Vierte Dimension

2/2022

Das Forth–Magazin

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

In dieser Ausgabe:

Fancy Tendril (Schicke Ranke) FORTH BLÜHT AUF!
Ausschnitt einer alten Wollstickerei auf Leinen; Internet; bearbeitet (mk).

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 1030
48481 Neuenkirchen
E–Mail: Secretary@forth-ev.de
Direktorium@forth-ev.de
Bankverbindung: Postbank Hamburg
BLZ 200 100 20
Kto 563 211 208
IBAN: DE60 2001 0020 0563 2112 08
BIC: PBNKDEFF

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00 € + Porto u. Verpackung

Manuskripte und Rechte

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

Liebe Leser,

mit Freude weise ich auf die Rückseite unseres Heftes hin: Es gibt im Sommer ein *Forth BarCamp*. In bewusster Abwendung von traditionellen Organisationsformen ohne zuvor festgelegtes Thema und ohne Trennung zwischen Publikum und Vortragenden. Kommt einfach so vorbei.

Neben den Treffen in lokalen Gruppen gab es traditionell im Frühjahr die Jahrestagung der Forth–Gesellschaft (FG), als Zusammenkunft, mit Vortragenden. Dieses Format wurde ab 2020 notgedrungen zugunsten von Videokonferenzen verlassen, welche inzwischen doch erstaunlich gut und flächendeckend funktionieren, mit Teilnehmerzahlen durchaus in der Größenordnung der früheren Zusammenkünfte. Und auch die Vorstandswahl ist auf diesem Wege geglückt. Das Protokoll findet ihr im Heft. Und SWAPPY hat dabei einen neuen Hüter gefunden, Philip Zembrod, und wohnt inzwischen schon bei ihm.

Wolfgang Strauss' abendliche Videokonferenzen, immer montags, sind inzwischen besser besucht, als die Treffen in Essen es damals gewesen sind. Nicht erst anreisen zu müssen, um sich über Forth auszutauschen, hat was, muss ich zugeben. Dennoch, auf das BarCamp freue ich mich schon.

Manfred Mahlow führt uns weiter in die Welt der kleinen MCUs, mit eForth darin. Klein und dennoch komfortabel? *VIS* ist das Geheimnis dahinter: Es spendiert dem Forth–System einen zweiten Suchpfad. Neben dem klassischen statischen noch einen dynamischen dazu. So gehen selbst *data types* im sonst so spartanischen Forth.

Jörg Völker kümmert sich um mehr Programmierkomfort, Betriebssicherheit und Fehlertoleranz in Forth. Und setzt es um in ein schickes Forth von elaborierter Struktur, dennoch kaum größer als andere in den kleinen Targets, die er da verbaut im Modellbau. Es basiert auf Fließkommarechnungen, wie ihr im letzten Heft ja schon lesen konntet.

Rafael Deliano schwenkt beruflich mehr ein auf Elektronik und Mechanik, ein Maker eben. Also nichts, was Controller benötigt, eigentlich. Doch die Bauteile wollen getestet sein, und da kommt Forth in MCUs dann doch wieder zum Einsatz. Z. B. bei so Klassikern wie *R2R–Chips*.

Erich Wälde hat ja das AmForth–Erbe von Mattias Trute angetreten, der sich angeschickt hatte, es für weitere MCUs als die AVR–Linie nutzbar zu machen. Doch das übersteigt, was ein Mann in der Freizeit noch schaffen kann. Also alles auf Anfang: Das ursprüngliche AmForth 5.0 ist der Stand, auf dem Erich seine Sachen nun aufbaut. Und ohne Kabel inzwischen? „NIE IM LEBEN! Ein verdrilltes Paar einer Ethernet Cat6–Leitung ist unschlagbar. Wenn DA jemand dazwischen quasselt, dann bin ich's selber ;-) "

Last, but not least, hilft Willem Ouwerkerk uns aus der Patsche, falls seltsame Programmabstürze das Leben schwer machen.

Und nun angenehme Lektüre. Bis bald, euer 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-2022-02`

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

Ulrich Hoffmann Kontakt: Direktorium@Forth-ev.de
Bernd Paysan
Gerald Wodni

UXNTAL–Modus für Emacs

Für den *besten Editor*™(Emacs) gibt es nun einen Modus für die (UXN)TAL Programmiersprache. TAL ist eine an Forth angelehnte „Maschinensprache“ für die UXN virtuelle Maschine [1]. Seit Mai 2022 ist der UXNTAL–Modus direkt im Emacs mittels des eingebauten Paketmanagers aus dem MELPA–Repository installierbar [2]. cas

[1] `https://wiki.xxiivv.com/site/uxntal.html`
[2] `https://github.com/rafapaezbas/uxntal-mode`

JonasForth für UEFI[1]

Moderne Rechner mit AMD– oder Intel–CPU starten mit einer UEFI–Firmware. UEFI stellt im Flash–ROM Routinen für den Zugriff auf die Hardware bereit [1]. UEFI ist von der Funktionalität grob vergleichbar mit MS–DOS, jedoch im 64–Bit–Modus und mit Funktionen für Netzwerk–Zugriffe.

