Vierte Dimension

3/2019

Das Forth–Magazin

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

In dieser Ausgabe:

Ein Dateisystem für den seriellen Flash–Speicher

Rechnen mit ganzen Zahlen in Forth

Linear interpolierte Tabelle

Uhr

One–Letter–Words

Tali–Forth–2 — ein modernes Forth für einen modernen Klassiker

show_registers

Fingerübungen

Swap's Tagebuch

36C3 — 36. Chaos Communication Congress

Dienstleistungen und Produkte

Leserbriefe und Meldungen 5

Ein Dateisystem für den seriellen Flash–Speicher 8
Willem Ouwerkerk

Rechnen mit ganzen Zahlen in Forth 10
Michael Kalus

Linear interpolierte Tabelle 16
Rafael Deliano

Uhr 18
Rafael Deliano

One–Letter–Words 23
Martin Bitter

Tali–Forth–2 — ein modernes Forth für einen modernen Klassiker 25
Carsten Strotmann

show_registers 26
Martin Bitter

Fingerübungen 28
Ulrich Hoffmann

Swap's Tagebuch 29
Matthias Koch

36C3 — 36. Chaos Communication Congress 32
27. bis 30. Dezember 2019 in Leipzig

Das Titelbild ist ein Ausschnitt aus: Mondphasenuhr mit Monatskranz AUTOR Sergio Fabris
wikimedia.org

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

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

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00 € + Porto u. Verpackung

Liebe Leser,

dieses Jahr war die EuroForth bei uns in Hamburg. Ich hoffe, etliche von euch waren dort. Und der Eine oder Andere berichtet noch, wie es gewesen ist. Ihr wisst ja, Beiträge für unser Forth-Magazin sind immer willkommen.

Es hat noch ein bischen gedauert, bis dieses Heft in Schwung gekommen ist. Doch nach dem journalistischen Sommerloch liefs dann doch.

Willem Ouwerkerk macht das Micro–Launchpad weiter fit. Ein winziger externer serieller Flash–Speicher liefert dazu fast unbegrenzte Möglichkeiten.

Und ich selbst hab mal niedergeschrieben, was es mit den ganzzahligen Berechnungen in Forth so auf sich hat. Deren Grenzen und Tücken, besonders bei der Division — im durch registerbreiten begrenzten Zahlenraum der kleinen MCUs ebenso wie in größeren Systemen wie dem Gforth. Mir war so danach, da mal etwas Klarheit zu schaffen. Was hoffentlich gelungen ist.

Da passte es ganz gut, dass Rafael Deliano am praktischen Beispiel vorführt, wie man in Forth skalieren kann, integer versteht sich, dennoch um zwei Nachkommastellen.

Und seine Uhr rund um ein Rubidium–Frequenznormal kann sich sehen lassen. Er zeigt auf, wie man das nötige Tempo dafür auch mit der GP32–CPU und etwas Peripherie hinbekommen kann.

Martin Bitter ist da über eine zunächst mysteriöse Sache gestolpert bei der Daten–Interpretation in Forth — aber lest selbst, was es damit auf sich hatte.

Und er lässt uns einen Blick werfen in seinen Werkzeugkasten. Ein gutes Werkzeug, um sich eine neue MCU zu erschließen, ist sein Register–Anzeiger.

Carsten Strotmann, viel unterwegs auch in vintage computing, hat was für die 6502-CPU ausgegraben. Ja, gibt's die denn noch? In der Tat! Zumindest als 65C02. Ja, sowas! Auch ich hab damit mal angefangen, und das ist nun schon seeeehr lange her — gemessen in Elektronik-Zeitaltern.

Ulrich Hoffmann schließlich gibt schon mal einen Blick hinter die Kulissen frei. Man kann ahnen, was alte Hasen auf einer Euroforth so treiben. Die ganzen Beiträge findet ihr übrigens auf dem Server der Forth–Gesellschaft.

`https://wiki.forth-ev.de/doku.php/events:ef2019:start`

Und falls ihr euch gefragt habt, was unser SWAP eigenlich inzwischen so macht, Matthias Koch hat sein Tagebuch stibitzt ...

Euch allen eine gute Zeit, Michael

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

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

Ulrich Hoffmann
Bernd Paysan
Carsten Strotmann

Kontakt: `Direktorium@Forth-ev.de`

Nachfolger für das Forth-Büro gesucht

Nach vielen Jahren möchten Andrea und Ewald Rieger das Forth–Büro abgeben. Die Forth–Gesellschaft sucht nun nach Nachfolgern, welche die Arbeit des Forth–Büros vollständig oder teilweise weiterführen möchten. Darunter fallen u. a.

- Mitgliederverwaltung
- Kassenführung
- Druck und Versand dieser Publikation „Vierte Dimension“

Wer sich berufen fühlt, diese Aufgaben für die Forth–Gesellschaft zu übernehmen, melde sich bitte bei den *Direktoren* oder beim derzeitigen Forth–Büro.

Vor dem großen Teich

From the other side of The Big Teich, unter diesem Titel hatte Henry Vinerts lange Jahre direkt von den Treffen der Silicon Valley FIG berichtet. Fred Behringer fasste dessen Berichte und die Inhalte des niederländischen „Feigenblattes“ in seiner Rubrik *Gehaltvolles* zusammen. Beides ist Vergangenheit.

Das „Feigenblatt“ wird nicht mehr verlegt. Aber der niederländische Forthverein besteht weiterhin: Die *HCC forth gg*[1] trifft sich regelmäßig jeden zweiten Samstag in jedem zweiten Monat. Recht kurz vor dem jeweiligen Termin erfolgt *online* eine gesonderte Einladung unter dem Titel `/ForthWords-<Jahr>-<Nr>/`, die unter anderem einen Überblick über aktuelle Themen und geplante Aktivitäten enthält. Wie so vieles hängt dies von dem Einsatz Einzelner ab.

> „Holländisch ist gar nicht so schwer. Es ähnelt sehr den norddeutschen Sprachgepflogenheiten. Und außerdem ist Forth sowieso international. Neugierig? Werden Sie Förderer der HCC–Forth–gebruikersgroep.“

Dies schrieb Willem Ouwerkerk in einer Anzeige in der *Vierten Dimension*. Und es stimmt! Tipp: Manchmal hilft es, laut zu lesen! Ein Verein lebt unter anderem von der Anzahl seiner Mitglieder ...

Juni

Drei Arbeitsgruppen geben Berichte ihrer Aktivitäten. Es wurden zwei Vorträge gehalten und Aufgaben gestellt.

Willem Ouwerkerk und Leon Konings stellen den *noForth–Assembler* vor. Normalerweise sind Forthworte schnell genug. Aber es kann sich vor allem in Interrupts lohnen, direkt in Maschinencode zu programmieren. Sie zeigen am Beispiel des MSP430G2553 und noForth, wie das mit den Assemblerroutinen des noForth gelingt. Eine Übersicht über den noForth–Assembler liegt als PDF bei oder kann Online erhalten werden: `http://home.hccnet.nl/anij/nof/noforth%20asm.pdf`

[1] HCC forth gebruikersgroep

Manchmal wird bei datenintensiven Anwendungen der Platz auf der MCU knapp. Dann kann man auf einen externen Speicher ausweichen. Willem geht dann noch weiter. Er zeigt, wie er für die MCU ein komplettes Filesystem in einem externen Speicher anlegt. Die konkrete Anwendung ist ein Programmiergerät für verschiedene MSP430Gxx, MSP430Fxx und MSP430FRxx, das die in einem FRAM gespeicherte Dateien verwendet.

Diesmal wurden drei Aufgaben gestellt: Mit dem noForth–Assembler sollen drei Worte geschrieben werden. Je eines, das aus einem 16–Bit–Wert das Low– bzw. das High–Byte isoliert und ein drittes, das dann beide Bytes zurückgibt.

Wer mitmachen will, kann seine Lösungen an Willem schicken: `w.ouwerkerk@kader.hcc.nl`

August

Hoffentlich nur! (MB) wegen der Ferienzeit sind keine Workshops zum Treffen angemeldet worden. Daher wurde es ein recht zwangloses Treffen mit Ad–hoc–Entscheidungen über Themen. Frans van der Mart berichtete in der Forthwrite über seine Arbeit mit noForth an einem Sensor von Bosch (BME280) und wie er ihn in das den HCC–Freunden bekannte Open–Source–Projekt *DOMOTICACZ* einbindet. Er wird das auch auf dem Treffen demonstrieren. Frans hatte sich für diese Arbeit Hilfe von Albert Nijhof zum Umgang mit doppeltgenauen Zahlen unter noForth geholt.

Leon Konings wird aus diesem Anlass einen Vortrag über doppeltgenaue Zahlen und Routinen in noForth halten.

Anmerkung

Ich selbst habe den BME280 in einem Messemodell, einem Stirlingmotor, im Einsatz. Es ist sehr interessant, die Codeschnipsel von Frans mit meinem Code zu vergleichen.

Martin Bitter

Das TI Launchpad wurde modernisiert

Texas Instruments hat sein legendäres Launchpad modernisiert. Für die MSP430 ultra–low–power MCUs gibt es nun das *MSP–EXP430G2ET Value Line MSP430 LaunchPad™ Development Kit*. Es hat dieselben Abmessungen wie seine Vorgängerverson V1.5, hat auch nach wie vor die Pfostenleisten für die booster packs und kommt nach wie vor mit der gesockelten DIP–Version der MCU-Serien MSP430G2xx1, MSP430G2xx2, MSP430G2xx3 und MSP430F20xx. Neu ausgelegt auf dem Board wurde der EZ–FET–Emulator, der nun eine „EnergyTrace“ genannte Komponente hat, die es erlaubt, den Stromverbrauch in Betrieb zu verfolgen. Benutzerseitig wurde eine 3–Farben–LED spendiert, zusätzlich zu den beiden bekannten LEDs, der roten und der grünen.

> Free software development tools are also available, such as TI's Eclipse–based Code Composer Studio™IDE (CCS) and IAR Embedded Workbench® for MSP430IDE (IAREW430). Both of these IDEs support EnergyTrace™ technology for real–time power profiling and debugging when paired with the MSP430G2553 LaunchPad development kit.

Klingt gut. Also schaun wir mal wie es sich bewährt.

`http://www.ti.com/lit/ug/slau772/slau772.pdf`

mk

Auf der Suche

#1 — ForthTree–Mailbox

Quellen der ForthTree–Mailbox gesucht: Ich beschäftige mich derzeit mit einem Seriell–zu–WLAN–Modem auf Basis des ESP8266. Mit einem solchem Modem kann ein alter Rechner (z. B. unter MS–DOS) per WLAN und Telnet auf Mailbox–Systeme (BBS) im Internet zugreifen. Davon gibt es weltweit noch mehr als 500 Stück. Ich würde gerne mit der alten Forth–Mailbox *ForthTree* ein paar Experimente machen. Gibt es die Software oder ein Backup des alten Mailbox–Systems noch irgendwo? Bitte bei Carsten Strotmann melden!

`https://subethasoftware.com/2018/02/28/wire-up-your-own-rs-232-wifi-modem-for-under-10-using-esp8266-and-zimodem-firmware/`

`https://www.telnetbbsguide.com/`

#2 — ZF–Forth

Die Forth–Gruppe *Moers* hat in den 1990er Jahren eine Zusammenstellung des ZF–Forth von Tom Zimmer auf Disketten herausgegeben. Ich würde diese Zusammenstellung gerne im Internet dokumentieren und archivieren. Die Gruppe Moers, speziell Friedrich Prinz, haben für das ZF–Forth viele interessente Programme und Artikel geschrieben. Es wäre schade, wenn diese Arbeit verloren ginge. Wer noch die Originaldisketten oder ein Backup der Zusammenstellung hat, melde sich bitte bei Carsten Strotmann.

cas

Forth spielend lernen — Ein Sommerprojekt

In seinem Phlog[2] mit dem Titel: „Writing 2048 in Forth, or How I Spent My Summer Vacation" beschreibt Dave Bucklin, wie er über die Sommerferien das Spiel 2048 in Gforth implementiert hat. Dave benutzt das Spiel als ein überschaubares Software–Projekt, um Forth zu lernen und ein wenig tiefer in die Forth–Programmierung einzusteigen. Der Text findet sich auf dem SDF.ORG–Gopher–Server.

`https://en.wikipedia.org/wiki/Phlog`

`https://en.wikipedia.org/wiki/2048_(video_game)`

`gopher://sdf.org/0/users/dbucklin/posts/2048.txt`

Benutzer allzu moderner Betriebssysteme, welche das Gopher–Protokoll nicht mehr unterstützen, können den Floodgap Gopher–zu–Web–Proxy benutzen, um den Text zu lesen.

`https://gopher.floodgap.com/gopher/`

Der Quellcode des Spiels 2048 für Gforth befindet sich auf GitLab unter:

`https://gitlab.com/davebucklin/2048.`

cas

Gedruckte Version des Forth–200x–Draft–Standards

Die Forth–Gruppe *Rhein–Ruhr* hat für Messen und Veranstaltungen einige Exemplare des aktuellen Forth–200x–Standards 18.1 von August 2018 drucken lassen. Das Ringbuch enthält auf 358 Seiten den aktuellen Arbeitsstand der Forth–Standard–Gruppe. Einige wenige Exemplare sind noch übrig und können zum Selbstkostenpreis von 20 Euro (zzgl. Versandkosten) bestellt werden.

Kontakt Carsten Strotmann: `cstrotm@forth-ev.de`

cas

Kreativ- und Technikfestival Herford

MAKEOWL
MACH DEIN DING!

Liebe Maker!

Es ist soweit: Die Maker Faire OWL geht in eine zweite Runde — allerdings mit neuem Namen und leicht verändertem Erscheinungsbild. Eins können wir Euch allerdings jetzt schon versprechen: die einmalige Atmosphäre des Events im Güterbahnhof Herford bleibt!

Am *20. und 21. Juni 2020* öffnet der Güterbahnhof Herford wieder seine Tore für Erfinder, Tüftlerinnen und Querdenker. Das *Kreativ- und Technikfestival MAKE OWL — Mach Dein Ding!* bietet Euch eine spannende Plattform, Eure Ideen und Projekte zu präsentieren und Euch mit Gleichgesinnten auszutauschen! Markiert den Termin bereits jetzt fett in Euren Terminkalendern – wir freuen uns auf Euch!

Weitere Informationen findet Ihr auf der Website. Der Call for Makers öffnet im Herbst 2019!

Bis bald

Euer Make OWL-Team[3]

[2] Phlog — ein Blog im Gopher–Space

[3] Quelle: E–Mail von Esther Rüssler, 04.09.2019, city2science GmbH, Herford

https://www.makeowl.de/

Gforth 0.7.9 — HELP

Der *stable release*, der mit der aktuellen Linux–Mint–Auslieferung daher kommt, ist die Version 0.7.3. Aktuell sind die Gforth–Entwickler bei 0.7.9. Wer da mitspielen und die wirklich sehr feine Hilfe HELP kennenlernen möchte, die Gforth nun bietet, muss sich an die Quellen machen. Oder der Synaptic–Paketverwaltung des Linux–Mint beibringen, was es zu tun hat.

Zunächst musste ich bei den .list Dateien, die Synaptic verwendet, die Datei net2O.list hinzufügen. In dieser Datei steht nur eine Zeile:

```
deb https://net2o.de/debian testing main
```

Danach war im Terminal folgendes zu tun:

```
sudo apt install apt-transport-https
sudo apt update
sudo apt upgrade
sudo
wget -O -
https://net2o.de/bernd@net2o.de-yubikey.pgp.asc
| sudo apt-key add - [4]
apt update
apt upgrade
sudo apt install gforth
gforth
```

Heureka! Das jüngste Gforth ist da! Und auch Synaptic zeigte nun an, dass ein modernes Gforth installiert worden war. Und die Updates des Gforth waren nun auch mit der GUI der Paketverwaltung zu bewerkstelligen.

