/O–Funktion auch als Output für den PWM–Timer. Er ist daher zur Ausgabe von Frequenzen besonders gut geeignet. Deswegen haben ihn die Atmel–Ingenieure wohl auch als Anschluss für den Piezo–Summer auf dem AVR–Butterfly–Board verwendet.

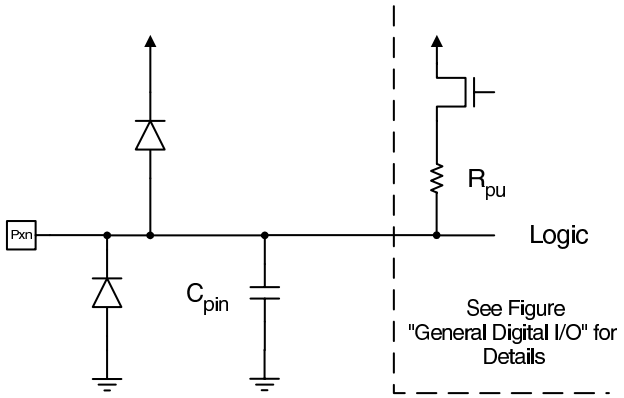
Aber auch durch einfaches Umschalten des Pins lassen sich dem Summer Töne entlocken: Siehe Forth–Beispiel–Code im Listing auf Seite 27



Ansteuerungsschaltung des Piezo–Elements

Funktion der Port–Pins

Die Output–Pins auf dem ATmega169 können alle einzeln gesteuert werden. Direkt mittels der `SBI`– und `CBI`–Befehle oder durch Beschreiben/Lesen des zugehörigen Data–Registers. Jeder Pin verfügt über hi–sink & hi–source–Eigenschaften und ist daher in der Lage, eine LED zu treiben. Auch hat jeder Pin einen eigenen pull–up–Widerstand mit Vcc–unabhängigem Wert sowie Schutzdioden gegen Vcc und Gnd.



Interne Schaltung der ATmega169 Ein/Ausgabe–Pins

Nomenklatur:
Registername, Portbuchstabe x, Pin Nummer n

DRB5	Port B data register pin 5;	w/r
DDRB5	Port B data direction register pin 5;	w/r
pinB5	Port B pin register B pin 5;	read only

Jeder Port — und damit jeder Pin einzeln auch — besteht aus 3 Registern. Dem Data–Register DRx, dem Data–Direction–Register DDRx und dem Port–Input–Register PINx. Die Pins können einzeln Eingang oder Ausgang sein. Das PINx spielt eine besondere Rolle, es kann nur eingelesen werden. Das geschieht aber über die gleichen Pins wie beim DRx.

Eine den ganzen Portx betreffende Kontrolle besteht über das MCUCR. Wird dort das pull–up–disable–bit (PUDbit) gesetzt, sind alle Pins des Ports ohne pull–up–Widerstand.

Jeder Pin eines Ports ist mehrfach belegt. Unser PORTB5 hat neben dem I/O noch die Funktionen OC1A/PCINT13, Bit 5.

OC1A, Output–Compare–Match–A–output: Der PB5–Pin dient zur Ausgabe für den Timer/Counter1 Output–Compare–A. Dazu muss der Pin als Output (DDB5 set (one)) gesetzt sein. Dann ist der OC1A der Ausgabepin für die PWM–mode–timer–function. Und schließlich dient er noch als PCINT13, Pin–Change–Interrupt–Source 13: Der PB5–Pin ist dann Eingang für eine externe Interrupt Quelle.

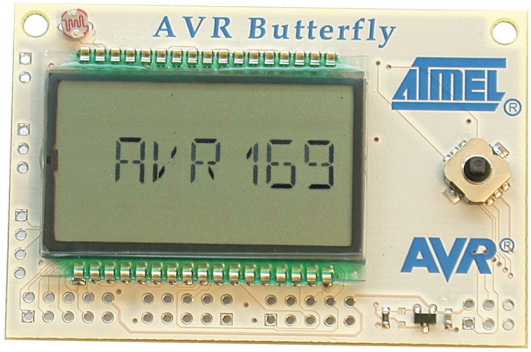
Über einen Multiplexer können diese alternativen Funktionen zugewiesen werden. Siehe Datenblatt des ATmega169.