Es ist möglich, Programme direkt unter der Kontrolle des UEFI zu starten und auszuführen, ohne ein separates Betriebssystem (Windows, Linux, macOS ...) zu starten. Solche Software läuft direkt auf der Hardware („Bare Metal“) und hat ungehinderten Zugriff auf diese Hardware.

JonasForth [2] ist ein Forth–System, basierend auf JonesForth, welches direkt unter UEFI auf modernen PC–Systemen startet. Auf der GitHub–Webseite des Projektes gibt es Anleitungen, wie JonasForth innerhalb von virtuellen Maschinen (z. B. Qemu) oder auf echter Hardware gestartet werden kann (z. B. von einem USB–Stick, keine Änderungen auf der Festplatte notwendig). cas

[1] `https://github.com/safayetahmedatge/efitutorial`
[2] `https://github.com/c2d7fa/jonasforth`

Short Clip on Forth

> „There are lots of shorts on lots of languages on YouTube. But not one on Forth. So I thought: I'm quite capable of doing that myself. So I did. You can view it on YouTube: 2 Minutes with Forth. Hans Bezemer“

Und weil dort auf den RTX2010 von Harris[2] Bezug genommen wird, folgt gleich ein Rückblick, wie es zu dem Chip gekommen ist.

Und Anfang des Jahres[3] brachte Jürgen Pintaske in *comp.lang.forth* ein MISC[4] ins Gespräch, der ja so ähnlich ist[5]. Rafael Deliano ergänzte dazu:

> „Die Leistungsfähigkeit von CPUs in FPGAs ist nicht wirklich wesentlich, da sie typischerweise spezielle Peripherie ansteuern, wegen der ein FPGA überhaupt verwendet wird. Und auch der RTX war zwar via Standardzelle implementiert worden, hat aber eben speziell einen ASIC–Bus, um kundenspezifische Peripherie anzusteuern. Marcel Hendrix meinte, er sei damals enttäuscht gewesen, als er teure RTX–Evaluation–Boards von Harris kaufte, aber die Rechenleistung nicht so üppig war, wie angenommen. Das hat zwei Aspekte: Der Speicher war wohl externes EPROM, das ist langsam, unabhängig davon, wie schnell die CPU könnte. Zweitens fiel man leicht aufs Image rein. Harris war der Premium–Aerospace–Halbleiterhersteller der rad–hard und MIL–SPEC lieferte und zweiseitige Hochglanzanzeigen in Zeitschriften schaltete. Die No–Name–Firma Novix hingegen machte es mehr zufällig auf die Titelseite einer Zeitschrift, doch für Harris RTX war Titelseite Normalzustand. Das war also der Zenit von Forth, danach gings bergab.“

mka

1988: RTX2000

Am 28.10.1988 erschien in der Zeitschrift *Elektronik* in der Rubrik *Mikroprozessoren/Bauelemente* ein Artikel von Max Diez über den *RTX2000:* „Ein Echtzeit–Mikrokontroller“.

RTX stand für „Real Time Express“. *Harris Semiconductor* bewarb ihn als einen RISC–Mikrocontroller und „2000“ klang damals sehr futuristisch. Nun, im Jahre 2022, ist er im Museum zu bewundern. Diez schrieb damals:

> „Er schließt die bisher vorhandene Lücke, die Entwickler vorfanden, wenn sie nach einer extrem leistungsfähigen Architektur mit hohem Integrationsgrad suchten. Der Befehlssatz des RTX2000 ... entspricht direkt den Primitiv–Funktionen der Sprache FORTH. Software–Entwicklungszeiten für Echtzeitanwendungen schrumpfen damit von Monaten auf Wochen zusammen, womit kurze Markteinführungszeiten realistisch werden. Bei 10 MHz Taktfrequenz führt er nämlich im Schnitt 10 Mio. Befehle pro Sekunde (MIPS) aus. Spitzenwerte reichen sogar bis 40 MIPS. Im Hinblick auf die Komplexität des Befehlssatzes kann er es durchaus mit vielen CISC–Prozessoren aufnehmen. Mit einigen wenigen Bausteinen (es werden RAMs, PROMs, Taktoszillator und Chip–Select–Dekodierlogik zum Aufbau eines Minimalsystems

[1] Unified Extensible Firmware Interface (UEFI)

[2] Es gibt Harris Semiconductor nicht mehr und die RTX20x0 werden nicht mehr hergestellt. Es gibt Restbestände teuer bei `ebay.com` bzw. bei Brokern. (RD)

[3] Ein Idee Mitte der 90iger Jahre. Doch wenn man eine gewisse Mindestmenge an Befehlen unterschreitet, wird das Design nicht mehr kleiner, sondern nur noch langsamer. (BP)