Das neue HELP des Gforth kann dann sowas:

```
michael@michael-ThinkPad-L412:~$ gforth
Gforth 0.7.9_20190912, Copyright (C) 1995-2018
Free Software Foundation, Inc.
Gforth comes with ABSOLUTELY NO WARRANTY; for
details type ‘license’
Type ‘help’ for basic help
help dup
/usr/share/gforth/0.7.9_20190912/doc/gforth.txt:3906
’nip’ w1 w2 - w2 core-ext "nip"
’dup’ w - w w core "dupe"
’over’ w1 w2 - w1 w2 w1 core "over"
’tuck’ w1 w2 - w2 w1 w2 core-ext "tuck"
’swap’ w1 w2 - w2 w1 core "swap"
’pick’ S:... u - S:... w core-ext "pick"
Actually the stack effect is ’ x0 ... xu u --
x0 ... xu x0 ’.
’rot’ w1 w2 w3 - w2 w3 w1 core "rote" ok
```

Wunderbar.

Besten Dank an Martin Bitter für die Aufklärung darüber, wie das in Linux–Mint gehen muss und Bernd Paysan und Anton Ertel für die flotte Fertigstellung der Dokumentation. mk

https://www.gnu.org/software/gforth/

ForthE[5]

Ein Forth in Emacs Lisp, in 11 Zeilen Code. :)

> Yes, it is a very simple FORTH macro. I don’t think elispers are about to drop their parentheses just yet because of this eleven line macro.

```
(defun eforth--inner (lang list)
  (let ((s))
    (dolist (i list)
      (if (assoc i lang)
          (let ((fn (cadr (assoc i lang))))
            (if s
                (progn (apply fn s) (setf s nil))
              (funcall fn)))
        (push i s)))))
(defmacro eforth (lang list)
  ‘(eforth--inner ,lang ’,list))
```

Aus dem Netz gefischt von Erich Wälde

https://www.reddit.com/r/emacs/comments/c40pmh/eforth_a_forth_in_emacs_lisp14_lines/

[4] wget ... add - ist eine Zeile!

[5] Der Autor hatte es zunächst eForth nennen wollen, aber das ist ja schon vergeben an Ting’s „Educational Forth“ Systeme.

Ein Dateisystem für den seriellen Flash–Speicher

Willem Ouwerkerk

Für den BSL–Programmer ist ein Massenspeicher für die Speicherung mehrerer (noForth) Binärdateien erforderlich. Die erste Wahl war ein I²C–FRAM, aber das ist relativ langsam und teuer. Aber da gab es noch einen W25Q16 in der „Noch–zu–tun–Box". Das ist ein 16–MBit–Flash–Speicher mit SPI–Schnittstelle. Mehr als genug (ungefähr 200) noForth–Binaries passen dort hinein. Es ist schnell — verwendet einen 16–MHz–SPI–Takt — und billig (0,40 €) und befindet sich in einem kleinen 8–poligen SO–Gehäuse. Also genau das Richtige für die Weiterentwicklung des BSL–Programmers.

Erster Eindruck vom W25Q16

- 2 MegaBytes Flash
- Kann Bytes, Wörter usw. lesen
- Schreiben sektorweise in 256–Byte–Sektoren
- Löschen erfolgt in 4–KByte–Sektoren oder mehr
- Schnell (max. 130 MHz Takt) und geringer Stromverbrauch

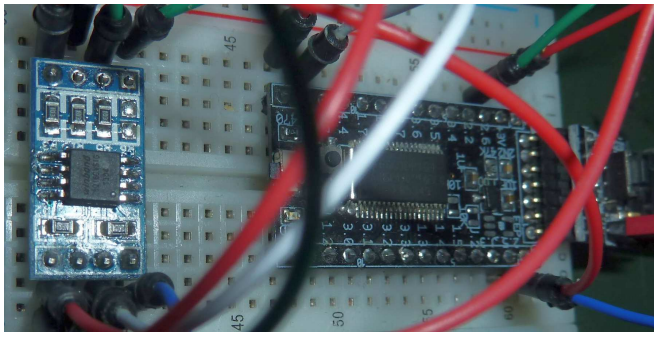


Abbildung 1: Micro–Launchpad FR5 mit W25Q16

Der SPI– und Flash–Befehlssatz

Die SPI–Routinen wurden aus dem *Egel Project* übernommen; nur die Adressen mussten angepasst werden (Tab. 1). Beachte, dass es viel mehr Befehle im Chip gibt, aber mit diesem Subset kommen wir schon prima aus.

Liste der Flash–Befehle	
Kommando	Funktion
02	Schreib einen 256–Byte–Sektor
02	Lies Data aus dem Flash (ein oder mehr Bytes)
05	Lies Statusregister 1
06	Erlaube Schreiben und Löschen
20	Lösche 4–kByte–Byte–Sektor
60	Lösche den gesamten Flash–Chip
90	Lies Chip–ID und Herstellercode

Tabelle 1: Befehle für die Verwendung des seriellen Flash-Speichers

Einige Codebeispiele

```
: WRITE-ON ( -- )
  6 {spi spi} ;
: STATUS1 ( -- b )
  5 {spi spi-in spi} ;
: BUSY ( -- )
  begin status1 1 and 0= until ;
: CHIP-ERASE) ( -- )
  write-on 60 {spi spi} busy ;
: CHIP-ERASE ( -- )
  0 #sector x! chip-erase) ;
: ADDR-FLASH ( da -- )
  spi-out split spi-out spi-out ;
: FC@ ( da -- b )
  3 {spi addr-flash spi-in spi} ;
\ Fill buffer +b from sa,
\  the address of a 256 byte sector
: READ-SECTOR ( sa +b -- )
  >r 03 {spi addr-sector r> buffer
  100 0 ?do spi-in over i + c! loop spi} drop ;
\ Write buffer +b to sa
\  address of a 256 byte sector
: WRITE-SECTOR ( sa +b -- )
  >r write-on 02 {spi addr-sector r> buffer
100 0 ?do count spi-out loop drop spi} busy ;
```

Das Dateisystem

Die Dateien werden im Flash gespeichert. Die Speicherverwaltung liegt im FRAM des FR5 ab Adresse 10000. Der MLP FR5 hat 16 KByte davon. Tab. 2 zeigt die Struktur der Dateiverwaltung. Der Dateiname selbst hat noch eine besondere Struktur (Tab. 3). Im High–Nibble des Längenbytes wird der Datei–Typ codiert, somit sind 16 Typen möglich.

Directory				
File Name	Startsektor	Länge	Token	ID–String
RAMBSL	0	10	DROP	First ...
V2x55	10	20	PROGRAM-FR	NoForth ...
Readme	30	10	SHOW	FAT Readme ...
etc.	...	...	...	...

Tabelle 2: Struktur der Dateiverwaltung

File Name												
Count		String										
0	6	R	A	M	B	S	L	-	-	-	-	-
1	5	V	2	x	5	5	-	-	-	-	-	-
2	6	R	e	a	d	m	e	-	-	-	-	-

Tabelle 3: Struktur des Dateinamens

```
\ Search directory for "name"
: FIND> ( "name" -- entry-addr true | false )
  bl-word
  xdir x@ 0 ?do
      dup 0 'entries i #entry * m+ s=
      if drop i 'entries true unloop exit then
  loop drop ." File not found " false ;
```

Dateien schreiben und lesen

Das Schreiben des Flash–Speichers ist nur in 256–Byte–Sektoren möglich. Ein RAM–Puffer wird als temporärer Speicher verwendet. Der MLP FR5 verfügt über 2 KByte RAM und damit über ausreichend Speicherplatz für einige solcher Puffer.

ASCII–Dateien werden so gespeichert, wie sie sind, mit `CR` und `LF` am Ende jeder Zeile. Das Ende der Datei ist durch eine `00` gekennzeichnet.

Intel–HEX–Dateien oder TI–TXT–Dateien werden als binäre Zeichenfolgen gespeichert. In einem zusammenhängenden Stück Speicher wird die Binärdatei abgelegt und diese Zeichenfolge dann durchlaufen, um sie zu speichern. Wenn eine HEX–Datei einen Adressensprung macht, wird eine neue Zeichenfolge gestartet. Auch hier wird das Ende der Datei mit einer Null markiert.

```
@)dir
 0 T: RAMBSL        0 10  First working F149 BSL
 1 H: V2x55        10 20  noForth V2x55 100519
 2 A: Readme       30 10  FRAM FAT system with Flash disk
Free 2032 kBytes  OK.0
@)  OK.0
@)  OK.0
@)use> V2x55
 1000:  0 18 28  0 C0 9D 1A 9E 2E 9E 44 9A 24 9E B6 9D |  (     . D $   |
 1010: 5C 9E 54 9B B2 94 28 84 6E 84 48 84 3E 80 6C 9E |\ T   ( n H > l |
 1020: 3E 20 E0 2E 30 2F 80 2F  7  0 10  0 32 18  E  0 |>  .0/ /    2   |  OK.0
@)
```

Abbildung 2: Screenshot der Bedienung

```
: B, ( b -- )
  #n 100 = if
    buf 1 !data
     -1 0 !data
    buf 1 xor to buf 0 to #n
  then
  buf buffer #n + c! incr #n ; \ Store byte
: <PATCH> ( -- ) \ Patch length
                 \ in saved sector 2
  @data if    \ Buffer used?
    ch * emit
    \ Get sector in spare buffer
    3 @data 2 read-sector
    \ Patch length
    2 @data 4 @data 2 buffer + !
    \ Save sector again
    3 @data 2 write-sector 0 2 !data
  then ;
```

Wie sieht eine Datei aus?

Weil wir im noForth nur ein einfaches Zeichen–Interface als Oberfläche haben und per Terminalprogramm damit arbeiten, müssen Dateien modifiziert werden. Abb. 3 zeigt eine (verkürzte) HEX–Datei und Abb. 4 zeigt eine modifizierte ASCII– oder Forth–Quellcode–Datei.

```
c" noForth C2x55 190513"   ' program-fr   INTELHEX: C2X55

:20180000C09D1A9E2E9E449A249EB69D5C9E549BB29428846E8448843E806C9E3E20E02E27
:08182000302F802F0700100098
:0E183200CF070006F400060801020400515510
:20FF7E0000137EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFFFD
:20FF9E007EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF73
:20FFBE007EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF53
:20FFDE007EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF7EFF33
:02FFFE00529916
:00000001FF
```

Abbildung 3: Muster HEX–Datei

```
c" Sector buffer pattern demonstration"   ' show   FORTH: DEMO1

( Fill sector buffer with a binary pattern )

inside

\ Set patterns in sector buffer
: >BUF      ( buf -- )     to buf  0 to #n ;   \ Set dest. buffer
: PATTERN1  ( buf -- )     >buf  0 FF do  i b,  -1 +loop ;
: PATTERN2  ( buf -- )     >buf  100 0 do  i b,  loop ;

: DEMO      ( u -- )       0 ?do  i . loop ;

forth

\ End

;;;
```

Abbildung 4: Quellcode Demo

Links

`http://home.hccnet.nl/anij/nof/noforth.html`

`http://home.hccnet.nl/willem.ouwerkerk/egel-for-msp430/egel%20for%20launchpad.html#e111`
Kapitel 111 — Cloning noForth

Rechnen mit ganzen Zahlen in Forth

Michael Kalus

Warum es überhaupt so viele Zahlen gibt, hat Jens Storjohann *hier kürzlich beleuchtet (Heft 2019-01). Da mich bei den digitalen Maschinen ja immer schon die Frage umtreibt, warum so ein Haufen Draht überhaupt mit mir spricht, und Forth dabei immer hilfreich war, hinter diese Geheimnisse zu kommen, nehme ich nun auch Forth als Werkzeug, um die Grundrechenarten zu enträtseln, jedenfalls die der ganzen Zahlen (integer). Und dabei besonders das, was man in der Schule teilen nennt. Aber fangen wir mal vorne an.*[1]

Beim Addieren werden zwei Zahlen zusammengezogen zu einer Zahl. „Eins und eins ist zwei“ sagen wir, „… und eins ist drei“ fahren wir fort, um dann schließlich auch sowas wie
234 + 12 = 246
rechnen zu können. Wir sind an diese arabische Rechenweise gewöhnt und machen den Übertrag zur höheren Stelle, wenn der Ziffernvorrat 0 bis 9 erschöpft ist. Das ist praktisch.

Obwohl wir unsere Kinder in Deutschland dann wieder ganz verrückt machen, weil wir die arabischen Ziffern *entgegen* ihrem ursprünglichen Sinn lesen. Wir lesen links nach rechts, also von den höheren Stellen hin zu den Einern, statt von den Einern nach oben. Und bei den unteren beiden Stellen dann auch noch verdreht! Sagen: „Zweihundertsechsundvierzig“, statt „sechs–vierzig–zweihundert“. Und wundern uns dann, wieso der arabische Händler so gut im Rechnen ist und wir nicht.

Nun ja, das hab ich soweit enträtselt heute. Vieleicht erlebe ich es ja noch, dass dem arabischen Rechnen auch die arabische Rechendenke samt ihrer rechengerechten Aussprache folgen darf.

Addieren

Im Computer, in der ALU[2], ist die Sache klar. Da wird arabisch gerechnet und binär — von der niedrigsten Stelle zu den höheren. Und der Überlauf — das Carry–Bit — wird zum nächsthöheren Register übertragen und dort hinzuaddiert. Der Computer hat's da gut, darf „straight forward“ denken, er muss das ja auch nicht „deutsch“ aussprechen.

Das Binärsystem kennt man. Das kennt nur zwei Ziffern, symbolisiert als Null oder Eins, oder Low– und High–Voltage L und H. Oder off/on. Eingebürgert hat es sich, dafür 0 und 1 zu schreiben.

```
decimal 246 2 base ! . 11110110 ok
```

Aber bleiben wir mal im Dezimalsystem mit den zehn Zeichen.

```
decimal ok
: add ( n -- ) 15 0 do dup u. 1+ loop ; ok
0 add 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 ok
```

Da sieht man es sehr schön. Die fortgesetzte Addition von *Eins* führt nach der *Neun*, wenn der Ziffernvorrat verbraucht ist, zum Übertag in die nächsthöhere Stelle, und bei den Einern fängt alles wieder von vorn an.

Und was passiert dann an der Registergrenze? Wenn alle Bits schon gesetzt sind und nun noch eines dazugezählt wird? Klar, es muss einen Übertrag geben. Doch wohin? Wenn kein weiteres Register dafür bereitgestellt wurde, ist der Übertrag einfach futsch. Und unser Register fängt wieder bei Null an — siehe: Excurs.

Excurs: Was ist die größte Zahl in meinem Computer?

Im Computer sind die Register und auch der Stack nicht beliebig breit. 64–Bit–Maschinen sind inzwischen üblich. Mein Forth–System ist noch 32–bittig. Das Wort `true` legt einen Wert auf den `TOS`, in dem alle 32 Bit gesetzt sind:

```
true .
-1 ok
```

Doch hoppla, was ist das? Aha, der Punkt `.` „dot“ druckt den Zahlenwert in der Zweier–Komplement–Darstellung und mit Vorzeichen. Um den Wert „pur“ anzuzeigen, braucht es das `u.` „unsigned–dot“, also so:

```
decimal 2 base !   true u.
11111111111111111111111111111111  ok
```

Nun sieht man, dass es eine 32–Bit–Maschine ist. Und wir lernen, dass die `-1` die größte darstellbare Zahl des Systems ist. Nun addieren wir noch eine Eins dazu.

```
true 1+ u.
0 ok
```

Aha, das Register springt um auf Null. Oder anders gesagt, der Übertrag in das 33. Bit ist nicht mehr darstellbar in der 32–Bit–Maschine. Es bleiben die unteren 32 Bit, und die sind dann Null — wie es sich für den Binärzähler auch gehört. Eine 2–Bit–Maschine spränge schon bei Vier um, und die Eins in der dritten Stelle existierte da gar nicht mehr, entfiele einfach:

```
00
01
10
11
```

[1] Dazu benutzte ich Gforth unter Linux–Mint in einem 32–Bit–System.