Links

AVR–Butterfly Manual:	`http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3146`
Piezo:	`http://www.forth-ev.de/wiki/doku.php/projects:avr:pietzo_speaker`
Datenblatt ATmega169:	`http://www.atmel.com/` Suche: ATmega169

Schaltungen und Foto entstammen den Atmel–Datenblättern. Sie sind hier nur zur Illustration wiedergegeben.

```
\ Butterfly Piezo

25 constant portB     \ used with c@ c!   20 higher than I/O-port
24 constant ddrB      \ see page 22 I/O Memory in ATmega169 manual
23 constant pinB
20 constant piezoMask \ uses pins PB5

: piezo-init ( -- )
   ddrB c@  piezoMask or ddrB c! ;

piezo-init

: click ( -- )  portB c@ piezoMask or portB c! ;
: clack ( -- )  portB c@ piezoMask not and portB c! ;

variable fudge  5 fudge !

: delay ( n -- )
   begin ?dup while  fudge @
       begin ?dup while 1- repeat
       1-
   repeat ;

: tone ( dur 1/pitch -- )
     dup >r / r>  \ quotient * pitch = duration
     begin
       over
     while
       click dup delay
       clack dup delay
       swap 1- swap
     repeat
     drop drop ;

\ name your tones

: beep ( -- )  100 3 tone ;
```



Gehaltvolles

zusammengestellt und übertragen von *Fred Behringer*

Vorbemerkung des Rezensenten

Totgesagte leben länger. Diesmal geht es nicht um Forth, sondern um das holländische Verkündungsblatt von Forth, het Vijgeblaadje. Kaum haben wir in der Besprechung von Vijgeblaadje 59 davon berichtet, dass Ron Minke sich gezwungen sieht, die Redaktion abzugeben und einen Nachfolger sucht, der sich leider nicht finden lässt – und schon liegt eine weitere Ausgabe des Vijgeblaadjes vor. Bestritten von zwei Aktiven der HCC–Fgg. Als Inhalt zwei interessante Artikel, aber leider kein Wort darüber, wer die Herausgabe, die ja einen Haufen Arbeit macht, bewältigt hat und überhaupt, wer ab jetzt die Verantwortung für die Redaktion übernimmt. Ähnlichkeiten mit der FG sind rein zufälliger Natur.

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 60, Februar 2007

Heeft je Forth geen floating point...? — Albert Nijhof

Übersetzung des Vorspanns: „In Forth–Steuerungsprogrammen kann man Gleitkomma–Zahlen meistens dadurch umgehen, dass man die Größen geeignet skaliert. Mit ganzen Zahlen kommt man so enorm weit, aber manchmal ist es doch etwas lästig. Unter dem Motto *Probier es mal mit Brüchen* habe ich ein paar kleine Programme angefertigt, die in solchen Fällen vielleicht nützlich sein können. Sie finden sich unter `http://home.hccnet.nl/anij`."
(Red.: Siehe auch Abbildung 1.)

Soweit der Autor. Im Rest des Artikels gibt er kurze Beschreibungen der Programme.

Timers en pseudo–multitasking — Willem Ouwerkerk

Übersetzung des Vorspanns: „Im Vijgeblaadje 57 und 58 hat Ron Minke die Implementation von Timern auf Mikro–Controllern behandelt, die eine Anzahl von nicht gleichlaufenden Prozessen genau und gleichzeitig steuern sollen. Sowas habe ich kürzlich auch gemacht, aber ich habe es anders angepackt. Ron wählte Maschinencode, Interrupts und definierende Worte und knetete daraus ein prächtiges komplexes Gebilde. Aber das lässt sich auch einfacher bewältigen."

Soweit der Autor. Es folgt dann eine Beschreibung seines Programmes. Im Wesentlichen geht es um die kunstvolle Ansteuerung von LEDs.

Brüche ausgeben	DOT.F	DOT-SHORT.F	DOT-LAZY.F
Allgemeine Rechenoperationen	ARITH.F		
Die Quadratwurzel	SQR.F		
Sin, Cos, Tan	CORDIC.F16	TAYLOR.F16	TAYLOR.F32

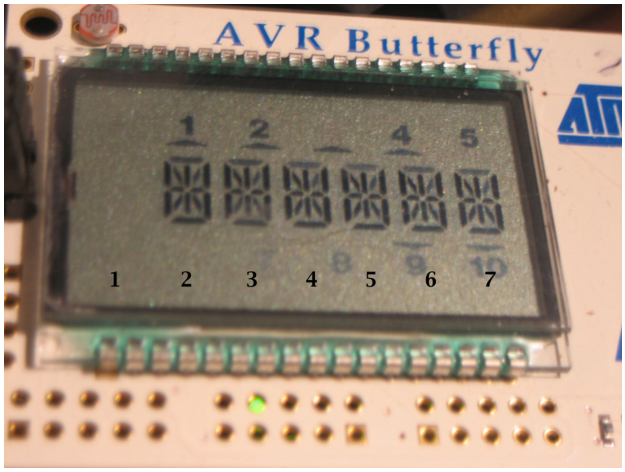
Abbildung 1: Brucharithmetik in Forth unter `http://home.hccnet.nl/anij`