[4] Minimal Instruction Set Computing

[5] Der RTX2000 ist Standardzelle, der MISC ist FPGA. Gemeinsam ist ihnen, dass Harris eine kleine CPU wollte, die über den ASIC–Bus I/O in kundenspezifischen ICs steuern konnte. Die sinnvolle Anwendung für CPUs in FPGAs ist identisch: I/O auf dem FPGA steuern. (RD)

benötigt) lässt sich bereits ein Hochleistungs–Mikrocomputer realisieren. Aufgrund der beim RTX2000 verwendeten CMOS–Technik (er benötigt im Standby lediglich 500 µA, im Betrieb 7 mA/MHz) und der geringen PCB–Fläche zum Aufbau eines Komplettsystems ist der RTX2000 bestens für den Einsatz in schnellen Steuerungen vor Ort (sogenannte ‚Embedded–Control–Anwendungen') geeignet . . .“

Integriert waren Timer, Multiplizierer und die Interruptsteuerung, sowie ein ASIC–Bus für die Peripheriebauteile — Kinder, wie die Zeit vergeht.

Rafael Deliano hatte das Elektronik–Heft noch und war so freundlich, den Beitrag zu scannen. So ist der nun im Forth–Archiv zu haben (8–Seiten–PDF mit technischen Details). Das Titelbild dieser deutschen Fachzeitschrift entstand anlässlich der Markteinführung des Chips auf der *Electronica* 1988 in München. Neuankündigungen von Produkten auf der ehedem weltgrößten Bauteilemesse war für US–Firmen recht üblich. Harris baute einen Teil des Messestandes als orangenen Schnellzug auf. Dort hielten die US–Vertriebler die Vorträge zum RTX, verteilten Prospekte, Buttons, Luftballons. mka

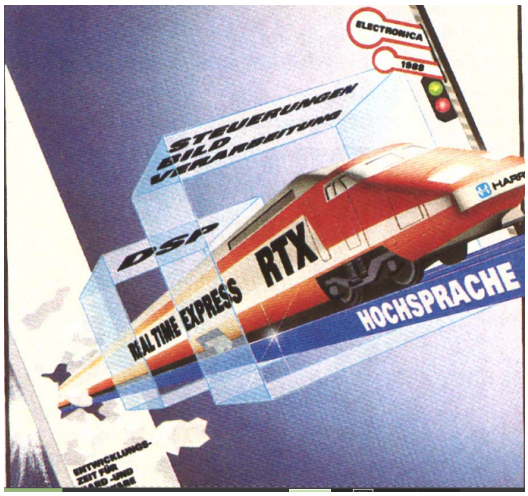


Abbildung 1: Titelbild der deutschen Fachzeitschrift „Elektronik“ 1988

Retrocomputer–Magazin LOAD

Neulich aus `de.alt.folklore.computer` gefischt von Rafael Deliano:

„Hallo in die Runde.

Hoffentlich ist mir keiner von euch böse, wenn ich diese Gruppe nutze, um das Erscheinen der neusten Ausgabe des Retrocomputer–Magazins LOAD aus dem *Verein zum Erhalt klassischer Computer e. V.* zu verkünden.

Die Themen könnten einige aus dieser Gruppe interessieren.

Titelthema: Geschichte der Workstations

Workstations
- Leere Uhrenchips umschiffen
- Netzkonfiguration bei Sun Solaris
- Iomega Laufwerke nutzen
- Editor vi benutzen
- UNIX für AMIGA
- Die HP PA RISC Story
- Sun Ultra1 neu installieren
- Linux auf Sun Hardware
- Atari Transputer Workstation

Hardware
- Texas Instruments Voyage 200
- Texas Instruments TI84plus
- Sinclair QNET entschlüsselt
- Der MicroPET
- Die Commodore Max Machine
- Commodore Max Machine wiederbeleben
- PC Emulatoren für Apple Macintosh

Software
- Atari ST CrossEntwicklungsumgebung
- Atari ST Cross Assembler
- Einstieg in dBASE.

Projekt
- SuperNOS im Eigenbau
- Temperaturfühler am Apple][
- Universal Cartridge für Commodore 64
- Commodore 1541 Laufwerk emulieren
- Das FluxcopyProjekt

Das Heft hat dieses Mal 84 Seiten (A4, professioneller Farbdruck). Wir geben das Heft gegen eine Spende ab, um die Kosten halbwegs zu decken.