[2] Arithmetic Logic Unit

```
1 00 <-- hier beginnt alles von vorne
1 01
1 10
1 11
1 00
...
```

Der Binärzähler zählt also im Kreis herum, der Addierer auch.

So bleiben wir bei der Addition immer im Zahlenvorrat all jener ganzen Zahlen, die die Maschine in ihren Registern darstellen kann. Jede Addition ganzer Zahlen ergibt wieder eine ganze Zahl. Und wenn das Ergebnis größer ist, als in ein Register passt? Entweder wird der Überlauf fallengelassen. Oder man fügt ein Auffang–Register an, dann wird *doppeltgenau* gerechnet.

Beipiele für einfachgenaues und doppeltgenaues Addieren ganzer Zahlen in Forth.

Wir rechnen weiterhin mit ganzen Zahlen[3]. Sind die Zahlen klein genug, ist die Addtion trivial.

```
12 3 + . 15 ok
```

Negative Zahlen gehen ebenso.

```
-12 3 + . -9 ok
```

Doch was passiert, wenn man die größte Integerzahl rechnen will?

```
-1 1 + u. 0 ok
```

Das ist vorzeichenrichtig gerechnet, 1 - 1 = 0.

Aber wie geht das ohne Vorzeichen? Wie kann ich den Überlauf in das 33. Bit mitnehmen? Hier kommen wir in die *doppeltgenauen* ganzen Zahlen. Leider expandiert das `d+` nichts von selbst:

```
.s <0> ok
-1 1 d+
:26: error: Stack underflow
-1 1 >>>d+<<<
```

Hier muss tatsächlich je eine leere Überlauf–Zelle vorher bereitgestellt werden. Beide Werte müssen also schon doppeltgenau sein, bevor `d+` angewendet werden kann.

```
( a ) -1 0 1 0 d+ d. 4294967296 ok
```

Dann klappt es. Zur Kontrolle:

```
-1 u. 4294967295 ok
```

Tatsächlich, die ganze Zahl in (a) wurde um eins größer.

Und nun die Eingabe mit dem Punkt in der Zahl, was ja in Gforth eine doppeltgenaue Zahl ablegt.

```
-1. 1. d+ d. 0 ok
```

Hm, das ist vorzeichenrichtig gerechnet, aber wir wollten ja *unsigned* rechnen. Das geht damit also nicht. Warum? `1.` legt zwar erwartungsgemäß `1 0` auf den Stack. Aber `-1.` legt `-1 -1` dort ab, eine „doppeltgenaue -1“ sozusagen. Und die Phrase *-1 s>d* macht dasselbe. Man muss also tatsächlich selbst eine Null–Zelle vorsehen für das höhere Byte, in das die doppeltgenaue ganzzahlige vorzeichenlose Addition überlaufen kann!

Dann gibt es da noch den Fall, dass eine einfache Zahl einer doppeltgenauen Zahl hinzugezählt werden soll. `m+` erledigt dies.

m+ (d1 n -- d2) „m-plus“

```
-1 0 1 m+  ok
.s <2> 0 1  ok
d. 4294967296  ok
```

Man muss also nicht unbedingt beide Summanden doppeltgenau machen. Es gibt ein extra Forthwort für diesen Fall der *mixed precision*.

Addieren, subtrahieren, multipizieren — es bleibt ganzzahlig

Sowohl beim Raufzählen (addieren) wie beim Runterzählen (subtrahieren) oder Malnehmen (multiplizieren), immer bewegt man sich im geschlossenen System der festen Registerbreite. Es gibt dabei keine „Zwischengrößen“, nur wieder ganze Zahlen.

Die Subtraktion ist wie die Addition. Auch dabei muss man die Überlauf–Zellen bereitstellen, wenn man vorzeichenlos doppeltgenau (unsigned double integer) bleiben will. `d-` und `d+` entsprechen da einander.

Für die Multiplikation wird gleich eine ganze Gruppe an Forth–Worten bereitgestellt. Für die unterschiedlichsten kombinierten Operationen, auch in *mixed precision*.

Der `*` ist einfachgenau. Nach `2147483647+1` wechselt es im 2er–Komplement–System auf `-2147483648`. Anders dargestellt, in Hex, ist also `$7FFFFFF` die größte positive 32–Bit–Zahl in meinem System.

```
hex
.s <1> 7FFFFFFF ok
dup decimal . 2147483647 ok
dup 1+ hex . -80000000 ok
dup 1+ decimal . -2147483648 ok
```

Das bedeutet, dass man ab der 2. Milliarde in Bereiche kommt, wo die Multiplikation falsch wird.

```
  2000   2000 * . 4000000  ok
 20000  20000 * . 400000000  ok
200000 200000 * . 1345294336  ok  <-- nee
200000 200000 m* d. 40000000000  ok
```

Schaun wir uns das wieder vorzeichenlos an:

```
-1 2 * u. FFFFFFFE ok <-- nee, natürlich nicht ok
```

Wie man sieht, findet kein Übertrag statt. Also besser so:

```
-1 2 um* ud. 1FFFFFFFE ok
```

`um*` nimmt einfachgenaue Werte vom Stack, rechnet aber doppeltgenau. Hexadezimal konnte man recht leicht sehen, was da passiert. Dezimal bleibt das eher undurchsichtig:

[3] engl.: integer

```
decimal -1 1 um* ud. 4294967295 ok
decimal -1 2 um* ud. 8589934590 ok
```

Ok, es ist das Doppelte.

Um nun nicht für jede Operation die Überläufe testen zu müssen, macht man sich also lieber gleich Funktionen, die intern doppeltgenau arbeiten, wobei einige auch gleich doppeltgenaue Ergebnisse liefern. Forth stellt die häufigsten Anwendungsfälle bereit.

*/	(n1 n2 n3 – n4) „star–slash“ n4=(n1*n2)/n3, with the intermediate result being double.
*/mod	(n1 n2 n3 – n4 n5) „star–slash–mod“ n1*n2=n3*n5+n4, with the intermediate result (n1*n2) being double.
m*	(n1 n2 – d) „m–star“
um*	(u1 u2 – ud) „u–m–star“
m*/	(d1 n2 u3 – dquot) „m–star–slash“

Wann man am besten welche Funktion nimmt, muss man ausprobieren. Und wenn diese Grundfunktionen des Standard–Forth nicht reichen, kann ein komplettes Set für das doppeltgenaue Rechnen geladen werden. [Luca Masini 2008]

Integer Division

Zahlen zu teilen gehört ebenfalls zu den Grundrechenarten. Doch passiert hier etwas Besonderes. Der Teiler kann so sein, dass bei der Division *keine* ganze Zahl herauskommt. Die 12 lässt sich durch 3 glatt teilen, die 3 passt 4 mal in die 12. Die 11 hingegen lässt sich nicht glatt durch 3 teilen, die 3 passt da nur 3 mal hinein und es bleibt ein Rest von 2. Soweit, so klar.

Binäre Zähler sind einfach, rauf wie runter, auch in verschiedenen Schrittweiten. Doch wie funktioniert ein binärer Teiler? Nun, dabei wird solange ein fester Wert abgezogen, der Teiler, bis der Zähler kleiner als der Teiler ist. Dann lässt sich der Teiler ja kein weiteres ganzes Mal abziehen. Täte man es dennoch, würde das Register durch die Null zählen und auf der anderen Seite weiter machen — Unterlauf. Also stoppt man dort und gibt den Rest an, der nicht mehr geteilt werden konnte. Sollte man meinen.

Doch in Forth bekommt man erstmal nur den ganzzahligen Quotienten, wenn man / als Operator für die Division nimmt.

```
decimal    ok
12 3 / .  4  ok
11 3 / .  3  ok
10 3 / .  3  ok
09 3 / .  3  ok
08 3 / .  2  ok
07 3 / .  2  ok
06 3 / .  2  ok
05 3 / .  1  ok
```

[4] Quote = Anteil, lat. quota (pars).

Mit anderen Worten: Man muss wissen, was man tut!

/ (n1 n2 – n) „slash“

Da ist `n1` der Divident, der zu teilende Wert, und `n2` der Divisor, der Teiler. Und `n` ist der Quotient, also der Anteil an Teilern, die in den Dividenden passten.[4]

Aber was ist mit dem Rest, wo ist der geblieben? Nun, der fiel vom Stack und ist weg. Man muss `/mod` benutzen, um auch den Rest `n3` auf dem Stack zu halten.

/mod (n1 n2 – n3 n4) „slash–mod“

Natürlich sieht man das umgekehrt, wenn mit . (dot) die Zahlen ausgegeben werden. Zuerst immer das, was oben liegt, der Quotient `n4`, dann der Rest `n3`.

```
11 3 /mod . .  3 2  ok
10 3 /mod . .  3 1  ok
09 3 /mod . .  3 0  ok
08 3 /mod . .  2 2  ok
...
```

Will man nur den Rest haben, wird `mod` benutzt.

mod (n1 n2 – n3) „mod“

```
decimal    ok
11 3 mod .  2  ok
10 3 mod .  1  ok
09 3 mod .  0  ok
08 3 mod .  2  ok
...
```

Also, die Division einfachgenauer Werte erfolgt mit dem Forthwort / und man erhält den Quotienten, jedoch ohne Rest. `/mod` liefert Quotient und Rest auf den Stack und `mod` allein nur den Rest.

Integer Division doppeltgenau, ohne Vorzeichen

Im Handbuch zum Gforth ist im Kapitel *Arithmetic* erklärt, welche Operationen Forth für die Division über die einfachgenauen Werte hinaus noch bereithält. Und da trifft man dann auf sowas:

um/mod	(ud u1 - u2 u3) „u–d–slash–mod“ ud=u3*u1+u2, u1>u2>=0

`ud` ist ein doppeltgenauer Divisor. Der wird durch den einfachgenauen Dividenden `u1` geteilt und man erhält `u2` als einfachgenauen Rest der Teilung und den einfachgenauen Quotienten `u3`, alle vorzeichenlos. Also ähnlich wie `/mod`, nur dass der Divisor größer ist. Das *„u“* bei alledem zeigt an, dass hierbei nicht auf das Vorzeichen geachtet wird. Es werden vorzeichenlose Berechnungen durchgeführt, wobei auch das höchste Bit mitverwendet und nicht als Vorzeichen betrachtet wird.

Das ist fein. Versuchen wir doch gleich mal die größtmögliche doppeltgenaue ganze Zahl vorzeichenlos zu dritteln.

```
decimal  ok
-1. 3 um/mod
 :2: error: Division by zero
-1. 3 >>>um/mod<<<
```

Oh, was ist das?

Im Standard steht dazu Folgendes:

> Divide ud by u1, giving the quotient u3 and the remainder u2. All values and arithmetic are unsigned. An ambiguous condition exists if u1 is zero or if the quotient lies outside the range of a single–cell unsigned integer.

Aha, das Ergebnis von `-1. 3 um/mod` ist zu groß. Das geht so also nicht. Was wir bräuchten, wäre ein

ud/mod (ud1 u1 -- u2rem ud3quot)
„u–double–slash–mod“

Doch das gibt es nicht im Standard! Was aber nicht weiter schlimm ist, denn im Gforth gibt es das `ud/mod`, und es ist ein High–Level–Wort[5]:

```
see ud/mod
: ud/mod
  >r 0 i um/mod r> swap >r um/mod r> ; ok
```

So, was passiert denn da? Nichts anderes als bei der gewohnten manuellen Berechnung. Man teilt erst die obere Portion, dann die untere. Also so wie im folgenden Beispiel:

```
129 : 3
12  : 3 = 4
  9 : 3 =  3
Ergebnis: 43
```

Damit kann man nun tatsächlich eine doppeltlange Zahl korrekt teilen — Jedenfalls bis 64 Bit im 32–Bit–System. Im 64–Bit–System entsprechend 128 Bit.

Wie sieht meine größte Zahl nun aus?

```
hex FFFFFFFFFFFFFFFF.  ok
.s <2> -1 -1  ok
decimal ud. 18446744073709551615  ok
```

Das ist 18.446.744.073.709.551.615 — sechsmal die Tausendergruppe, da kommen wir in die Exabyte–Speicheradressen–Klasse, 100.000x Terabyte!

Zur Erinnerung die 32 Bit:

```
hex FFFFFFFF decimal u. 4294967295 ok
```

4.294.967.295 — damit adressieren wir immerhin schon 4 Gigabyte.

Und nun endlich die Drittelung der 18 Exabytes.

```
-1. 3 ud/mod  ok
.s <3> 0 1431655765 1431655765  ok
ud. 6148914691236517205  ok
. 0  ok
```

6.148.914.691.236.517.205 Bytes, ca. 6 Exabytes, das kommt hin, und der Rest ist null, lässt sich also glatt teilen. Oder in HEX ausgedrückt:

```
hex  ok
-1. 3 ud/mod  ok
.s <3> 0 55555555 55555555  ok
ud. 5555555555555555  ok
. 0  ok
```

Hübsch, oder?

Machen wir doch gleich mal die Gegenprobe und rechnen zurück. Aber wie multipliziert man doppeltgenau? `D*` existiert nicht, nicht im Standard und auch nicht in Gforth. Also selber machen, [Luca Masini 2008]:

```
: d*    ( d1 d2 -- d3 )
  3 pick * >r  tuck * >r  um*  r> +  r> +    ;

6148914691236517205. 3. d*  ok
.s <2> -1 -1  ok
```

Stimmt.

Und wenn so eine Rechnung auch überlaufen würde, brauchen wir die *tripel–genauen Instruktionen*. Auch die hat Masini angegeben. Studiert man seinen High–Level–Code, versteht man bald, wie das geht und könnte weiter hoch gehen.

Mit dem so Erarbeiteten könnte man eigentlich zufrieden sein. Doch dann sieht man da auf einmal noch zwei seltsame Forthworte für die Division.

Floored und Symmetric Division — vorzeichenrichtige Division mit Rest, aber wie?

fm/mod (d1 n1 – n2 n3) „f–m–slash–mod“
Floored division: d1 = n3*n1+n2, n1>n2>=0 or 0>=n2>n1.

sm/rem (d1 n1 – n2 n3) „s–m–slash–rem“
Symmetric division: d1 = n3*n1+n2, sign(n2)=sign(d1) or 0.

Ja, was ist das denn nun schon wieder? Zwei verschiedene Divisionen? Das kam in der Schule nicht vor. In meiner jedenfalls nicht, damals ...