Das Display auf dem AVR–Butterfly mit amforth ansteuern

Michael Kalus

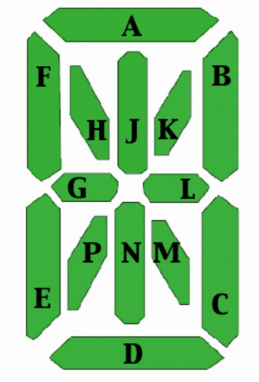
Der AVR–Butterfly ist komplett ausgerüstet, um den AVR–Microcontroller ATmega169 zu erkunden und dessen Fähigkeiten darzustellen. Mit Hilfe des amforth kann das AVR–Butterfly interaktiv erkundet werden. Beispiele dafür findet ihr im Forth-eV Wiki `www.forth-ev.de/wiki`. Mir hat es Spass gemacht zu sehen, wie das Display betrieben wird. Die Einzelheiten sind beschrieben im doc2530.pdf, dem LCDdriver–Dokument von Atmel.

Der ATmega169 hat den Display–Treiber integriert. Das Display auf dem AVR–Butterfly ist sehr einfach. Es kann 6 Zeichen darstellen, zwei Doppelpunkte und einige Rauf–Runter–Pfeile. Auch einige der Nummernsymbole können angesteuert werden. Folgende Segmente sind angeschlossen (mehr Treiber hat der ATmega169 nicht):



Das Display auf dem AVR–Butterfly–Board

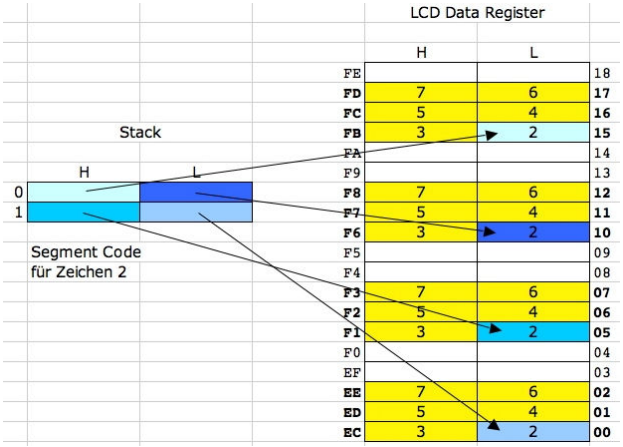
Jedes Zeichen besteht aus 14 Segmenten. Die Zeichen Nr. 2 bis Nr. 7 können benutzt werden. Die Segmente werden einzeln geschaltet von den Bits in den LCD–Data–Registern LCDDR0..18 im ATmega169. Diese Register sind in 4 Gruppen angeordnet, je *common backplane* eine Gruppe.



Ein Display–Zeichen besteht aus 14 Segmenten.

Die Anordnung der Leitungen hat eine eigenartige Verteilung dieser Nibbles im LCD–Data–Register zur Folge. Vom Basisregister aus sind die Nibbles mit einem Offset von 0, 5, 10, 15 in den Registern aufgestapelt.

Die Zeichen Nr. 3, 5, 7 werden von den oberen Nibbles, also Bits 7..4 eines Data–Byte–Stapels geschrieben. Die Zeichen Nr. 2, 4, 6 werden mit den unteren, also Bits 3..0, geschreiben. Dabei codieren immer 4x4 Nibbles eines Stapels ein ganzes Zeichen.



Aus einer Segment–Code–Tabelle kann entnommen werden, welche Segmente an sein müsssen, um ein ASCII–Zeichen darzustellen. Der Segment–Code braucht 14bit. Somit kann ein Datenwort im Flash des ATmega169 ein ganzes Zeichen codieren (cell=16Bit). Diese look-up-table ist im Beispielcode angegeben.

Gesteuert wird der Displaytreiber über 4 Control–Register im ATmega169, wie Tabelle 2 auf Seite 30 zeigt.