Details findet ihr hier:

`https://www.classic-computing.org/load8/`

Gruß — Georg B.“

Abbildung 2: 2022: LOAD Ausgabe #8

Nachlese: cc64

Neuerdings hat Google entdeckt, dass ich mich für Videos über Forth interessiere und fördert immer wieder was zu Tage darüber. Erstaunlich, was es da so gibt. Einmal so etwas angeklickt und man bekommt mehr davon. Z. B. den Vortrag von Philip Zembrod. Da wird erklärt, wie ein *C–Compiler* geschrieben werden kann, in Forth natürlich:

> „I have recently joined Carsten Strotmann in the effort to renovate the VolksForth code base; my focus so far is the C64/C16 implementation. I had used VolksForth aka UltraForth ~25–30 years ago to implement a Small C compiler on the C64, targeting 6502 systems. Last year I have picked up that project again and open–sourced it [1] — this is what re–kindled my interest in Forth.
>
> In this talk I want to show some aspects of cc64, reasons for and advantages of writing it in Forth, some design details, a few challenges I faced, and the benefits of migrating to emulators."
>
> Quelle: YouTube: Philip Zembrod: cc64 — a Small C compiler written in Forth; Dec 29, 2020 mka
>
> [1] `https://github.com/pzembrod/cc64`

COMPILE AND EXECUTE

Auf der unermüdlichen Suche nach feinen Beiträgen für unser Magazin ergab die Korrespondenz mit Wolf Wejgaard neulich[6] dieses Thema: Vergleich einiger Interpretersprachen beim Verarbeiten einer Befehlszeile. Er hatte diesen Vergleich vor zwei Jahren in der Facebook–Gruppe „Forth/21–Century" publiziert und jüngst in der Nachbargruppe „Forth2020" erneut vorgestellt (ihr verfolgt diese Diskussionen auch?) und stellte es jetzt für das Magazin zur Verfügung:[7]

```
PYTHON:
for x in range(1,6): print(x,end=" ")
1 2 3 4 5

TCL
% set i 1; while {$i<6}
  {puts -nonewline "$i "; incr i}
% 1 2 3 4 5

PHP
with file test.php:
<?php $i = 1; while ($i < 6)
  {echo "$i "; $i++;} ?>
1 2 3 4 5

RUBY
for i in 1..5 do puts i end
1
2
3
4
5

GFORTH:
ok 6 1 DO I . LOOP
"Interpreting a compile-only word"
6 0 >>>DO<<< I . LOOP"

VFX FORTH:
ok 6 1 DO I . LOOP
"Err# 14 -14 ERR: Attempt to interpret
    a compile only definition"

volksFORTH-83 rev. 3.81.41:
ok 6 1 do i . loop DO compile only

TCLFORTH
ok 6 1 do i . loop
1 2 3 4 5 ok

FREEFORTH
ok 6 1 do i . loop
1 2 3 4 5 ok

volksFORTH-83 rev. 3.80a Z80 CP/M:
ok 6 1 do i . loop
1 2 3 4 5 ok
\ mit etwas Zauber von UH
```

Ihr seht an den Beispielen, dass GFORTH, VFX FORTH und volksFORTH–83 (z. B.) sich weigern, das zu machen, was in PYTHON, TCL, PHP und RUBY normal ist: Conditionals auf der Kommandozeile. Ist das bemerkenswert? Ich denke schon. Zu Zeiten, als diese anderen Sprachen entstanden[8], war das natürlich auch Thema bei Forth [1][2] und zieht sich durch bis heute, wie es scheint.

Also, was ist da gemeint? Ulrich Hoffmann teilte dazu mit:

> „Meiner Meinung nach geht es hier um den Unterschied zwischen klassischer Forth–Kommandozeile mit Interpret– und Compile–Zustand, wo ja in beiden Zuständen unterschiedliche Regeln gelten. Oder der Strategie, immer erst zu kompilieren und danach am Ende der Eingabezeile den kompilierten Code auszuführen und so eine interaktive Illusion zu produzieren. Letzteres bezeichnete Wolf als Compile & Execute. Peter Jakacki nannte es CompEx."

Aha. Und wo liegt nun eigentlich die Tücke, wenn man im Forthsystem solche Conditionals, wie Schleifen, „interpretativ" machen will? UH:

[6] Mai 2022

[7] Der Zeilenumbruch in den Beispielen ist der Tatsache geschuldet, dass es hier in eine Spaltenbreite eingepasst werden musste. Es sind aber Einzeiler.

[8] Python 1991, TCL 1988, PHP 1995, RUBY 1995

> „Klassisch kompilieren die Kontrollstrukturworte die benötigten Sprünge. Das machen sie, sobald sie ausgeführt werden, also auch, wenn interpretiert wird. Da zu der Zeit dann aber gar keine Definition entsteht, ist das sinnlos. Einige Forth–Systeme geben deswegen einen Fehler aus, wenn Kontrollstrukturworte interpretativ verwendet werden."

Nun sieht es aber so aus, als könnten es einige Forthsysteme doch. Ulrich selbst hatte es in das erwähnte VolksForth ja eingebaut. UH:

> „Die Idee dazu geht auf Mitch Bradley zurück [1]. Die CP/M–VolksForth–Kontrollstrukturworte müssen dabei die Schachtelungstiefe überwachen und interaktiv eine *temporäre Definition* beginnen. Beim Ende der Kontrollstruktur wird diese dann ausgeführt und anschließend wieder aus dem Weg geräumt. Das benutzt VolksForth–spezifische Möglichkeiten. Dabei wird die gesamte temporäre Definition auf den Return–Stack kopiert, dort ausgeführt und danach wieder vergessen. Nicht alle Systeme unterstützen diese Möglichkeiten und sie werden auch von den Standards nicht zugesichert. Der Aufwand in CP/M–VolksForth ist ca. 1 Screen. Aber Vorsicht: Innerhalb von Kontrollstrukturen gelten die Compile–Zustand–Regeln, weil ja in Wirklichkeit eine Definition kompiliert wird."