Untersuchen wir das mal. Der `divtest` teilt die Werte von -5 bis +6 durch 3 und zeigt den Quotienten und den Rest an. Jeweils für `fm/mod` und `sm/rem`.

```
decimal
: divtest ( -- )
cr ." fm/mod"
cr ." i  : "
   6 -5 do i 4 .r loop
cr ." quo: "
   6 -5 do i s>d  3  fm/mod 4 .r drop loop
cr ." mod: "
   6 -5 do i s>d  3  fm/mod drop 4 .r loop
cr
cr ." sm/rem"
```

[5] Die Forth–Worte `i` und `R@` sind synonym, also nicht wundern, was das `i` da soll.

```
cr ." i  : "
   6 -5 do i 4 .r loop
cr ." quo: "
   6 -5 do i s>d  3  sm/rem 4 .r drop loop
cr ." rem: "
   6 -5 do i s>d  3  sm/rem drop 4 .r loop
cr .s ;

divtest
fm/mod
i  : -5  -4  -3  -2  -1   0   1   2   3   4   5
quo: -2  -2  -1  -1  -1   0   0   0   1   1   1
mod:  1   2   0   1   2   0   1   2   0   1   2

sm/rem
i  : -5  -4  -3  -2  -1   0   1   2   3   4   5
quo: -1  -1  -1   0   0   0   0   0   1   1   1
rem: -2  -1   0  -2  -1   0   1   2   0   1   2
<0>  ok
```

Obwohl der Divisor bei beiden +3 ist, gibt es im Bereich des negativen Dividenden unterschiedliche Ergebnisse! Im Bereich der positiven Zahlen hingegen liefern beide das gleiche Ergebnis. Michael Pruemm schrieb in `www.nimblemachines.com`

> Since its 1983 standard (Forth–83), Forth has implemented floored division as standard. Interestingly, almost all processor architectures natively implement symmetric division.

Er hat dort übrigens auch eine hübsche Grafik, in der er zeigt, wie es sich damit verhält.

Schaun wir uns mal genau an, was da passiert. Kommt man von Minus–Unendlich, dem „floor“, geht der Quotient `quo` bei der floored division brav immer um einen `mod` rauf. Auch über die Null hinaus geht es so weiter:
-unendlich … -2 -1 0 1 2 … +unendlich
Und `mod` zählt immer positiv:
-unendlich … 0 1 2 0 1 2 … +unendlich
also bis in alle Ewigkeit.

Bei der symmetrischen Division hingegen ist das Ganze im negativen Bereich um ein Modul nach links verschoben. Das ist so, als ob die Null doppelt vorkäme. Einmal nähert man sich ihr von links und einmal von rechts. An ein und derselben Stelle liegt sozusagen einmal die -0 und dann die +0. Daher wohl der Name „symmetrische“ Division. Denn die Ergebnisse sind spiegelbildlich zur Mittellinie.

Kopfrechnen geht damit leichter. Der Satz „5 durch 3 geht einmal, Rest 2“ funktioniert so auf beiden Seiten, im Positiven und im Negativen, und man braucht dann nur das Minus vor beide zu setzen, und schon ist es auch vorzeichenrichtig.

Und auch technisch gesehen sind dann bei der symmetrischen Division beide Operationen gleich! Man merkt sich das oberste Bit, macht die Division und setzt das oberste Bit wieder ein, sowohl im Quotienten als auch im Rest, sehr einfach. Vermutlich ist das auch der Grund, warum das in Hardware auch so gemacht wird.

Bei der floored division muss man da schon um die Ecke denken. Der Satz „-5 durch 3 geht 2x, weil -5 eigentlich nur noch 1 mod von -6 weg ist“ leuchtet nicht sofort ein — mir auch nicht.

Obschon `fm/mod` eindeutig die systematischere Zahlenfolgeanordnung ist, logisch ästhetischer sozusagen und ohne den Widerspruch der zwei Nullen an einem Ort, ist es dennoch unpraktisch. Symmetrische Division vereinfacht das Rechenwerk, weil man es für die positiven wie negativen ganzen Zahlen gleichermaßen benutzen kann. Nur das Vorzeichen muss zwischengespeichert und wieder eingesetzt werden.

Will man einen solchen symmetrisch arbeitenden Hardware–Divider ins Forth einbinden, muss man bei der Implementation, z. B. auf einer MCU, darauf achten, so ein symmetrisches Ergebnis wieder „floored“ zu machen. Oder man rechnet „symmetrisch“ weiter. Solange man sich im Bereich positiver Zahlen bewegt, sind beide sowieso nicht unterschiedlich.

Noch ein paar Anmerkungen

<Ziffernfolge>.

Die Zahleneingabe mit einem *Punkt* dahinter wird in Gforth als *doppeltgenaue* Zahl interpretiert. Und so ein *double* wird durch zwei Zellen auf dem Datenstack repräsentiert.

```
hex -1.
.s <2> -1 -1
```

Ausgedruckt als unsigned double `ud.` wird daraus:

```
ud. FFFFFFFFFFFFFFFF ok
```

Nix auf dem Stack???

```
.s <0> ok
-1.-1 ok
.s <0> ok
```

Eingabe ok, dennoch nichts auf dem Datenstack? Dann ist das wohl woanders gelandet.

```
f.s <1> -1.0000000000E-1 ok
```

Aha! Die Eingabe ging also auf den Floatingpoint–Stack. Die Form `<VorzeichenZahl>.<VorzeichenZahl>` wird als Gleitkommazahl interpretiert. Mit oder ohne „e“ in der Ziffernfolge.

Helferlein

```
: .dhex ( n -- ) base @ >r hex ud. r> base ! ;
: .dbin ( n -- ) base @ >r 2 base ! ud. r>
base ! ;
decimal -1. .dhex FFFFFFFFFFFFFFFF ok
decimal -1. .dbin
```

[6] Hier wurden 64 „Einsen“ ausgegeben; drucktechnisch gekürzt.

```
1111111111111111...[6] ok
.s <0> ok
```

Anhang

Auszug aus dem Gforth–Handbuch.

5.5.5 Mixed Precision

`http://www.complang.tuwien.ac.at/forth/gforth/Docs-html/Mixed-precision.html#Mixed-precision`

Die Listen zeigen den Namen des Forthwortes, dann in runden Klammern den Stack–Kommentar, gefolgt vom kursiv gesetzten Wordset zu dem das Wort gehört und schließlich in Anführungszeichen die Aussprache des Wortes.

m+ (d1 n – d2) *double* „m–plus“

*/ (n1 n2 n3 – n4) *core* „star–slash“ n4=(n1*n2)/n3, with the intermediate result being double.

*/mod (n1 n2 n3 – n4 n5) *core* „star–slash–mod“ n1*n2=n3*n5+n4, with the intermediate result (n1*n2) being double.

m* (n1 n2 – d) *core* „m–star“

um* (u1 u2 – ud) *core* „u–m–star“

m*/ (d1 n2 u3 – dquot) *double* „m–star–slash“ dquot=(d1*n2)/u3, with the intermediate result being triple–precision. In ANS Forth u3 can only be a positive signed number.

um/mod (ud u1 – u2 u3) *core* „u–m–slash–mod“ ud=u3*u1+u2, u1>u2>=0

fm/mod (d1 n1 – n2 n3) *core* „f–m–slash–mod“ Floored division: d1 = n3*n1+n2, n1>n2>=0 or 0>=n2>n1.

sm/rem (d1 n1 – n2 n3) *core* „s–m–slash–rem“ Symmetric division: d1 = n3*n1+n2, sign(n2)=sign(d1) or 0.

5.5.2 Double Precision

For the rules used by the text interpreter for recognising double–precision integers, see Number Conversion.

A double precision number is represented by a cell pair, with the most significant cell at the TOS. It is trivial to convert an unsigned single to a double: simply push a 0 onto the TOS. Since numbers are represented by Gforth using 2's complement arithmetic, converting a signed single to a (signed) double requires sign–extension across the most significant cell. This can be achieved using `s>d`. The moral of the story is that you cannot convert a number without knowing whether it represents an unsigned or a signed number.

These words are all defined for signed operands, but some of them also work for unsigned numbers: `d+`, `d-`.

s>d (n – d) *core* „s–to–d“

d>s (d – n) *double* "d–to–s“

d+ (d1 d2 – d) *double* „d–plus“

d- (d1 d2 – d) *double* „d–minus“

dnegate (d1 – d2) *double* „d–negate“

dabs (d – ud) *double* „d–abs“

dmin (d1 d2 – d) *double* „d–min“

dmax (d1 d2 – d) *double* „d–max“

Links

`https://forth-standard.org/`

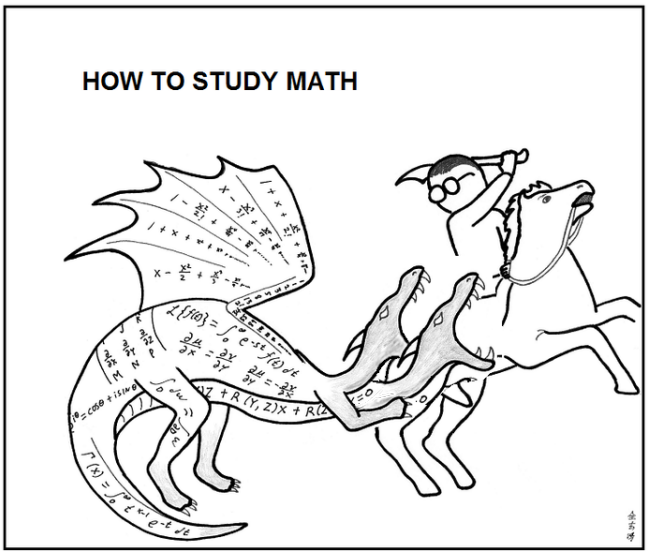
`http://www.complang.tuwien.ac.at/forth/gforth/Docs-html/Arithmetic.html#Arithmetic`

`https://wiki.forth-ev.de/doku.php/words:doubleext2_wordset`

`https://wiki.forth-ev.de/doku.php/words:d-number.fs` Double Number Word Set by Luca Masini 2008.

`https://wiki.forth-ev.de/doku.php/words:d-star-collection`

`https://www.nimblemachines.com/symmetric-division-considered-harmful/`



Don't just read it; fight it!

— Paul R. Halmos

Linear interpolierte Tabelle

Rafael Deliano

Um den Temperaturgang von Sensoren zu linearisieren, bieten Tabellen die größte Flexibilität. Man kann den Speicherbedarf deutlich senken, wenn man zwischen den Stützwerten interpoliert. Beliebt, weil mit geringstem Aufwand bei der Berechnung verbunden, sind Geraden (Abb. 1).

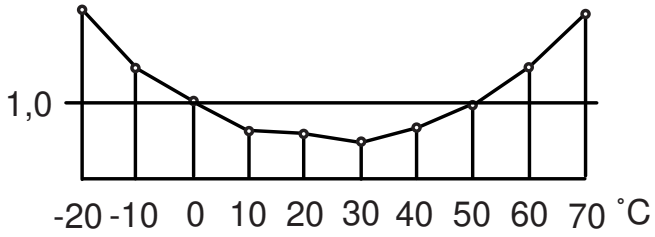


Abbildung 1: Sensorlinearisierung

Hier sind zwei dezimale Nachkommastellen bei Eingabe- und Ausgabewert vorgesehen. Erster Schritt der Berechnung ist Addition eines Offsets, damit verschwinden die negativen Zahlen (Abb. 2).

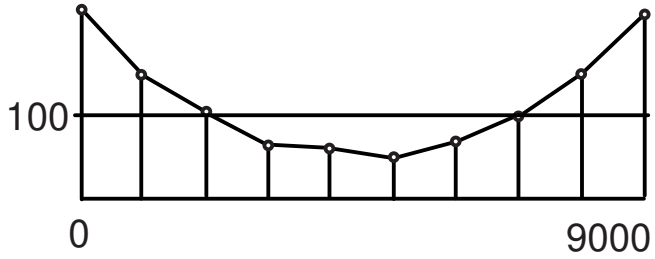


Abbildung 2: Mit X-Offset

Die `U/MOD`–Division ergibt außer einem Rest `REM` den Zeiger `X` auf den unteren Stützwert eines Segments (Abb. 6). Wenn der Rest `0` ist, hat man einen Stützwert getroffen, es ist keine Interpolation nötig.

Interpolation

Anhand der Stützwerte in der Tabelle muss man die beiden Fälle steigende oder fallende Kennlinie unterscheiden (Abb. 3). Für letztere ist der `MOD`–Index umzurechnen. Und es sind die beiden Werte für die Berechnung der Differenz (Abb. 4) zu vertauschen. Die Differenz wird danach anhand von `MOD` skaliert.

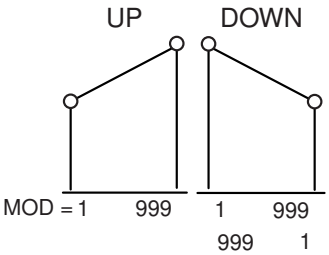


Abbildung 3: Steigende oder fallende Kennlinie

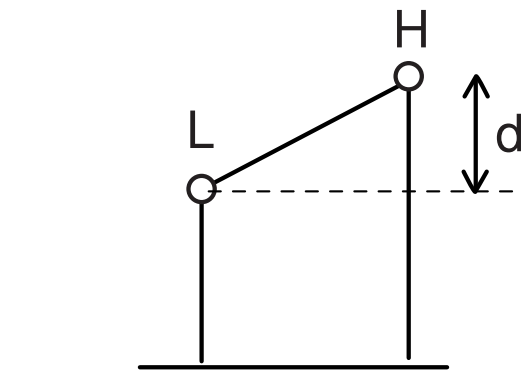
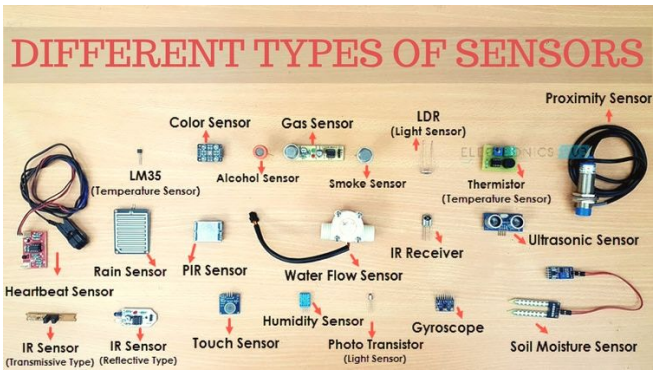


Abbildung 4: Differenz der Stützstellen



Negative Werte

In anderen Anwendungen können auch negative Werte in der Tabelle auftreten (Abb. 5). Man wird dann eine Tabelle mit positiven Werten verwenden und abschließend die Subtraktion eines Offsets vornehmen (Abb. 6).