Mein Beispiel zeigt, wie man Text und Ziffern im Display des Butterfly zeigen kann. Es wurde der Ansatz verfolgt, die Zeichen zunächst in einen Ausgabepuffer zu schreiben von dem aus das LCD Display geladen werden kann. Das gibt die Möglichkeit den Ursprung der Zeichen später frei bestimmen zu können. So können nun Zeichen aus dem regulären Ausgabestrom dorthin umgelenkt werden mit der Phrase: `<lcd> ." hallo" </lcd>` oder ähnlichem. Oder es wird `lcd"` oder `lcd.` benutzt. Natürlich ist auch die direkte Beschriftung des `TOB` möglich mittels `c!` oder `move`. Die Länge des Puffers ermöglicht es auch die Anzeige zu scrollen oder Menüs zu machen. Aber das sei der Fantasie der Leser überlassen. Viel Vergnügen beim Ausprobieren und Experimentieren.

Mein Dank gebührt Matthias Trute für das amforth und seine Hilfe bei Fragen dazu. Und Ulrich Hoffmann für die anregende Disskussion und gute Hinweise wenn ich nicht weiter wusste.

Links

Butterfly im Forth–Wiki	`http://www.forth-ev.de/wiki/doku.php/projects:avr:atmega169`
Atmel–Homepage	`http://www.atmel.com/`
Atmel–LCD–Treiber	AVR065: LCD Driver for the STK502 and AVR Butterfly (doc2530.pdf)

Adresse	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Page
(0xFF)	reserved	-	-	-	-	-	-	-	-	
(0xFE)	LCDDR18	-	-	-	-	-	-	-	324	225
(0xFD)	LCDDR17	323	322	321	320	319	318	317	316	225
(0xFC)	LCDDR16	315	314	313	312	311	310	309	308	225
(0xFB)	LCDDR15	307	306	305	304	303	302	301	300	225
(0xFA)	reserved	-	-	-	-	-	-	-	-	
(0xF9)	LCDDR13	-	-	-	-	-	-	-	224	225
(0xF8)	LCDDR12	223	222	221	220	219	218	217	216	225
(0xF7)	LCDDR11	215	214	213	212	211	210	209	208	225
(0xF6)	LCDDR10	207	206	205	204	203	202	201	200	225
(0xF5)	reserved	-	-	-	-	-	-	-	-	
(0xF4)	LCDDR8	-	-	-	-	-	-	-	124	225
(0xF3)	LCDDR7	123	122	121	120	119	118	117	116	225
(0xF2)	LCDDR6	115	114	113	112	111	110	109	108	225
(0xF1)	LCDDR5	107	106	105	104	103	102	101	100	225
(0xF0)	reserved	-	-	-	-	-	-	-	-	
(0xEF)	LCDDR3	-	-	-	-	-	-	-	24	225
(0xEE)	LCDDR2	23	22	21	20	19	18	17	16	225
(0xED)	LCDDR1	15	14	13	12	11	10	9	8	225
(0xEC)	LCDDR0	7	6	5	4	3	2	1	0	225
(0xEB)	reserved	-	-	-	-	-	-	-	-	
(0xEA)	reserved	-	-	-	-	-	-	-	-	
(0xE9)	reserved	-	-	-	-	-	-	-	-	
(0xE8)	reserved	-	-	-	-	-	-	-	-	
(0xE7)	LCDCCR	LCDD2	LCDCD1	LCDCC0	–	LCDCC3	LCDCC2	LCDCC1	LCDCC0	223
(0xE6)	LCDFFR	–	LCDPS2	LCDPS1	LCDPS0	–	LCDD2	LCDD1	LCDD0	221
(0xE5)	LCDCRB	LCDCS	LCD2B	LCDMUX1	LCDMUX0	–	LCDPM2	LCDPM1	LCDPM0	220
(0xE4)	LCDCRA	LCDEN	LCDAB	–	LCDIF	LCDIE	–	–	LCDBL	219
(0xE3)	reserved	-	-	-	-	-	-	-	-	