Das heißt, in diesem Ansatz muss *in* der Schleife `[char]` verwendet werden, damit ein `+----------+` ausgegeben wird:

```
CHAR + EMIT
10 0 DO [CHAR] - EMIT LOOP
CHAR + EMIT
```

Tja, da muss der Forthbenutzer *doch* wissen, was da passiert, sonst ist er hoffnungslos aufgeschmissen. Was wohl der Grund dafür ist, dass man es doch besser lässt, so zu tun, als könne man Schleifen einfach mal so interpretieren.

Gforth hat, um *conditional compiling* aus Quelldateien zu ermöglichen, einen anderen Weg genommen. Da gibt es sowas wie `[if] ... [else] ... [then]` , die das machen. Aber das wusstet ihr ja schon.

Ulrich war so freundlich, die maßgeblichen Quellen herauszusuchen, Bradleys grundlegende Arbeit und Hayes' Replik dazu.

[1] Bradley, Mitch. „Interpreting Control Structures — The Right Way", 1987 FORML ConferenceProceedings, pp. 126–130

[2] John R. Hayes „Interactive Control Structures", 1990 Forth Dimension, Volume XII, Number 2, pp. 28ff
`http://www.forth.org/fd/FD-V12N2.pdf`

Die Ideen von Hayes finden sich im VolksForth CP/M wieder. Diese beiden sehr aufschlussreichen PDFs findest du auch in unserem Forth–Wiki: `material-zu-4d2022-02.zip`

`https://wiki.forth-ev.de/doku.php/vd-archiv#jahrgaenge`

Viel Vergnügen bei der Lektüre! mka

Und als Zugabe bekommt ihr noch den oben erwähnten Screen. Hm, was passiert denn da? (Auflösung im nächsten Heft :)

```
35 list SOURCE.SCR
Scr 35
0 \ interpretive conditionals UH 25Jan88
1
2 | Create: remove>> r> rp! ;
3 | : >>r ( addr len -- addr ) r> over rp@ under swap - dup rp!
4 swap >r remove>> >r swap >r dup >r swap cmove r> ;
5
6 | Variable saved-dp 0 saved-dp !
7
8 | Variable level 0 level !
9
10 | : +level ( -- ) level @ IF 1 level +! exit THEN state @
?exit
11 1 level ! here saved-dp ! ] ;
12
13 | : -level ( -- ) state @ 0= Abort" unstructured"
14 level @ 0=exit -1 level +! level @ ?exit compile unnest
15 [compile] [ saved-dp @ here over dp ! over - >>r >r ;
```

Ja, wohin denn mit dem Forthprogramm, wenn's fertig ist?

Habt ihr euch auch schon mal gefragt, wie das Forth eigentlich von der HDD oder SSD[9] ins RAM kommt, neben all die anderen Anwendungen, die dort auch gerade laufen? Und wie das geht, das Forth mit dem, was man damit selbst hinzuprogrammiert hat, als Binary zu speichern und wieder zu laden? Und zu verbreiten? Ich hab nachgefragt. Und Matthias Koch hat für sein *Mecrisp–Stellaris* kurz erklärt, wie das gemacht worden ist, denn das läuft seit 2015 ja bekanntlich auch auf Laptops — unter Linux[10] und FreeBSD:

> „Im Prinzip ist eine Programmdatei für Linux ein Speicherabbild mit ein bisschen Verwaltungsinformationen zum Platzbedarf und zur Einsprungadresse. Das Format unter Linux heißt `ELF` und diese Datenstruktur, die Linux sagt, wo was hin soll, heißt `ELF header`. Normalerweise wird diese Struktur automatisch vom Assembler oder C–Compiler generiert; aber man kann das auch manuell tun, wenn man das denn möchte [1]. Darin ist diese Struktur einmal manuell definiert, so dass ich alle Daten darin manuell unter Kontrolle kriege. Damit kann Linux dann das Speicherabbild, welches sehr ähnlich wie das im Mikrocontroller ist, nur eben mit *Syscalls* statt Peripheriezugriffen

[9] Hard Disc Drive; Solid State Drive

[10] Du musst vorher das Paket `qemu-user-static` installieren, welches einen ARM–Emulator enthält, dann kannst Du es wie ein ganz normales Linux–Programm auf deinem Laptop starten. Am besten beginnst du mit `mecrisp-stellaris-2.6.3/linux-ra/terminal-with-tools`
Falls Du auf einem Raspberry Pi bist, brauchst Du keinen Emulator.

fürs Terminal, erkennen und laden. Wenn schließlich der Benutzer seine eigenen Definitionen kompiliert hat, wird in [2] die neue Größe im ELF header eingestellt und ein Speicherabbild in eine Datei geschrieben, also wirklich einfach nur den Speicherbereich mit dem angepassten ELF header rausschreiben.