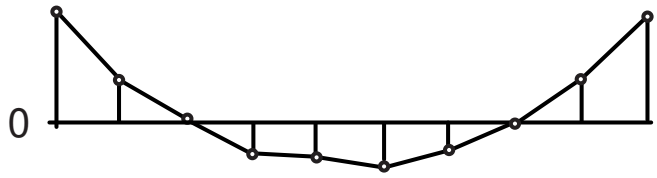


Abbildung 5: Mit Y–Offset

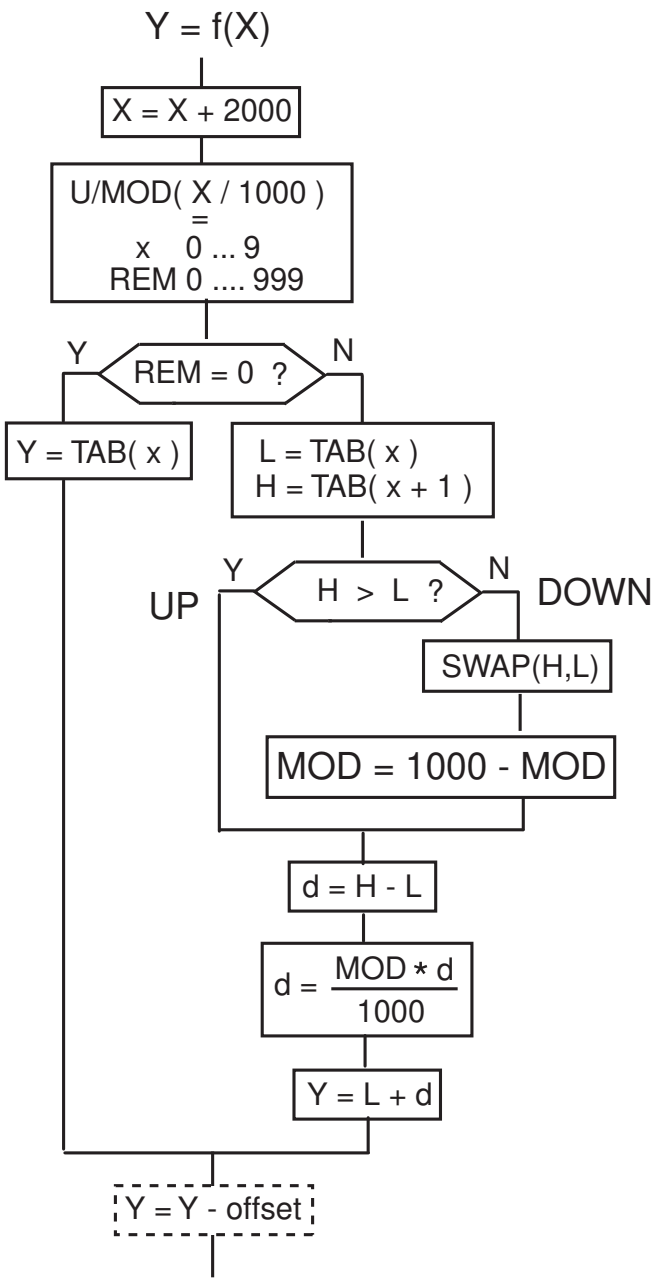


Abbildung 6: Flussdiagramm

Listing

```
1   <| \ SRC0.txt
2
3   DECIMAL
4
5   TABLE K-TAB  \ Sensor
6   145 , \ -20°C
7   130 , \ -10°C
8   110 , \  0°C
9   104 , \ +10°C
10  102 , \ +20°C
11   83 , \ +30°C
12   82 , \ +40°C
13   71 , \ +50°C
14   81 , \ +60°C
15  100 , \ +70°C
16
17  HEX
18
19  CR ." -> SRC1.TXT"
20  CR |>
21
22  <| \ SRC1.txt
23
24  \ Temperatur als Integer
25  \ um zwei Nachkommastellen skaliert
26  \ +25,00 -> D% 2500        K@
27  \ -25,00 -> D% 2500 NEGATE K@
28
29  \ Gain-Faktor K kommt als Integer
30  \ um zwei Nachkommastellen skaliert:
31  \ 125 -> 1,25
32
33  : K-TAB@   1<SHIFT K-TAB + @ ; \ ( i --- UN1 )
34
35  : (K@)                  \ ( MOD i-L --- UN3 )
36  DUP     K-TAB@          \ --- MOD i-L UN1 )
37  SWAP 1+ K-TAB@          \ --- MOD UN1 UN2 )
38  2DUP U>
39  IF   SWAP               \ --- MOD UN1 UN2 ) UN1 < UN2
40       ROT D% 1000 SWAP - \ --- UN1 UN2 MOD" )
41  ELSE ROT
42  THEN                    \ --- UN1 UN2 MOD )
43  SWAP HOPP -             \ --- UN1 MOD Dif )
44  U*                      \ --- UN1 Dif*MOD )
45  D% 1000 U/              \ --- UN1 DIF*MOD" )
46  + ;
47
48  : K@             \ ( UN1 --- UN2 ) UN1 = F830 ... 1B58
49  DUP  F830 U<
50  OVER 1B58 U>
51  LAND 88 ?ERROR   \ test for out of range
52  D% 2000 +        \ -2000 ... +7000 -> 0 ... 9000
53  0 D% 1000 U/MOD  \ --- MOD, UC1 ) UC1 = 0 ... 9 ;
54                                      MOD = 1 ... 999d
55  OVER
56  IF  (K@)
57  ELSE             \ MOD = 0 : no interpolation
58      SWAP DROP K-TAB@
59  THEN ;
60
61  CR ." -> "
62  CR |>
63
```

Uhr

Rafael Deliano

Klassisches, simples Demo-Projekt. Hier mal nicht als DCF77, sondern mit einem externen stabilen 10–MHz–Takt (Abb. 2).

Abbildung 1: Uhr

Das Programm kann in einer Endlosschleife laufen, die Durchlaufzeit muss aber exakt 1 Hz betragen. `WAIT` verzögert passend:

```
: RUN
  BEGIN ... WAIT AGAIN ;
```

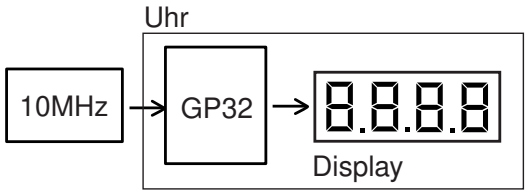


Abbildung 2: Blockschaltbild

Frequenzteiler

Der 16–Bit–Timer des GP32 erwies sich als nicht verwendbar. Also ist ein 2^n–Vorteiler in Hardware nötig, der die Geschwindigkeit reduziert (Abb. 3). Das ist auch sinnvoll, um TTL auf 5V zu wandeln. Der 32,768–kHz–Uhrenquarz ist passend für 2^{14}–Binärteiler ausgelegt, um 1 Hz zu erzeugen. Hier wird es komplizierter. Aber 10.000.000 ist glücklicherweise keine Primzahl. Ein krummer Teiler in Software ist an sich kein Problem; es ist ein Zähler, der nach Vergleich mit dem Endwert zurückgesetzt wird. Er sollte im Interrupt laufen und kann auch 24 Bit breit sein. Das Produkt 128*78125 würde einen angenehmen, glatten Teiler liefern. Aber der GP32 ist absehbar für 78–kHz–Interrupts nicht schnell genug.

Erhöht man den 2^n–Teiler um eine Stufe, ist eine Untersetzung von 0,5 nötig (Abb. 4). D. h., man teilt abwechselnd durch 39062 und 39063. Nach dem Schema kann man die IRQ–Frequenz immer weiter senken, aber das Programm wird zunehmend komplizierter. Und der Phasenjitter steigt, auch wenn der Mittelwert der Frequenz exakt bleibt. Hier wird die Variante 19,5 kHz verwendet (Abb. 5). Bei Sinussignal empfiehlt sich ein 74HC4060 als Teiler (Abb. 6).

Der Interrupt inkrementiert einen 8–Bit–Zähler `TICK` mit 1 Hz. In der Hauptschleife pollt `WAIT` diesen Zähler und dekrementiert ihn, wenn ein neuer Puls eingetroffen ist. Ein 8–Bit–Zähler ist besser als ein binäres Flag, wenn das Timing der Schleife überlastet ist — „race condition“. D. h., wenn die Ausführung länger als eine Sekunde dauern sollte. Unbearbeitete Pulse werden mittels Zähler quasi in einem FIFO gespeichert und verzögert abgearbeitet; sie gehen nicht verloren.

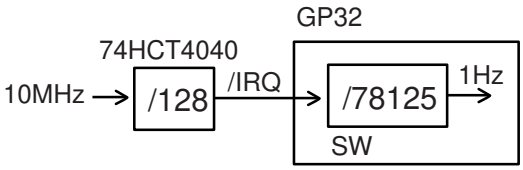


Abbildung 3: 1–Hz–Teiler

```
HW   SW
128 * 78125 = 10'000'000

256 * 39062,5 = 10'000'000
       ( 39062 + 39063 ) /2 = 39062,5

512 * 19531,25 = 10'000'000
      ( 19532 + 19531 + 19531 + 19531 ) /4 = 19531,25
```

Abbildung 4: Komplizierte Teiler

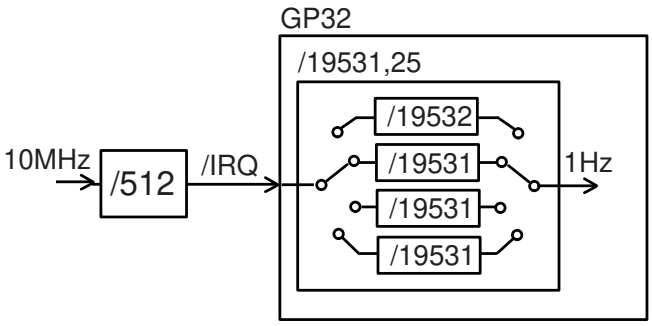


Abbildung 5: verwendete Variante

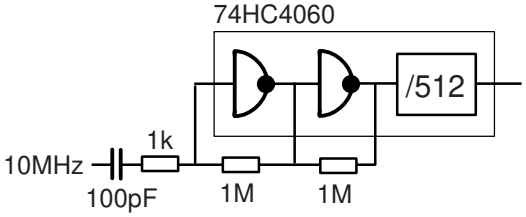


Abbildung 6: Pegelwandler für Sinus

Test

Der 16–Bit–Timer des GP32 kann Frequenzen mit hoher Auflösung am Portpin erzeugen. Das ist bequem für die Softwareentwicklung (Abb. 7). Mit 2,45 MHz Busfrequenz der CPU dauert der Interrupt etwa 3,5 … 4,5 µs (Abb. 8), schnell genug für den 51–µs–Takt.

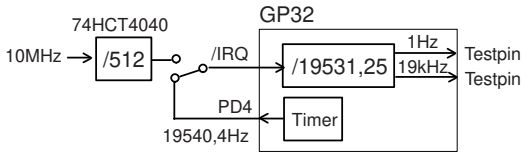


Abbildung 7: Test Teiler

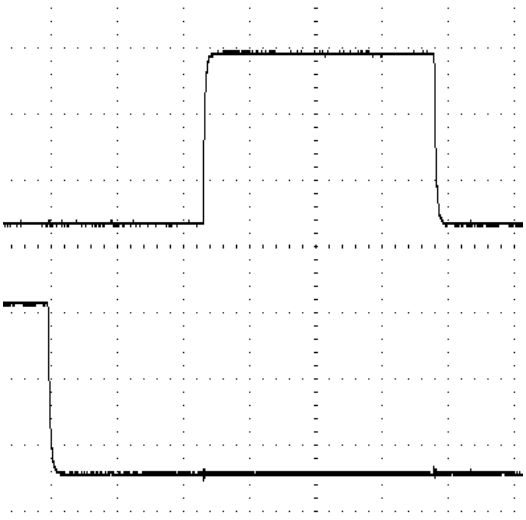


Abbildung 8: Timing 1 µs/div. Unten: fallende Flanke 19,5–kHz–Takt. Oben: Laufzeit IRQ über Testpin

Display

Hier wird ein HD44780–kompatibles LCD mit 2x16 Ziffern verwendet (Abb. 1). Es hat einen unkomplizierten, schnellen Zugriff, weil die Ziffern quasi „memory mapped“ einzeln anspechbar sind. Schreiben von Hardware außerhalb des Controllers ist langsam und sollte minimiert werden. Muss oft auch künstlich gebremst werden, um EMV–Probleme zu beheben. Das Breadboard war im Fundus bereits vorhanden (Abb. 9), die Beschaltung ist sehr einfach (Abb. 10).

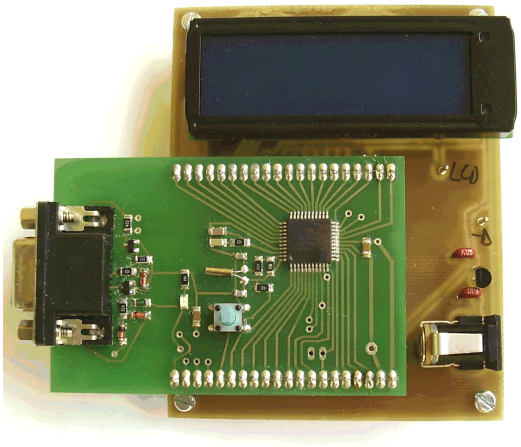


Abbildung 9: Breadboard

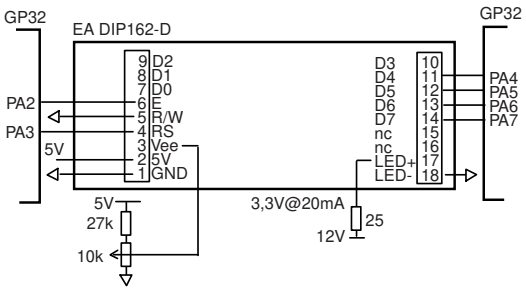


Abbildung 10: Schaltung Display

Zähler

Die unterste Sekunden–Ziffer verändert sich dauernd, sie wird einmal pro Sekunde geschrieben. Die 2. Stelle aber schon viel seltener. Alle übrigen Digits sind fast statisch. Der Jahreswechsel ist der Worst Case mit der maximal Rechenzeit. Der Inhalt des Displays befindet sich ursprünglich im RAM des Controllers („shadow register“), von wo er nach extern kopiert wird. Format ist 8–Bit–ASCII passend zum Display.

BCD

Erst in Taschenrechnern und bis spät in die 70er Jahre auch in anderen Geräten wurde meist 4–Bit–BCD verwendet, wenn Daten auf Displays ausgegeben werden müssen. Für numerische Berechnungen ist natürlich eine Binärzahl das bessere Datenformat. Hier sind aber nur Inkrements erforderlich, BCD bietet sich daher an. Allerdings besser 8–Bit–BCD, das dem ASCII entspricht, d. h. mit Ziffern 30h … 39h.

Die Situation ist oft ähnlich, aber einfacher, wenn man 5–stellige Seriennummern hochzählt (Listing 1). Da ist der Überlauf immer dezimal einheitlich von 39 auf 30. Bei Time&Date ist eine Routine sinnvoller, bei der man den Überlaufwert mit `MOD` steuert:

```
DIGIT+    ( C Adr Mod --- C )
```

Diese Routine kopiert jede veränderte Ziffer auch auf das LCD. Das minimiert die Zugriffe aufs Display. Ein

steuerbarer MOD–Wert löst die Probleme für die 0...59 Zähler von Sekunden und Minuten (Abb. 11). Für die 0...23 Stunden ist aber bereits wieder ein kleiner Patch nötig (Listing 2). Der Zähler für Tage ist wie zu erwarten besonders kompliziert, weil die Monate unterschiedlich viele Tage haben. Da hilft nur eine Tabelle. Februar hat zusätzlich, abhängig davon, ob Schaltjahr ist oder nicht, eine unterschiedliche Länge. Für die Bestimmung des Schaltjahres wird hier nur geprüft, ob es durch 4 teilbar ist. Bis 2099 ist das ausreichend.

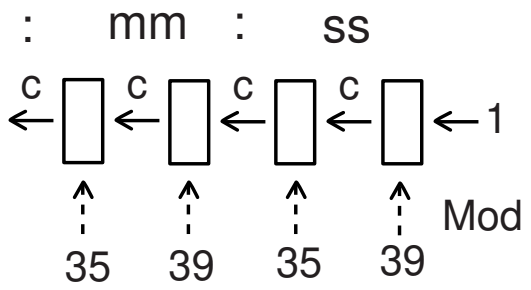


Abbildung 11: Zähler Sekunden & Minuten

Sowohl Tage als auch Monate zählen ab 01 und nicht ab 00. Auch dafür ist ein Patch vorhanden. Am unkompliziertesten ist der Zähler für Jahre. Er erzeugt pro forma ein Carry–Flag. Der einzelne Sekunden–Inkrement mit Zugriff aufs Display dauert auf einem 2,45–MHz–GP32 mit laufendem IRQ etwa 11 ms. Als Worst Case wurde der Inkrement von *23:59:59 31.12.9999* getestet, er dauert 140 ms.

Sommerzeit

Die ist hier nicht berücksichtigt. Man könnte aber für eine manuelle Umschaltung einen Taster oder einen Schalter vorsehen, der das dann auch im Display anzeigt. Auf Knopfdruck die zwei Stunden–Ziffern mit HOUR+ zu inkrementieren ist einfach. Um am Sommerende entsprechend zu dekrementieren, muss erst ein DIGIT- implementiert werden, auf das HOUR- aufsetzt.