Tabelle 1: Die LCD–Register im ATMega169

```
$E7 constant LCDCCR \ contrast control register
   \ LCDCCR7..5 = LCDCD CLK driver time disply is on; %111 = 50%.
   \ LCDCCR4 ist nc
   \ LCDCCR3..0 = LCDCC Contrast Control; %1111 = 3,35V; %0000 = 2,60V.

 $E6 constant LCDFRR \ frame rate register
   \ LCDFRR7 is nc
   \ LCDFRR6..4 = LCDPS prescaler for clock divide; %0000 = CLK/16.
   \ LCDFRR3 ist nc
   \ LCDFRR2..0 = LCDCD clock divide; %110 = 7 is common framerate.

 $E5 constant LCDCRB \ control register B
   \ LCDCRB7 = LCDCS clock source; %0 = internal clock.
   \ LCDCRB6 = LCD2B bias; %1 = 1/3 bias für alle 4 backplanes.
   \ LCDCRB5..4 = LCDMUX duty cyles; %11 = 1/4 weil es 4 backplanes sind.
   \ LCDCRB3 ist nc
   \ LCDCRB2..0 = LCDPM port mask; %111 = alle Segmentleitungen werden benutzt.

 $E4 constant LCDCRA \ control register A
   \ LCDCRA7 = LCDEN enable display driver; %1 = on, %0 = off.
   \ LCDCRA6 = LCDAB low power wave form; %1 = use low power.
   \ LCDCRA5 is nc
   \ LCDCRA4 = LCDIF interrupt flag; gesetzt wenn ein Frame fertig ist.
   \ LCDCRA3 = LCDIE interrupt enable; %1 = interrupt enabled.
   \ LCDCRA2..1 sind nc
   \ LCDCRA0 = LCDBL blank display; entlädt das Display über alle Segmenttreiber; data register bleiben erhalten.
```

Tabelle 2: Der Display–Treiber verwendet 4 ATmega169–Control–Register

```
1   \ ***** lcd look-up table for AVR Butterfly *****
2
3   create sgmt \ segment lookup table
4   0A51 , 2A80 , 0000 , 0A00 , 0A51 , 0000 ,
5   5559 , 0118 , 1e11 , 1b11 , 0b50 , 1b41 ,
6   1f41 , 0111 , 1f51 , 1b51 , 0000 , 0000 ,
7   0000 , 0000 , 0000 , 0000 , 0000 , 0f51 ,
8   3991 , 1441 , 3191 , 1e41 , 0e41 , 1d41 ,
9   0f50 , 2080 , 1510 , 8648 , 1440 , 0578 ,
10  8570 , 1551 , 0e51 , 9551 , 8e51 , 9021 ,
```

```
11  2081 , 1550 , 4448 , c550 , c028 , 2028 ,
12  5009 , 0000 , 0000 , 0000 , 0000 , 0000 ,
13
14  \ ***** control display *****
15
16  : lcdon ( -- ) \ init lcd control registers
17    \ EF E7 c!   \ lcdccr: driver time 50%; contrast 3,35V
18    E0 E7 c!   \ ... 2,60V looks better.
19    07 E6 c!   \ lcdfrr: (ps=0 == 1/16, cd=7) --> 36,6 Hz
20    77 E5 c!   \ lcdcrb: cs=0, 2b=1, mux=11==1/4 duty, pm=111==all
21    C0 E4 c! ; \ lcdcra: en=1, ab=1, if=0, ie=0, bl=0=on
22
23  : lcdoff ( -- ) 0 E4 c! ; \ disable LCD display.
24
25  \ turn segment blank bit on&off
26  : segon  ( -- ) E4 c@ 0FE and E4 c! ;
27  : segoff ( -- ) E4 c@   1  or E4 c! ;
28
29  \ ***** move characters into lcd display *****
30
31  : cseg ( char -- seg )     \ get segment code.
32    dup 02a <
33      if drop 0 exit then  \ we start with: $2A='*'
34    dup 40 > if 5F and then  \ upper characters only
35    02a - sgmt + i@ ;
36
37  : cnib ( seg -- n0 n1 n2 n3 ) \ convert to nibbles.
38    >r
39    r@            0f and
40    r@  04 rshift 0f and
41    r@  08 rshift 0f and
42    r>  0C rshift 0f and ;
43
44  : lo! ( n0 n1 n2 n3 lcddr0 -- ) \ fit in even digits
45    >r
46    r@ 0F + c!
47    r@ 0A + c!
48    r@ 05 + c!
49    r>      c! ;
50
51  : hi! ( n0 n1 n2 n3 lcddr0 -- ) \ fit in odd digits
52    >r
53    4 lshift  r@ 0f + c@    or r@ 0f + c!
54    4 lshift  r@ 0a + c@    or r@ 0a + c!
55    4 lshift  r@ 05 + c@    or r@ 05 + c!
56    4 lshift  r@      c@ or r>         c! ;
57
58  : (wlcd) ( tob -- tob n0 n1 n2 n3 lcddr0 )
59    dup c@ cseg cnib 0ec ;
60
61  : wlcd ( adr -- ) \ move 6 chars from adr to lcd data registers.
62    (wlcd)     lo! 1+
63    (wlcd)     hi! 1+
64    (wlcd) 1+  lo! 1+
65    (wlcd) 1+  hi! 1+
66    (wlcd) 2 + lo! 1+
67    (wlcd) 2 + hi! drop ;
68
69  \ ***** create a character buffer *****
70
71  variable >out
72  heap e@ constant tob  0f allot \ make terminal output buffer
73
74  : cnul ( -- ) \ clear tob.
75    0f 0 do bl tob i + c! loop ;
76
```

```
77  : lcdcr ( -- ) 0 >out ! cnul ;  lcdcr \ reset cursor.
78
79  : lcdemit ( c -- ) \ emits character into character buffer.
80    >out @ 0f > if drop  \ 16 characters, cut rest
81    else tob >out @ + c!   1 >out +! then ;
82
83  \ ***** redirect regular output to lcd *****
84
85  : <lcd>   ( -- )   lcdcr ['] lcdemit 'emit ! ;
86  : </lcd>  ( -- )   tob wlcd ['] tx0     'emit ! ;
87
88  \ Now we can do a phrase like: ... <lcd> ." hallo" </lcd> ...
89  \ Or have something like: ... ( n -- ) <lcd> . </lcd> ...
90
91  \ ***** special lcd output *****
92
93  : lcd. ( n -- )  <lcd> . </lcd> ;
94
95  \ You can have something like: ... 1122 lcd. ...
96
97
98  : postpone ( -- ) \ 'ccc'
99    bl word find dup 0<
100   if drop compile compile , exit then
101   if , exit then
102   drop [ base @ decimal ] -13 throw [ base ! ] ; immediate
103
104 : lcd" ( 'ccc' -- )
105     postpone <lcd> postpone ." postpone </lcd> ; immediate
106
107 \ Now we can compile a phrase like: ... lcd" hallo" ...
108
109
110 \ finis 25.03.2007 mka
```