Die Speicherverwaltung sorgt dafür, dass wir (wie jedes andere Programm unter Linux auch) einen eigenen Adressbereich zugewiesen bekommen, wir können uns also frei aussuchen, was für einen (virtuellen) Adressbereich wir haben wollen, solange er nicht insgesamt zu groß ist, so dass sich die Adressen bei nächsten Laden nicht verändern, auch, wenn das Forth ganz woanders im Arbeitsspeicher steht als letztes Mal.

Stimmt, dazu ist im Netz wenig zu finden, da nur noch ganz wenige Programme selbstmodifizierend sind, vor allem Viren, und eher eine Konfigurationsdatei verwenden würden als sich selbst neu zu schreiben. Dazu kommt, dass einige Betriebssysteme es aus Sicherheitsgründen (Pufferüberlaufangriffe) nicht erlauben, einen Speicherbereich, auf den Daten geschrieben werden, auszuführen.

Diesen ELF–Mechanismus hat Robert Clauseck für mich ausgetüftelt, von ihm sind die ursprünglichen Beispiele [3] und er hat mir sehr geholfen, das zu implementieren ... Liebe Grüße, Matthias“

Das Thema ist auf jeden Fall interessant. Bin gespannt, wie das in anderen Forth–Systemen gemacht wird — und was für Beiträge dazu kommen werden. :) mka

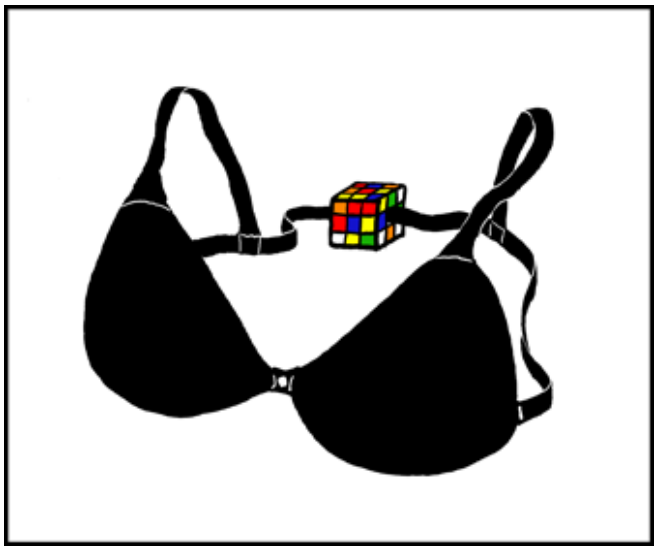
```
Quellenangaben:
[1] mecrisp-stellaris-2.6.3/ \
     mecrisp-stellaris-source/ \
     linux-ra/elfheader.s
[2] mecrisp-stellaris-2.6.3/ \
     linux-ra/save.txt
[3] mecrisp-stellaris-2.6.3/ \
     mecrisp-stellaris-source/ \
     linux-ra/selfreplicator...

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

Forth + Rust = Frust? Nein — Frustration!

„Frustration — Escaping a Turing Tar Pit with Forth“ [1]

Das Projekt *Fruststration* von Peter Fidelman (Universität Washington) implementiert eine virtuelle Forth–Maschine in *Rust*. Das ist nicht neu, in diesem Projekt dokumentiert der Autor jedoch detailliert den Aufbau der virtuellen CPU und das darauf aufbauende Forth–System. Sowohl der Rust–Code als auch die Forth–Quellen des Projektes sind sehr gut beschrieben. Ideal für den Forth–Programmierer, welcher ein wenig in die Programmiersprache Rust hineinschnuppern möchte, oder für Rust–Programmierer, welche die Magie von Forth in eigene Rust–Programme einbauen wollen. cas

[1] https://gitlab.cs.washington.edu/fidelp/frustration



[„Frustration“ : https://xkcd.com/457/]

DuskOS

DuskOS ist ein neues Projekt von Virgil Dupras, dem Autor von dem im Heft 4d2022–01 vorgestellten *CollapseOS*. Während CollapseOS auf kleinen 8– und 16–Bit–Maschinen läuft, wird DuskOS auf Basis eines 32–Bit–Forth implementiert und auf „großen“ Rechnern laufen. Ziel ist ein System mit grafischer Oberfläche und Dateisystem, welches z. B. auch PDF–Dateien anzeigen kann (keine kleine Aufgabe). Dabei wird wie beim kleinen Bruder CollapseOS großer Wert auf Einfachheit und Verständlichkeit des Systems gelegt.

In den ersten Entwicklungsschritten läuft DuskOS auf einem Linux–Kern, später soll es direkt ohne Linux auf der Hardware laufen.

DuskOS compiliert sich beim Start neu aus den Quelltexten. Teil des „Bootstrappings“ ist ein in Forth geschriebener C–Compiler, welcher neben Forth benutzt werden kann. DuskOS ist erst ein paar Tage alt, aber es wird fleißig daran gearbeitet. Die Projektseite und der Quellcode sind unter [1] zu finden. Unter [2] gibt es Informationen zur Mailingliste des Projektes. cas

[1] https://git.sr.ht/~vdupras/duskos
[2] https://lists.sr.ht/~vdupras/duskos-discuss

VIS HOWTO 03 — Data Types and Structures

Manfred Mahlow

In a Forth System almost all words are global and data typing is not supported. It's the responsability of the programmer to access data with adequate functions. This can be seen as an important handicap for design and implementation of applications and software components and also for programmer cooperation.

In view of this I developed the VIS extension during the last years. VIS stands for VOCs, ITEMs and STICKY Words. The basic idea of the VIS concept is to handle named wordlists in a less statical way by using vocabulary prefixes (VOCs) instead of F83 style vocabularies (VOCABULARYs).

A VIS extended Forth system has two search orders, a static one and a transient one. The static search order is the default one.

A VOC differs from a VOCABULARY by setting and activating the transient search order, not the static one. A transient search order then stays active until the next word from the input stream is interpreted or compiled. VOCs are immediate, never compiled.

The context switching semantics of a VOC can be assigned as an extra immediate semantics to any type of Forth word, making it a context switching item too. This can be used to implement data types, data structures and software components.

Simple Data Types

Defining a simple data type is a really simple task. Only two words from the VIS word set are needed, VOC and ITEM. Let's see how to define a data type for cell–sized integers.

Listing 1