Links

https://www.sprut.de/electronic/referenz/frequenz/rubidium/rubidium.html

https://en.wikipedia.org/wiki/Rubidium_standard

Listings

Listing 1: Seriennummer

```
<| \ 5 digit decimal serial number

5 ZVARIABLE SN

: DIGIT+               \ ( carry >adr --- carry )
SWAP
IF SN + DUP C@         \  --- carry adr ASCII )
   DUP 39 =
   IF  DROP 30 SWAP C! 1
   ELSE     1+ SWAP C! 0
   THEN
ELSE DROP 0 THEN ;

: SN+                  \ (  ---  ) increment SN
1 4 DIGIT+
  3 DIGIT+
  2 DIGIT+
  1 DIGIT+
  0 DIGIT+
IF SN 5 30 FILL THEN ; \ clear on overflow
|>
```

Listing 2:

```
<| \ init.txt

\ /IRQ-pin: input CLK
PA 2 DCONSTANT LCD_EN       \ out 0
PA 3 DCONSTANT LCD_RS       \ out 0
PA 4 DCONSTANT LCD_D0_D4    \ out 0
PA 5 DCONSTANT LCD_D1_D5    \ out 0
PA 6 DCONSTANT LCD_D2_D6    \ out 0
PA 7 DCONSTANT LCD_D3_D7    \ out 0
PD 2 DCONSTANT T0           \ out 0
PD 3 DCONSTANT T1           \ out 0
PD 5 DCONSTANT T2           \ out 0

: INIT                   \ (  ---  )
B% 11111111 DPA C!   B% 00000000  PA C!
B% 11101111 DPD C!   B% 00000000  PD C! ;

CR ." -> Disp.txt" CR |> <| \ Disp.txt

\ R/W=0  pin wired low
\ PA 2 DCONSTANT LCD_EN       \ out 0  CS
\ PA 3 DCONSTANT LCD_RS       \ out 0  Adr
\ PA 4 DCONSTANT LCD_D0_D4    \ out 0
\ PA 5 DCONSTANT LCD_D1_D5    \ out 0
\ PA 6 DCONSTANT LCD_D2_D6    \ out 0
\ PA 7 DCONSTANT LCD_D3_D7    \ out 0

: PULSE-EN   LCD_EN B1! LCD_EN B0! 1 MSEC ;

: i4LCD!        \ ( bbbb0000 ---  ) RS=0
                      PA C! PULSE-EN ;

: iLCD!     \ ( UC1 ---  ) RS=0  instruction
DUP F0 AND            PA C! PULSE-EN
    4<SHIFT           PA C! PULSE-EN ;

: dLCD!         \ ( UC1 ---  ) RS=1  data
DUP F0 AND  B% 1000 OR PA C! PULSE-EN
    4<SHIFT B% 1000 OR PA C! PULSE-EN ;

: LCD!   \ ( UC1 addr ---  ) addr: 00 ... 1F
DUP 10 AND IF 30 + THEN
80 OR iLCD! dLCD! ;

: CLEAR   B% 0001 iLCD! 2 MSEC ; \ Clear
: LCD-OFF B% 1000 iLCD! ;        \ LCD off
: LCD-ON  B% 1100 iLCD! ;        \ LCD on

: BLANK      \ (  ---  )
0F 00 DO 20 I LCD! LOOP
4F 40 DO 20 I LCD! LOOP ;

: INIT-DISPLAY        \ (  ---  )
B% 00110000  i4LCD!  5 MSEC
B% 00110000  i4LCD!
B% 00110000  i4LCD!
B% 00100000  i4LCD!
B% 00100000  i4LCD!   \ 28  Function set
B% 10000000  i4LCD!
B% 00000000  i4LCD!   \ 08  Display on/off
B% 10000000  i4LCD!    B=0  Blink off
B% 00000000  i4LCD!   \ 01  Clear Display
```

```
64  B% 00010000  i4LCD!
65  B% 00000000  i4LCD!   \ 06 Entry Mode Set
66  B% 01100000  i4LCD! ;
67
68  : INIT-LCD            \ (  ---  )
69  INIT-DISPLAY LCD-OFF BLANK BLANK LCD-ON ;
70
71  \ init init-lcd 31 0 lcd!  32 1 lcd! \ test
72
73  CR ." -> test-clk .TXT" |> <| \ test-clk.txt
74
75  :CODE 19,5kHz   \ (  ---  )  19,50404 kHz
76  0020  B% 00110000 #. MOV,  \ T1SC   Stop reset
77  0023           00 #. MOV,  \ T1MODH
78  0024  D% 62       #. MOV,  \ T1MODL
79  25 B% 10010100 #. MOV,     \ T1SC0
80  0020  B% 00000000 #. MOV,  \ T1SC Prescaler /1
81                    RTS,
82  CODE;
83  CR ." -> IRQ.TXT" CR |> <| \ IRQ.txt
84
85  HEX
86  3 ZVARIABLE M       \ counter, commutator
87  1 ZVARIABLE TICK    \ semaphore 8 bit counter
88
89  :CODE WAIT            \ (  ---  )
90  1 $: TICK      LDA,   \ wait for TICK = 01
91  1 $            BEQ,
92       TICK      DEC,   \ decrement TICK
93                 RTS,
94  CODE;
95
96  :CODE CLK-IRQ
97               1D 2 MBS,   \ clear interruptflag
98       M 1+       INC,   \ inc M
99  1 $             BNE,
100      M          INC,
101 1 $:
102 3 $  M 2+      0 BBS,   \ branch on bit set
103 3 $  M 2+      1 BBS,   \ branch on bit set
104      M 1+       LDA,   \ test M for 19532
105      D% 19532 FF AND  #. CMP,
106 2 $             BNE,
107      M          LDA,
108      D% 19532 8SHIFT> #. CMP,
109 2 $             BNE,
110 4 $             BRA,
111 3 $: M 1+       LDA,   \ test M for 19531
112      D% 19531 FF AND  #. CMP,
113 2 $             BNE,
114      M          LDA,
115      D% 19531 8SHIFT> #. CMP,
116 2 $             BNE,
117 4 $: M 2+       INC,   \ commutator
118      M          CLR,   \ clear M
119      M 1+       CLR,
120      TICK       INC,   \ increment TICK
121 2 $:            RTI,
122 CODE;
123
124 :CODE INIT-IRQ   \ (  ---  ) IRQ-interrupt
125 M    CLR,   M 1+ CLR,  M 2+ CLR,  TICK CLR,
126 1D         CLR,   \ enable falling edge
127            CLI,   \ interrupt enabled
128            RTS,
129 CODE;
130
131 \ Vector in SYS-EE to IRQ
132 CC        80ED EC!   \ JMP,-Opcode
133 ' CLK-IRQ 80EE E!
134
135 CR ." -> counter.txt" CR |>  <| \ counter.TXT
136
137 HEX
138
139 $HEAP CONSTANT DISP  \ 32d byte RAM
140
141 : DISP.     \ (  ---  )
142 CR D% 31 0 DO DISP I + C@ EMIT LOOP ;
143
144 \ 00 00 00   first line of LCD
145 \ 01 34 67
146 \ hh:mm:ss
147 \ 00 00 00
148 \ .. .. ..
149 \ 23 59 59
150
151 \ 11 11 1111  second line of LCD
152 \ 01 34 6789
153 \ dd.mm.yyyy
154 \ 01 01 2000
155 \ .. .. ....
156
157 : DISP+C@  DISP + C@ ; \ ( >addr --- ASCII )
158
159 : DISP+C!             \ ( ASCII >addr --- )
160 2DUP DISP + C!
161      LCD! ;            \ output to LCD
162
163 TABLE INIT-LCD-TAB
164 30 C, \ 00  0
165 30 C, \ 01  0
166 3A C, \ 02 ":"
167 30 C, \ 03  0
168 30 C, \ 04  0
169 3A C, \ 05 ":"
170 30 C, \ 06  0
171 30 C, \ 07  0
172 20 C, \ 08
173 20 C, \ 09
174 20 C, \ 0A
175 20 C, \ 0B
176 20 C, \ 0C
177 20 C, \ 0D
178 20 C, \ 0E
179 20 C, \ 0F
180 30 C, \ 10  0
181 31 C, \ 11  1
182 2E C, \ 12 "."
183 30 C, \ 13  0
184 31 C, \ 14  1
185 2E C, \ 15 "."
186 32 C, \ 16  2
187 30 C, \ 17  0
188 30 C, \ 18  0
189 30 C, \ 19  0
190 20 C, \ 1A
191 20 C, \ 1B
192 20 C, \ 1C
193 20 C, \ 1D
194 20 C, \ 1E
195 20 C, \ 1F
196
197 : INIT"      \ (  ---  )
198 INIT INIT-LCD
199 1F 0 DO INIT-LCD-TAB I + C@ I DISP+C! LOOP ;
200
201 : DIGIT+     \ ( carry >addr mod  --- carry )
202 ROT
203 IF SWAP DUP DISP+C@  \  --- mod >addr ASCII )
204    ROT OVER =       \  --- >addr ASCII flag )
205    IF  DROP 30 SWAP DISP+C! 1
206    ELSE     1+ SWAP DISP+C! 0
207    THEN
208 ELSE 2DROP 0 THEN ;
209
210 : SEC+      \ ( c --- c ) inc seconds
211 7 39 DIGIT+  6 35 DIGIT+ ;
212
213 : MIN+      \ ( c --- c ) inc minutes
214 4 39 DIGIT+  3 35 DIGIT+ ;
215
```

```
: HOUR+     \ ( c --- c ) inc hours
0 DISP+C@ 32 =   \ test for "2x"
IF    1 33 DIGIT+  0 32 DIGIT+
ELSE  1 39 DIGIT+  0 39 DIGIT+ THEN ;

TABLE MONTH-DAYS
00 C, \ days mm
31 C, \ 31   01 Jan
00 C, \ 28   02 Feb   29 if MOD(YEAR/4)=0
31 C, \ 31   03 Mar
30 C, \ 30   04 Apr
31 C, \ 31   05 May
30 C, \ 30   06 Jun
31 C, \ 31   07 Jul
31 C, \ 31   08 Aug
30 C, \ 30   09 Sept
31 C, \ 31   10 Oct
30 C, \ 30   11 Nov
31 C, \ 31   12 Dec

: DISP@*+              \ ( UN1 k adr --- UN2 )
DISP+C@ 0F AND U* DROP + ;

: MD@             \ (  --- UC1 ) UC1 = 30 , 31
0 D%    1 14 DISP@*+
  D%   10 13 DISP@*+ MONTH-DAYS + C@ ;

: LY?           \ (  --- flag ) flag = 1 if LY
0 D%    1 19 DISP@*+
  D%   10 18 DISP@*+
  D%  100 17 DISP@*+
  D% 1000 16 DISP@*+ B% 11 AND LNOT ;

: DAY+            \ ( c --- c ) inc day
10 DISP+C@ 32 =  \ test for dd="2x" and mm=Feb
13 DISP+C@ 30 = LAND
14 DISP+C@ 32 = LAND
IF LY?               \ test for leap-year
  IF   11 39 DIGIT+  10 32 DIGIT+
  ELSE 11 38 DIGIT+  10 32 DIGIT+
  THEN
ELSE
  10 DISP+C@ 33 =    \ test for dd="3x"
  IF    11 MD@ DIGIT+  10 33 DIGIT+
  ELSE  11 39  DIGIT+  10 33 DIGIT+ \ "0x""1x"
  THEN
THEN
10 DISP+C@           \ patch "00" to "01"
11 DISP+C@ OR 0F AND LNOT
IF 31 11 DISP+C! THEN ;

: MONTH+            \ ( c --- c ) inc month
13 DISP+C@ 31 =     \ test for "1x"
IF   14 32 DIGIT+  13 31 DIGIT+
ELSE 14 39 DIGIT+  13 39 DIGIT+     \ "0x"
THEN
13 DISP+C@          \ patch "00" to "01"
14 DISP+C@ OR 0F AND LNOT
IF 31 14 DISP+C! THEN ;

: YEAR+             \ ( c --- c ) inc year
19 39 DIGIT+   18 39 DIGIT+
17 39 DIGIT+   16 39 DIGIT+ ;

: RUN     \ (  ---  )
INIT" 19,5KHZ INIT-IRQ
BEGIN
1 SEC+   MIN+ HOUR+
  DAY+ MONTH+ YEAR+ DROP
WAIT TERMINAL? UNTIL ;

CR ." -> summertime.txt "  CR |>
<| \ summertime.txt

HEX

: DIGIT-      \ ( carry >addr mod  --- carry )
ROT
IF SWAP DUP DISP+C@   \  --- mod >addr ASCII )
   DUP 30 =
   IF DROP      DISP+C!      1
   ELSE 1- SWAP DISP+C! DROP 0
   THEN
ELSE 2DROP 0 THEN ;

: HOUR-     \ ( c --- c ) dec hours
0 DISP+C@ 30 =
1 DISP+C@ 30 = LAND \ test for "00"
IF    1 33 DIGIT-  0 32 DIGIT-
ELSE  1 39 DIGIT-  0 39 DIGIT- THEN ;

CR ." ->  "  CR |>
```

Abbildung 12: Rubidium–Frequenznormal des Autors

One–Letter–Words

Martin Bitter

„Four–Letter–Words“ haben in den Medien einen schlechten Ruf. Gibt es noch Schlimmeres? „One–Letter–Words“ in Forth! Das hängt, wie oft in ethisch–philosophischen Fragen, ganz von der Wertebasis ab, mit der man operiert.

Das Problem

Die MCU eines Prüfstands schickt mittels des Printbefehls[1] . über eine serielle Schnittstelle numerische Daten zu einem angeschlossenen Standardterminal. Standard soll hier bedeuten: ANSI–kompatibel zum VT100, wie z. B. e4thcom, picocom und viele andere. Die Daten sollen nun weiter ausgewertet und dargestellt werden. Die Menge der darstellbaren Daten in einer bestimmten Zeit ist beschränkt durch die Verbindungsrate von 115.200 Baud. Die Daten werden vorerst als Ziffernstring gesendet.

Lösungsansätze

Differenzbildung

Die Menge der Daten soll verkürzt werden. Dies geschieht unter anderem dadurch, dass nach einer Initialisierung nur noch Differenzen des aktuellen Datums zum vorherigen Datum übertragen werden.

Schon mal ein kleiner Erfolg, statt 7 Bytes benötigt ein Datum jetzt nur noch 4 Bytes.

Roh–Bytes

Die Daten werden in Bytes zerlegt und diese einzeln übertragen. Dazu muss ein eigenes Terminalprogramm geschrieben werden, das Rohbytes entsprechend verarbeiten kann. Das war mir erst einmal zu aufwendig, ist aber — späte Erkenntnis — wohl der Königsweg.

Große Zahlen–Basis

Aber halt: Forth bietet ja die Möglichkeit, Zahlen in (fast) beliebigen Zahlensystemen darzustellen. Binär, oktal, dezimal, hexadezimal – um nur die gängigsten zu nennen. Also ganz schnell die Zahlenbasis in der MCU auf einen hohen Wert gesetzt, zum Anfang mal auf 48, und — tatsächlich: Es sieht zwar aus wie Kauderwelsch, aber die Länge eines Datums schrumpft auf maximal 2 Bytes, oft ist es nur ein Byte lang. Wunderbar!

Meinen und Verstehen – Wunsch und Wirklichkeit

Ein Bereich der Linguistik beschäftigt sich mit dem Unterschied zwischen Meinen (Sender) und Verstehen (Empfänger) und was dabei alles schieflaufen kann.

Das trat sofort ein in meinem Entwurf. Der Empfänger, Gforth auf dem PC[2], verstand was ganz anderes als der Sender, die MCU, meinte. Wie das? Beide hatten doch die gleiche Basis — `&48 base !` — bekommen?