Grafische Forth–Dokumentation

André Elgeti

Dieser Artikel beschäftigt sich mit dem *Urschleim*. Wer im technischen Bereich tätig ist/war, weiß, dass zu einer Maschine oder Anlage ein ganzer Schwung an Dokumenten gehört. Dazu gehören z. B. Ansichten, Schaltpläne, Funktionsbeschreibungen und Wartungsvorschriften.

Ich würde mir wünschen, dass Autoren in der *Vierten Dimension* die Funktion größerer Wörter oder Vokabularien mit Hilfe grafischer Darstellungen veranschaulichen könnten, um sie so auch den *Nicht-Ingenieuren* erschließbar zu machen.

Als ich in den 80–ern mit dem Programmieren angefangen habe, waren noch Ablaufpläne und Struktogramme in Mode. Sie ersetzten grafisch den Pseudeocode, wurden aber durch die OOP weitgehend abgelöst, und sind heute oft unbekannt. Wobei ich aber auch eingestehen muss, dass sie schlecht handhabbar sind.

Durch Zufall fiel mir vor einiger Zeit das Buch *Software–Engineering* in die Hände, wo genau diese Dinge beschrieben wurden, wobei es dafür sogar DIN– Vorschriften gibt. Dabei ging es u. a. um:

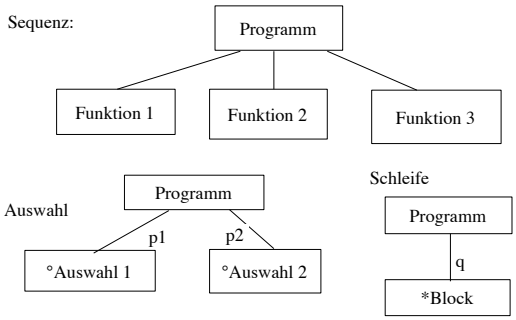
- Ablaufpläne
- Struktogramme
- Entscheidungstabellen
- Jackson–Pläne

Ich möchte auf Letzteres näher eingehen. In der Programmierung gibt es diese Möglichkeiten:

- Befehlssequenz
- Schleife
- Entscheidung

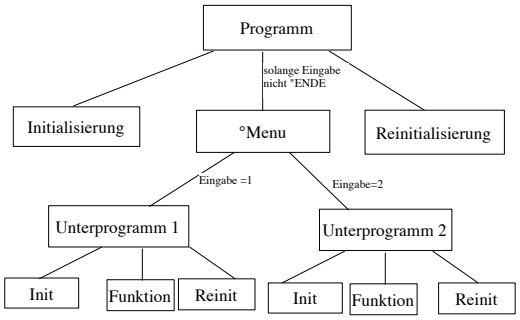
Die Programmabschnitte werden als Rechtecke dargestellt.