```
1   forth definitions
2
3   voc int  int definitions
4
5     : @ ( a -- x ) forth @ ;
6     : ! ( x a -- ) forth ! ;
7     : ? ( a -- )   forth @ . ;
8
9     : item: ( "name" -- )
10    \ name ( -- a ; V: int )
11      item create 1 cells allot
12    ;
13
14    : field: ( "name" u1 -- u2 )
15    \ name ( a1 -- a2 ; V: int )
16      item 1 cells +field
17    ;
18
19  forth definitions
20
21  int item: I1    0 I1 !
22
23  \ or   int field: I1   in a data structure
24
```

We have to define

1. A VOCabulary for the type–specific words (Listing 1, line 3).
2. Functions to read, write and show the value of a data item (line 5 to 7). Here we can use the words from the VOCabulary `forth`.
3. A word to create an item of the new data type (line 9 to 12).
4. A word to create a field of the new data type in a data structure (line 14 to 17).

Done. Now we can create and use as many items of the new data type as needed. In line 21 one is created and initialized in the `forth` context.

After loading Listing 1[1] we can use the VIS dictionary browser `??` to check if the code works as expected.

`I1` and `int` are visible in the `forth` context:

```
forth words
I1  int  ...   ok
```

`I1` entered in interpret mode puts the data address on the stack and activates the `int` search order. Only the words defined in the VOCabulary `int` (and in `root`) are visible now:

```
I1 ??
 544 <sp
CONTEXT: int  root
CURRENT: forth
item:  ?  !  @   ok
```

Entering `?` displays the initial value of `I1`:

```
?  0 ok
```

After changing the value with the `!` function:

```
123 I1 !  ok
```

`@ .` and `?` return the correct value.

```
I1 @ .  I1 ?  123 123 ok
```

[1] `430eforth-g2553-43n7vis` is used here, see [4]

Data VOCabularies

What must be changed in Listing 1 to define another simple data type?

- The VOCabulary name.
- The functions for data access.
- The data size.

Obviously the code for the defining words can be unchanged, except the data size. If we define this as a constant `u/i`[2] inside the VOCabulary we can move the defining words `item:`[3] and `field:`[4] to the `root` VOCabulary.

Listing 1 is then stripped down to:

Listing 2

```
1   forth definitions
2
3   voc int  int definitions
4
5    1 cells constant u/i \ adress units per item
6
7    : @ ( a -- x ) forth @ ;
8    : ! ( x a -- ) forth ! ;
9    : ? ( a -- )   forth @ . ;
10
11  forth definitions
12
```

So, for a simple data type we only have to define the data size and the functions/words to access the data. That's all and that's what I call a Data VOCabulary.

Using simple data types makes programming much more relaxed, although the Forth data stack still knows nothing about data types:

- No hard to find errors caused by destructive write access.
- Write and read access are restricted to the data memory.
- Data write and read access are always type–conform.
- Simple data types are nice building blocks for data structures.

Data Structures

A data structure is an arrangement of data or locations for data, organized especially to match the (given) problem [1].

A never ending story. Forth comes with basic support for data structures, `+FIELD` for named data structures and `CREATE DOES>` or `<BUILDS DOES>` for what's left over.

[2] `u/i ( -- u ) \ address units per data item`

[3] `: item: ( "name" -- ) ?u/i item create allot ;`

[4] `: field: ( "name" u1 -- u2 ) ?u/i item +field ;`

Named Data Structures

Listing 3 shows a basic example how a named data structure can be defined with standard Forth words and how to create a data item.

Listing 3