Das gesendete Datum wird beim Empfänger, Gforth, in einen Puffer gelegt und mit `evaluate` ausgewertet. Somit landet das Datum als Zahl zur Weiterverabeitung auf dem Stack, oder? Eine Weile ging das gut. Doch in nicht reproduzierbaren Abständen verabschiedete sich Gforth recht freundlich. Mit freundlich meine ich, es gab ausführliche Fehlermeldungen aus. Diese Meldungen waren aber *nicht* immer die gleichen!? Rätselhaft!

Versuche mit anderen Zahlen–Basen brachten marginale Verbesserungen, selbst bei Hexadezimalzahlen kam es zu solchen Abstürzen. Nur mit Dezimalzahlen funktionierte die Übertragung wie es sein soll. Das gab zu Denken ...

Irgendwann fiel es mir wie Schuppen von den Augen: „****“[3].

Der vom `evaluate` angestoßene Interpreter ruft ja zuerst `find` auf und erst dann, wenn `find` kein passendes Wort findet, wird `number` aufgerufen, das dann im Erfolgsfall die entsprechende Zahl auf den Stack legt.

Was nun, wenn eine Zeichenkette, die zwar ein Ziffernstring sein soll, dennoch als Wort gefunden wird? Nun, dann wird versucht, das Wort auszuführen, was in den allermeisten Fällen zu glücklicherweise erkennbaren Fehlern führt.

Z. B. bei der Zahlenbasis „16“. Die Zahl sei „dezimal 11“, was in hexadezimaler Darstellung ein „B“ ist. Dieser String wird nun gesendet und an `evaluate` übergeben, Gforth[4] findet das Wort `B` und führt es aus. Dumm gelaufen.

Umgehungen des Problems?

Das Übertragungsproblem mit diesen Ziffernstrings kann gelöst werden, indem:

- Die gesendeten Ziffernstrings mit einem Präfix oder einer führenden Null versehen werden. Das macht die Strings aber länger.
- Die empfangenen Ziffernstrings, bevor sie an `evaluate` übergeben werden, mit einem Präfix oder einer führenden Null versehen werden. Eine gute Lösung.

[1] Der Forth–„dot“ ist gemeint.

[2] Die Datei serial.fs stellt die Verbindung von Gforth mit der MCU bereit; ist in der Gforth–Distribution enthalten.

[3] Four–Letter–Word zensiert. (Der Sätza)

[4] Gforth 0.7.9_20190829, Copyright (C) 1995-2018 Free Software Foundation, Inc.

- Auf den Einsatz von `evaluate` zu Gunsten von `number` verzichtet wird. Das ist nicht ganz so bequem.
- Die Reihenfolge vorübergehend beim Interpreter umgekehrt wird. Zuerst `number` und dann `find` ausführen. Ein unnötig tiefer Eingriff ins Forthsystem.
- Die Daten als Bytes (nicht als Ziffern vulgo Chars) gesendet werden.
- Die anfallenden Daten gestreckt werden. Stroboskoplösung — es wird nicht jeder Messwert übertragen. Da sich die Daten annähernd periodisch wiederholen, reicht es aus, zum Beispiel nur jedes dritte Datum zu übertragen.

Das Problem ist dann ganz anders gelöst worden: durch die Realität! Der Motor drehte gar nicht so hoch wie erwartet. Die Datenrate bei dezimaler Zifferndarstellung reicht für die Übertragung aus — wenn auch knapp.

Fazit und mal wieder eine philosophische Erkenntnis

Ich muss mir noch deutlicher machen, welche Unterschiede Forth im Allgemeinen zwischen Zahl und Ziffer macht. Oft ist der einfache Weg — hier `evaluate` — nur scheinbar einfach!

Prinzipiell bleibt festzuhalten: Wenn die Basis so hoch gesetzt wird, dass man mit dem Ziffernvorrat in den Bereich von Buchstaben kommt, besteht die Gefahr, dass Zahlendarstellungen, also Ziffernstrings, entstehen, die als gemeines Wort interpretiert werden können.

Wie man etwas wahrnimmt, hängt letztlich also tatsächlich von der eigenen Wertebasis ab!

Listing

```
1   \ Find all words that are one, two, three or n letters
2   \ wide.    See also: mwords.fs
3
4   : name-length? ( n xt -- n )
5       name>string 2 pick =
6       IF space over type ELSE drop THEN ;
7
8   : x-letter-words ( n -- )
9       get-current ['] name-length? map-wordlist
10      drop ;
11
12  : zwei-wort-listen ( -- )
13      2 0 DO I 1 + dup cr . cr x-letter-words LOOP ;
```

Einblicke

Fein wäre es, ließe sich auflisten, welche One–Letter–Words Gforth so verwendet. Denn das sind eine Menge. Und viele davon sind nicht mal Gforth spezifisch, sondern gehören zum Standard. Mit dem tollen `map-wordslist` lässt sich das schnell untersuchen. Im Listing ist gezeigt, wie das geht. Hier die damit erzeugte Ausgabe:

```
zwei-wortlisten [ret]

1

g b n l ) { ? ( ; : ] [ ' , \ ( . # ! @ > < =
/ * - + k j i

2

lb l1 A> <A \c bt .% k9 k8 k7 k6 k5 k4 k3 k2
k1 +t ~ pn #n ;> s+ th x, l, w, x@ l@ w@ /l /w
]] [[ ]L sh $? $. {: of C" .( f pi f. f$ f, e?
.4 .s ." s" ;] [: DO IF $@ $! #! IS TO ," S, A,
2, c, :, \G u. d. .r #s #> <# ms ns cr on A!
dp bl r@ o> >o !@ x! l! w! >l v* f/ f* f- f+
f@ f! f> f< f= 2@ 2! c! c@ +! r> >r d> d< d=
u> u< u= >= <= <> 0> 0< 0= u/ or d- d+ m+ m*
2/ 2* */ 1- 1+ i' ;s ok
```

Wieviele Basen gibt es wohl, in denen eine Zahl als `DO` dargestellt wird?

Links

`https://www.gnu.org/software/gforth/`

Wieviele Basen bilden eine Zahl mit dem Ziffernstring DO ab?

Das hängt davon ab, wie groß die Basis werden kann. Nennen wir diesen Wert `maxbase`. Einige Überlegungen: Wenn der Ziffernvorrat `0`...`9` erschöpft ist, werden Buchstaben und ASCII–Codes verwendet. Das `D` ist schon bei der durchaus geläufigen Zahlenbasis &16 (hexadezimal) enthalten. Diese endet bei `F`. Von `F` bis `O` sind es im Alphabet 9 Schritte. Die Basis &16 um &9 erhöht auf &25 enthält also bereits das `O` als Ziffer. Ergo können alle Basiswerte, die >=25 sind, die Ziffernfolge `DO` enthalten.

Probe:

```
25 base ! 0do dup . decimal . DO 713 ok
24 base ! 0do dup . decimal .
*the terminal*:31: error: Undefined word
```

Wir sehen: Die Vermutung stimmt. Wir sehen auch: Gforth macht bei der Zahlerkennung keinen Unterschied zwischen Groß- und Kleinschreibung.

Die Antwort lautet also: $maxbase - 24$

Zum Nachdenken: Das Beispiel funktioniert nur im interpretierenden Modus von der Eingabezeile aus. Kompilieren (`: test 25 base ! 0do dup . decimal . ;`) lässt es sich nicht. Martin Bitter

Tali–Forth–2 — ein modernes Forth für einen modernen Klassiker

Carsten Strotmann

Selten, dass mich ein Handbuch gleich in den ersten Seiten überzeugt. Bei Tali–Forth–2 war das mit dem Satz "It is a well–established fact that humanity reached the apex of processor design with the 6502 in 1976" der Fall. Als Kind der 1980er Jahre, in denen ich Maschinensprache auf einem Atari 800XL mit eben jener 6502–CPU gelernt hatte, musste ich dieses Forth lieben.

Für die 6502–CPU gibt es schon viele Forth–Systeme, angefangen mit dem *6502–FIG–Forth* über das *VolksForth* der Forth–Gesellschaft bis zu neuen Systemen wie *Durex–* oder *Pettil–Forth*. Was Tali–Forth–2 in dieser Reihe besonders macht, ist sein modernes Forth nach dem ANSI– und neuestem Forth–200x–Standard. Ziel ist es, ein kompatibles Subset von GNU/Forth (Gforth) zu implementieren, sodass Programme zwischen Tali–Forth–2 und GNU/-Forth kompatibel sind. Tali–Forth–2 benutzt den erweiterten 6502–Befehlssatz der CMOS–Version dieser CPU (65C02). Der 65C02 wird auch heute noch hergestellt und verkauft[1] und ist z. B. auf Mikrocontroller–Boards der Firma *Western Design Center* zu finden. Der 65C02 ist voll kompatibel zum alten 6502, in den meisten Fällen kann ein 6502 durch einen 65C02 ersetzt werden.

Tali–Forth–2 wurde von Scot W. Stevenson als Forth–System für einen 65C02–basierten Selbstbau–Computer, den *Übersquirrel* [1], entwickelt. Es ist der Nachfolger der ersten Version von Tali–Forth und einem Liara–Forth, dass für den großen Bruder des 65C02, den 65816 [2] gemacht war. In der Zwischenzeit wurde es von Thomas Woinke auf das *Steckschwein* (ein weiterer Selbstbaucomputer [3]) und von Sam Colwell auf den *SamCo's Single Board Computer [4]* und vom Autor auf den *Apple I* [5] (und Replicas) portiert. In der Tradition früherer Forth–Systeme ist Tali–Forth–2 „Public Domain", der Quellcode findet sich auf GitHub [6].

Tali–Forth–2 beinhaltet einen Assembler und Disassembler; beide benutzen *Simple Assembler Notation* (SAN) für den 6502/65c02/65816 [7]. Auch enthalten sind eine Implementation für Forth–Blocks im Speicher oder auf Speichermedien, ein traditioneller Block–Editor und ein Klon des Unix–Editors „ed" (für Forth unüblich, aber cool) sowie Wortlisten, eine Test–Suite und eine umfangreiche Dokumentation.

Tali–Forth–2 ist ein Subroutine–Threaded–Forth (STC) und in Assembler geschrieben. Um aus dem Quell–Assemblercode ein lauffähiges Forth–System zu erzeugen, wird der Ophis–Assembler [8] benutzt. Ophis ist in Python geschrieben, sodass Tali–Forth–2 auf einem PC übersetzt und dann auf das 6502–Zielsystem übertragen wird.

[1] 2019

Ist Ophis installiert und der Tali–Forth–2–Quellcode aus dem GitHub–Repository geladen, kann Tali–Forth–2 für eines der Zielsysteme mit dem Befehl

```
make taliforth-<platform>.bin
```

übersetzt werden, wobei `<platform>` durch den Namen der Zielplattform ersetzt wird. Die plattform–spezifischen Assembler–Quelltexte finden sich im Verzeichnis `platform`. Die Namen der dort befindlichen Dateien können als Zielsystem im Make–Befehl benutzt werden.

Tali–Forth–2 ist ein komplettes und modernes Forth–System für die 65xx–Prozessorfamilie. Das Forth wird aktiv entwickelt. So sind neue Zielplattformen wie das W65C134SXB–Board von WDC in Sicht.

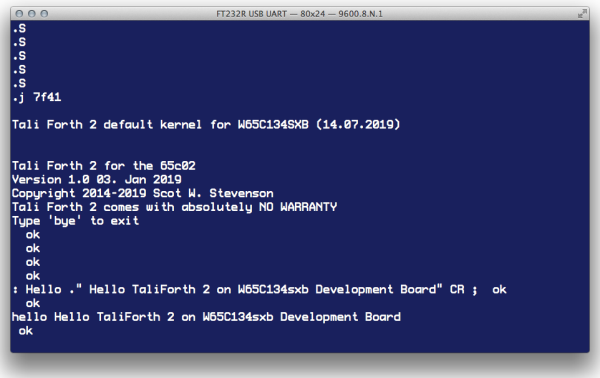


Abbildung 1: Screenshot des Tali–Forth–2

Links

[1] https://web.archive.org/web/20190608064330/http://uebersquirrel.blogspot.com/
[2] https://github.com/scotws/LiaraForth
[3] https://steckschwein.de/
[4] https://github.com/SamCoVT/SBC
[5] https://github.com/cstrotm/TaliForth2
[6] https://github.com/scotws/TaliForth2
[7] https://github.com/scotws/SAN
[8] https://michaelcmartin.github.io/Ophis/

show_registers

Martin Bitter

Oft kommt es beim Erkunden einer MCU vor, dass man Werte in Register schreibt oder ausliest. Und oft will man auch wissen, was alles in bestimmten Registern steht. Die schnellen Zugriffe, die ein Forth mit sich bringt, sind verschiedene Variationen von `@` *und* `!` *.*

Aber es kann mühsam sein, für viele Register jedesmal ein `@` und ein `.` einzugeben[1], ganz zu schweigen davon, dass die einfache Printausgabe von `.` nicht immer übersichtlich ist. Will man die Ausgabe in binärer Form sehen, hilft vielleicht ein Umschalten der Basis:

```
2 base !
```

Danach ausdrucken mit `.` und zurückschalten mit `decimal`. Wohl dem, dessen System über ein `.bin` oder `bin.` verfügt. Weggefallene führende Nullen muss man sich dabei selbst hinzu denken.

Etwas Selbstdisziplin beim Benamsen von Registerkonstanten und das im Folgenden gezeigte Wort `show_registers` kann dann eine große Hilfe sein.

Zuerst definiere ich ein Wort `.bin` , das einen Wert vom Stack nimmt und ihn in binärer Form und schön formatiert ausgibt. Da das Beispiel auf Mecrisp–Stellaris läuft, wird der jeweilige Wert als 32–Bit–Wert angezeigt. Hier muss man Anpassungen an „sein“ Zielsystem vornehmen.

Weiterhin benötige ich ein Wort, das erkennt, ob ein Wort, dessen Adresse auf dem Stack liegt, eine Konstante ist und ob sein Name einen vorgegebenen String enthält. Wenn beides zutrifft, soll der vollständige Name des Wortes und sein Inhalt formatiert in binärer und hexadezimaler Form ausgegeben werden.

Das mit dem übereinstimmenden Wortteil ist schon im Artikel *mwords* (Heft 2016-03) erklärt, bei `https://theforth.net/` findest du den kommentierten Quellcode dazu. Hier kommen die Worte `0_foldable?` und `.reg` hinzu. Das `.reg` wird an `travers_wordlist` übergeben — fertig.

Der Screenshot weiter unten zeigt so eine formatierte Information zu Registern.

Einschränkungen

Das Beispiel funktioniert nur, wenn die einzelnen Registeradressen tatsächlich als Konstanten gespeichert sind. Bei Adressierung mit Offset funktioniert dieser Ansatz nicht. Er lässt sich relativ leicht darauf anpassen. Das habe ich (noch) nicht gemacht, da ich es bisher nicht benötigt habe.

Links

`http://mecrisp.sourceforge.net/`

`https://theforth.net/packages`

Beispielhafter Srceenshot

```
show_registers porta
40010800 0100.0100.0100.0100-0100.0100.0100.0100 44444444 PORTA_CRl
4001080C 0000.0000.0000.0000-1010.0000.0000.0000 0000A000 PORTA_ODR
40010810 0000.0000.0000.0000-0000.0000.0000.0000 00000000 PORTA_BSRR
40010810 0000.0000.0000.0000-0000.0000.0000.0000 00000000 PORTA_BSRR ok.
show_registers portb
40010C00 0100.0100.0100.1000-0100.0100.0100.0100 44484444 PORTB_BASE
40010C00 0100.0100.0100.1000-0100.0100.0100.0100 44484444 PORTB_CRL
40010C08 0000.0000.0000.0000-1111.1111.0011.0011 0000FF33 PORTB_IDR
40010C0C 0000.0000.0000.0000-0000.0000.0001.0000 00000010 PORTB_ODR
40010C10 0000.0000.0000.0000-0000.0000.0000.0000 00000000 PORTB_BSRR ok.
```

[1] Mit dem Punkt ist der forth'sche „dot“ gemeint, die numerische Standard–Ausgaberoutine.