- Bei der Sequenz verbindet die aufrufende Routine die Rechtecke, die von links nach rechts abgearbeitet werden.
- Bei der Schleife erhält der Schleifenkörper einen Stern (*) und die Verbindung die Schleifenbedingung.
- Bei der Verzweigung enthalten die Pfeile die Entscheidungsbedingung und die aufgerufenen Blöcke einen ° (siehe auch Zeichnung 1).



Zeichnung 1: Darstellung der grundlegenden Programminhalte

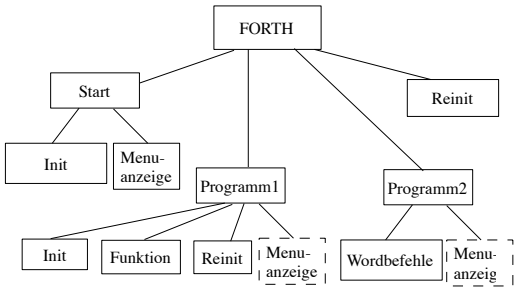
Ein typisches Programm mit Menu–Auswahl sähe dann etwa so aus (Zeichnung 2): Beim Start wir eine Initialisierungsroutine durchlaufen. Anschließend wird eine Funktionsauswahl angezeigt. Der nächse Programteil fragt die Auswahl ab und verzweigt entsprechend zu einem Unterprogramm. Nach dessen Ende wird wieder die Funtionsauswahl angezeigt. Beim Beenden wird die Reinitialisierungsroutine ausgeführt, die das System wieder in seinen Ursprungszustand zurücksetzt, und wieder in das Betriebssystem zurückführt.



Zeichnung 2: Darstellung eines Programmes mit Menu–Steuerung

In Zeichnung 3 habe ich dargestellt, wie dasselbe Verhalten unter Ausnutzung der Eigenarten von Forth ermöglicht wird. Dabei ist eben auch zu beachten, dass das Programm *FORTH* die Wörter/Unterprogramme mit ihrem Namen aufruft. Das Schöne ist dabei, dass der Aufruf eines Wortes durch ein anderes (oft) ein Unterprogrammaufruf ist.

Zum Beispiel:
Beim Aufruf des Wortes *Start* erfolgt die Initialisierung. Anschließend erscheint ein Bildschirm mit den weiteren Wörtern. Jedes Wort ruft nach seiner Abarbeitung diesen Bildschirm auf. Das Schlaue daran ist aber, dass der versierte Anwender auch andere Wörter aufrufen kann, und damit die Funktionalität des Programmes selbst erweitern, oder auch andere Aufgaben dazwischen schieben kann.



Zeichnung 3: Darstellung der gleichen Funktion unter Nutzung der Eigenheiten von Forth

Literatur

1. Ekbert Hering, *Software Engineering*, Friedr. Vieweg & Sohn Brauschweig/Wiesbaden, 3. durchgesehene Auflage 1993

2. Dr.-Ing. Gert-Ulrich Vack, *Programmieren in FORTH*, VEB Verlag Technik Berlin, 1. Auflage 1990

Forth 200X–Treffen auf der EuroForth 2006

Anton Ertl

Am Tag vor der EuroForth 2006 fand wieder einmal das Treffen des Forth 200X–Komittees statt, das am nächsten Forth–Standard arbeitet. Die Teilnehmer waren Willem Botha, Federico de Ceballos, Anton Ertl, Peter Knaggs, Nick Nelson (Beobachter), Stephen Pelc, Jannus Poial (Beobachter), und Bill Stoddart.

Der RfD/CfV–Prozess und seine Beziehung zum Treffen wurde diskutiert, und folgender Zeitplan festgelegt:

- Ein RfD sollte mindestens 12 Wochen vor dem Treffen erstmals veröffentlicht werden, um die anderen Termine wahrscheinlich einhalten zu können.
- Der CfV muss spätestens 6 Wochen vor dem Treffen eingereicht werden, bei dem er besprochen werden soll.
- Der Wahlleiter muss den Zwischenstand spätestens 4 Wochen vor dem Treffen veröffentlichen.
- Beim Treffen wird dann über den Vorschlag abgestimmt. Die Mehrheit entscheidet.

Die Kontakt–Daten der Kommittee–Mitglieder werden veröffentlicht, damit sich die Forth–Gemeinde an sie wenden kann und nicht selbst zum Treffen kommen muss.

Peter Knaggs, der Editor des Standard–Dokuments, stellte das sehr gelungene Dokument vor, das als LaTeX–Quellcode und in PDF–Form vorlag (und in beiden Formen veröffentlicht werden wird). Es kann u. a. in einer Inline–Version erzeugt werden, wo bei jedem Wort die Begründung direkt dabei steht statt in einem Anhang. Natürlich gab es immer noch Verbesserungsvorschläge.

Ein zentraler Punkt war natürlich die Diskussion der Vorschläge, die den RfD/CfV durchlaufen hatten (siehe `http://www.forth200x.org/rfds.html`):

EKEY return values Hierbei geht es darum, dass man mit den von EKEY gelieferten Werten auch etwas anfangen kann, z.B. erkennen, ob eine Cursor–Taste gedrückt wurde und welche. Bei diesem Vorschlag verlangte das Kommittee noch eine Klarstellung, welche Beziehung zwischen EKEY und KEY herrscht, ansonsten wurde der Vorschlag akzeptiert.

FP–stack Forth–Systeme sollen garantieren, dass es einen separaten Gleitkomma–Stack gibt. Bei diesem Vorschlag verlangte das Kommittee, dass er so umformuliert werden soll, sodass die Auswirkungen auf existierende Programme und Systeme deutlicher werden.

One–time file loading Dieser Vorschlag standardisiert das Wort `required`, das so funktioniert wie `included`, wenn die Datei noch nicht geladen wurde, aber nichts tut, wenn sie schon geladen wurde; zusätzlich werden noch `include` und `require` standardisiert. Dieser Vorschlag wurde angenommen.

Ausserdem wurden noch Vorschläge diskutiert, die noch nicht die CfV–Stufe erreicht hatten. U. a. resultierte daraus eine Umformulierung von `TO`, sodass künftige Änderungen (`2VALUE`, `FVALUE`) leichter gemacht werden können.

Weiters wurde beschlossen, dass der Sitzungsbericht in diversen Papiermedien veröffentlicht werden sollte (wichtig für eine offizielle Standardisierungsorganisation), was u. a. hiermit geschieht.

Das nächste Treffen findet wieder am Tag vor der EuroForth statt, und Ihr seid herzlich eingeladen, ihm beizuwohnen.

Links

Forth200x Homepage `http://www.forth200x.org`

CfV with results
Separate FP Stack
EKEY return values
One-time file loading
[DEFINED] and [UNDEFINED]
PARSE-NAME
Extension queries
Deferred words.

RfDs without CfVs
Directories
2VALUE
Enhanced local variable syntax
THOW codes and iors
Escaped Strings
Structures
SYNONYM
FVALUE
XCHAR wordset

Forth–Gruppen regional

Moers

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

Mannheim

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

München

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

Hamburg

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

Mainz

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

Gruppengründungen, Kontakte

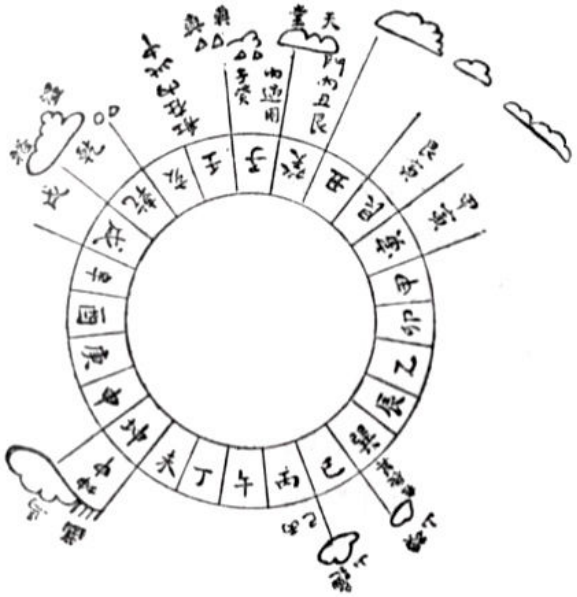
Hier könnte Ihre Adresse oder Ihre Rufnummer stehen — wenn Sie eine Forthgruppe gründen wollen.

µP–Controller Verleih

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

Spezielle Fachgebiete

FORTHchips (FRP 1600, RTX, Novix)	**Klaus Schleisiek–Kern** Tel.: **(0 40) – 37 50 08 03** (g)
KI, Object Oriented Forth, Sicherheitskritische Systeme	**Ulrich Hoffmann** Tel.: **(0 43 51) – 71 22 17** (p) Fax: **– 71 22 16**
Forth–Vertrieb volksFORTH ultraFORTH RTX / FG / Super8 KK–FORTH	Ingenieurbüro **Klaus Kohl–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.





Krainerhütte Baden Wien

Einladung zur Forth–Tagung 2007

Anton Ertl

Zeit und Ort

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

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

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

Vorläufiges Programm

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

Umgebung

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

Anreise

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

Weitere Informationen

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