Listing

```
\ show_registers.frt
\ mecrisp header:
\ (addr), 4 byte link, 2 byte flags, 1 byte count,
\ n Bytes namestring (1 Byte align), code address


: traverse-wordlist ( xt -- )
  dictionarystart
  BEGIN
      over execute
      dictionarynext
    UNTIL
  2drop ;

: name. ( addr -- )
   dup 6 + ctype space
;

: (name.) ( addr -- addr )
  dup name.
;

: list ( -- )
  ['] name. traverse-wordlist
;

\ --- sophisticated example
\     how to use traverse-wordlist

\ ---- show_registers ( --- \ name )
\ shows all constants/variables
\ starting with pattern 'name'
: bin. ( u -- )  \ prints value u nicly formatted
  base @ swap 2 base !
  0
  <#  32 hold
     3 0 DO # # # # [char] . hold LOOP
     # # # #        [char] - hold
     3 0 DO # # # # [char] . hold LOOP
     # # # # #>
  type
  base ! ;

$20 buffer: findetext

: get_token ( -- / name ) \ parse inputstream,
                          \ save parsed text
  token                   \ parse
  >r                      \ save count
  findetext 1+ r@ move    \ move to buffer
  r> findetext c!         \ adjust count of buffer
;

: match_head? ( addr -- flag ) \ does sting at addr
                               \ match the sting
                               \ in findetext?
  7 + findetext count dup -rot
  compare
;

: 0_foldable? ( addr -- flag ) \ in mecrisp constants
                               \ are 0-foldable
  4 + h@ $40 =
;

: >code ( addr -- xt-addr )  \ skip to execution code
  6 + skipstring
;

: .reg ( addr -- addr' ) \ if word matches
                         \ print its content
  dup                 \ dup addr of word
  match_head?         \ does it match the searchsting
  IF
    dup 0_foldable?   \ is it a constant?
    IF
      dup             \ dup addr
      >code execute   \ execute word
      cr dup hex.     \ print addr of register
      @ dup bin. hex. \ print content of register
                      \ in bin and hex format
      name.           \ print name of register
      THEN
    THEN
;

: .(reg) ( addr1 addr -- addr1 addr' )
  dup                 \ dup addr of word
                      ( -- addr1 addr addr )
  match_head?         \ does it match the searchsting
                      ( -- addr1 addr flag )
  IF
    dup 0_foldable?   \ is it a constant?
                      ( -- addr1 addr flag )
    IF
      dup             \ dup addr
                      ( -- addr1 addr addr )
      >code execute   \ execute word
                      ( -- addr1 addr addr-n )
      2 pick +        \ add to base of register
                      ( -- addr1 addr addr+off )
      cr dup hex.     \ print addr of register
                      ( -- addr1 addr addr+off )
      @ dup bin. hex. \ print content of register
                      \ bin and hex
                      ( -- addr1 addr )
      dup name.       \ print name of register
                      ( -- addr1 addr )
      THEN
    THEN
;


: show_registers ( -- \ name ) \ parse string
  get_token
  ['] .reg
  traverse-wordlist \ show all matching values
;


: show(registers) ( base-reg -- \ name )
  get_token
  ['] .(reg)
  traverse-wordlist ; \ show all matching values

( finis)
```

Fingerübungen

Ulrich Hoffmann

Ein polyglottes Programm (polyglott = mehrsprachig) ist ein Computerprogramm, dessen Quelltext in mehr als einer Programmiersprache abläuft. Das Beispiel[1] *erzeugt bei der Ausführung in den verschiedenen Sprachen jeweils dieselbe Ausgabe. Das hat keinen offensichtichen Nutzen; es zu erstellen war eine unterhaltsame Übung für den Programmierer. So ein Forth–C–Polyglott könnte aber auch einen Forth–Kernel im C–Teil und die High–Level–Electives im Forth–Teil haben: Forth in einem File. Die Anwendungsgebiete sind vielfältig.*

EuroForth 2019

A Forth C Polyglot

```
1  char _= ( /*)drop
2
3  .( Welcome to EuroForth 2019 in Hamburg!) cr
4
5  bye */0);
6
7  #include <stdio.h>
8
9  int main(int argc, char* argv[]) {
10   printf("Welcome to EuroForth 2019 in Hamburg!\n");
11   return 0;
12 }
```

EuroForth-2019-Forth-C-polyglot.c

```
$ gcc -o EuroForth-2019-Forth-C-polyglot EuroForth-2019-Forth-C-polyglot.c
$ ./EuroForth-2019-Forth-C-polyglot
Welcome to EuroForth 2019 in Hamburg!

$ forth EuroForth-2019-Forth-C-polyglot.c
Welcome to EuroForth 2019 in Hamburg!
```

[1] Als 'forth' auf der Kommandozeile sollte ein jedes Forth gehen. Gforth geht auf jeden Fall.

Swap's Tagebuch

Matthias Koch

Es war eine tolle Überraschung für mich, als auf einmal die beiden Drachenboten und ehemaligen Drachenträger Manfred Mahlow und Klaus Zobawa an mein Küchenfenster klopften. Bei den beiden fühlte sich Swap einmal sehr wohl, so dass ich zur Begrüßung nur ruhig mit vier klaren Augen gemustert wurde. Doch gegen Abend, frei in seinem neuen Quartier, fing er an, unruhig herumzuflattern und alles anzuschauen. Für einen Forth–Drachen mit einer feinen Nase gibt es hier im Mecrisp–Hauptquartier natürlich überall etwas Interessantes zu erschnuppern . . .

Lange sollte es nicht dauern, bis Swap neue Freunde gefunden hat. Wobei ich mir nicht ganz sicher bin, ob Swap sich wirklich nur für Forth interessiert oder nicht vielleicht doch auch ein bisschen für Jungfrauen, die aus dem gleichen „Holz" geschnitzt sind wie er? Ob er dabei wohl lieber die Bronzenen mag oder die Silbernen? Hmmmm . . .

Abbildung 1: Swap mit gehaltvoller Lektüre und einer bronzenen Tänzerin.

Einen kleinen Flirt kann ich ihm ja nicht verübeln! Die nächsten Tage wurde es sehr ruhig in der Ecke der metallischen Wesen, nur manchmal vernahm ich ein leises Kichern, dass doch mehr verriet als ich hier schriftlich wiedergeben möchte. Zum Glück sind die Köpfe drangeblieben, ich kann die Leserinnen und Leser also beruhigen, niemand ist gefressen worden (Abb. 1).

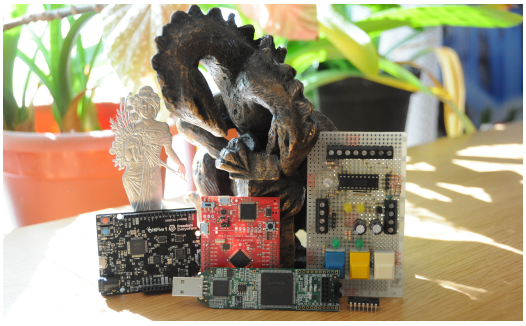


Abbildung 2: Swap inmitten von Forth-Geschichte: HiFive1 (Mecrisp–Quintus), Stellaris Launchpad (Mecrisp–Stellaris), Icestick (Mecrisp–Ice), meine Ausprobierplatine für den MSP430G2553 (Mecrisp) und schließlich, ganz klein in der Ecke: Ein MSP430F2012 (Mecrisp–Across).

Als es mir zu bunt wurde, rief ich Swap streng zur Ordnung: „Du bist hier, um mich zu neuen Forth–Entwicklungen zu inspirieren; nicht, um Tag und Nacht als schillernder Schmetterling durch die Gegend zu flattern!" Zwei beschuppte Köpfe blickten mich schuldbewusst an. Dann nickte er erst links und nach einer ganzen Weile auch rechts. Wir verstanden uns. Wenn ich meine Augen schloss, war Swap frei, aber während mein Blick auf ihm ruhte, gab er sich die größte Mühe, mir mit seiner Weisheit neue Ideen zu bescheren. Als Dank bereitete ich ihm ein Nest aus den vielen bereits unterstützten Portierungen, in das er sich gemütlich während des Dienstes einrollen konnte. Was könnte für Swap schöner sein als eine knusprige Sammlung der historisch wichtigen Chips, auf denen ich neue Architekturen erobert habe (Abb. 2)?

Unter den inspirierenden Drachenblicken wuchs und gedieh Mecrisp–Quintus prächtig, ein Assembler spross aus dem Boden, Interrupts kamen als willkommene Abwechslung dazu, viele kleine Fehler schmolzen wie Schneeflocken in der Sonne und kleine Verbesserungen blühten auf der weiten Wiese der Möglichkeiten.

Abbildung 3: Die Hüter Klaus, Matthias und Ulli mit Swap.

Doch dann, eines schönen Sommertages, sollte sich alles ändern. Es begann mit einer einfachen Frage: „Matthias, was sind Erdbeeren?", überlegte Swap, während seine Augen rollten, wie immer, wenn eine Idee zwischen seinen beiden Köpfen hin- und herpendelt. Ich muss wohl in dem Moment innerlich ganz woanders gewesen sein und erzählte ihm gedankenverloren etwas von Sammelnussfrüchten, die botanisch gesehen alles andere als eine Beere sind. Sein Blick daraufhin sagte mir wortlos, dass es nicht die Antwort gewesen ist, die er sich gewünscht hatte. So ging ich in den Garten, um Erdbeeren zu holen. Sein Vorgänger hätte ihm so oft von Erdbeeren vorgeschwärmt, meinte er, vorsichtig die ihm unbekannten Früchte kostend. Seitdem war er wie ausgewechselt. Erdbeeren! Allein der Gedanke daran lies ihn schon sehnsüchtig darauf warten, dass ich den Laptop zuklappen und mich auf den Weg in den Garten machen würde (Abb. 4).

Auch ehemalige Drachenträger kamen zu Besuch und so freute sich Swap gleich doppelt, als es Zeit für die Maker Faire wurde und er Klaus und auch Ulrich Hoffmann wiedersehen konnte (Abb. 3). Mit dem gefühlten Herbstanfang werden Swap und ich an regnerischen Samstagen wieder unsere drei Köpfe zusammenstecken … Doch was wir im Schilde führen, mag vorerst noch unser Geheimnis bleiben.

Abbildung 4: Jetzt weiß auch der bronzene Swap, wieso sein Vorgänger aus Plastik immer nach Erdbeeren duftete! Ob wohl bei mir ein Leckermäulchen aus ihm geworden ist?

Forth–Gruppen regional

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

München **Bernd Paysan**
Tel.: (0 89) – 41 15 46 53
bernd.paysan@gmx.de
Treffen: Jeden 4. Donnerstag im Monat um 19:00 in der Pizzeria La Capannina, Weitlstr. 142, 80995 München (Feldmochinger Anger).

Hamburg **Ulrich Hoffmann**
Tel.: (04103) – 80 48 41
uho@forth-ev.de
Treffen alle 1–2 Monate in loser Folge
Termine unter: http://forth-ev.de

Ruhrgebiet **Carsten Strotmann**
ruhrpott-forth@strotmann.de
Treffen alle 1–2 Monate im Unperfekthaus Essen
http://unperfekthaus.de.
Termine unter: https://meetup.com/de-DE/Essen-Forth-Meetup

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

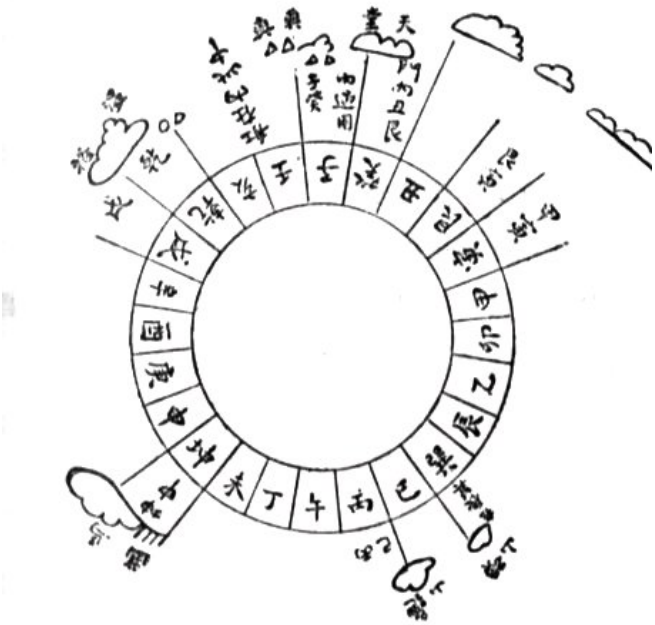
Forth-Hardware in VHDL microcore (uCore)	**Klaus Schleisiek** Tel.: **(0 75 45) – 94 97 59 3**$_p$ kschleisiek@freenet.de
KI, Object Oriented Forth, Sicherheitskritische Systeme	**Ulrich Hoffmann** Tel.: **(0 41 03) – 80 48 41** uho@forth-ev.de
Forth–Vertrieb volksFORTH ultraFORTH RTX / FG / Super8 KK–FORTH	Ingenieurbüro **Klaus Kohl–Schöpe** Tel.: **(0 82 66)–36 09 862**$_p$

Termine

Donnerstags ab 20:00 Uhr
Forth–Chat net2o forth@bernd mit dem Key keysearch kQusJ, voller Key:
kQusJzA;7*?t=uy@X}1GWr!+0qqp_Cn176t4(dQ*

27.–30.12.2019
36C3 in Leipzig
https://events.ccc.de/2019/10/04/36c3-in-leipzig/

Details zu den Terminen unter http://forth-ev.de



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

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

36C3 — 36. Chaos Communication Congress

27. bis 30. Dezember 2019 in Leipzig

> „Der Kongress beschäftigt sich in zahlreichen Vorträgen und Workshops mit Themen rund um Informationstechnologie, Computersicherheit, der Make–Szene, dem kritisch–schöpferischen Umgang mit Technik und deren Auswirkungen auf die Gesellschaft. An den sogenannten Assemblies kann an eigenen Projekten gebastelt, sich mit anderen ausgetauscht oder einfach nur gechillt werden.
>
> An den vier Kongress–Tagen zwischen Weihnachten und Neujahr kommen tausende Interessierte in Deutschland zusammen, um sich zuzuhören, auszutauschen, voneinander zu lernen und miteinander eine gute Zeit zu haben. ‚Der Congress' lebt durch die Vielfalt."[1]

Die offizielle Einladung ist auf der Club–Webseite veröffentlicht.

`https://events.ccc.de/2019/10/04/36c3-in-leipzig/`

Obacht: Da der Call for Participation aus Camp–Staub–Gründen erst spät begonnen hatte, endete die Einreichungsfrist für Vorträge diesmal sehr zügig, bereits am 26. Oktober 2019. Hoffentlich seid ihr dabei.

Abbildung 1: Die Rakete „Fairydust" ist das Logo der CCC-Veranstaltungen. Auf dem Bild sieht man die 7–Meter–hohe Rakete in der Leipziger Messehalle während des Chaos Communication Congress. Urheber: Yves Sorge, flickr, Creative Commons license.

[1] [Der Text „Chaos Communication Congress" wurde von www.kleiner-kalender.de entnommen und gekürzt.]