```
1   forth definitions decimal
2
3   0
4     1 CELLS +FIELD p.x
5     1 CELLS +FIELD p.y
6   CONSTANT point
7
8   CREATE P1 point ALLOT  0 P1 p.x !  0 P1 p.y !
9
10  \ or   point +FIELD P1   in a data structure
11
```

All new words are global ones and the data is untyped:

```
forth words
P1  point  p.y  p.x  ...   ok
```

The programmer needs to know

- what structure a data item belongs to.
- what data fields are defined for it.
- what the valid data access functions are.

This can become quite a challenge if many data structures are defined.

Using a wrong field or function name may create a hard to find error that crops up later during program execution.

Defining named data structures with typed data fields inside a Data VOCabulary can avoid this:

Listing 4

```
1   forth definitions
2
3   voc point  point definitions
4
5   0
6     int field: x
7     int field: y
8   CONSTANT u/i
9
10  forth definitions
11
12  point item: P1  0 P1 x !  0 P1 y !
13
14  \ or   point field: P1   in a data structure
15
```

The structure and its fields are typed and the fields are local items in the Data VOCabulary.

Only `P1` and `point` are global words:

```
forth words
P1  point  ...  int  ...   ok
```

`P1` puts its data address on the stack and activates the `point` search order:

```
P1 ??  \ ?? is the Dictionary Browser
 544 <sp
CONTEXT: point  root
CURRENT: forth
u/i  y  x   ok
```

y from `point`, adds its address offset and activates the `int` search order:

```
y ??
 546 <sp
CONTEXT: int  root
CURRENT: forth
?  !  @  u/i   ok
```

`?` from `int` , displays the current value of `y` and deactivates the `int` search order:

```
? ??  0
 <sp
CONTEXT: forth  root
CURRENT: forth
P1  point  ...  int  ...   ok
```

The context switching of data items and data fields is immediate, not compiled. So a phrase like

```
P1 y ?
```

is compiled as

```
56450: P1
56452: point y
56454: int ?
```

Some extra words to access the data and error checking words[5] at begin and end of the data structure may be added:

Listing 5

```
1   forth definitions decimal
2
3   voc point  point definitions
4
5   struct begin
6     int field: x
7     int field: y
8   struct end
9
10   @voc first  \ or: @voc also
11   : @ ( a -- x y ) dup x @  swap y @ ;
12   : ! ( x y a -- ) dup >r  y !  r> x ! ;
13   : ? ( a -- ) dup x ?  y ? ;
14   forth first  \ or: previous
15
16  forth definitions
17
18  point item: P1  0 0 P1 !
19
20  \ or   point field: P1   in a data structure
21
```

The result is more than a named data structure and might be called a software component, „*the smallest set of interacting data structures and algorithms that share knowledge about how they collectively work*“[1].

More about data structures and software components in a next HOWTO.

Referenzen

[1] Leo Brodie, *Thinking FORTH*, `http://thinking-forth.sourceforge.net`

[2] Manfred Mahlow, *VIS HOWTO 01 — Vocabularies and Libraries*, Forth–Magazin Vierte Dimension 1/2020

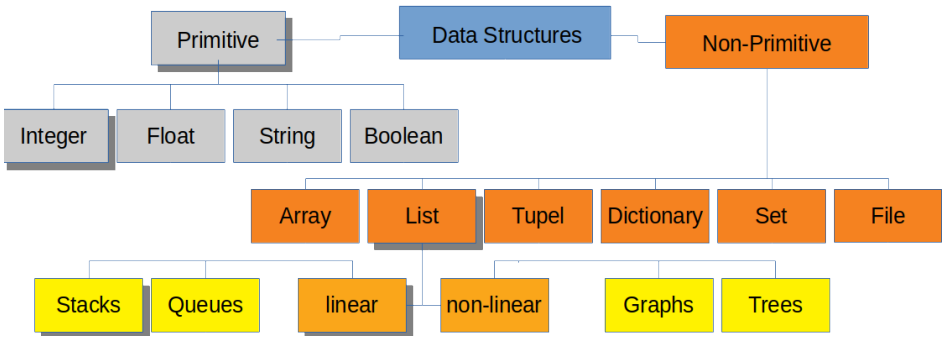
[3] Manfred Mahlow, *VIS HOWTO 02 — Creating Register Maps*, Forth–Magazin Vierte Dimension 2/2020

[4] 430eForth Workbench Project for the MSP430 MCU Family

`https://wiki.forth-ev.de/doku.php/en:projects:430eforth-workbench:start`

[5] Manfred Mahlow, *Namespaces and Context Switching for a Tiny Forth*, Forth–Magazin Vierte Dimension 4/2019

[6] Manfred Mahlow, *Namespaces and Context Switching for Mecrisp–Stellaris*, Forth–Magazin Vierte Dimension 1/2020



(Bildnachweis: Selbst erstellt. mk)

[5] `struct begin` and `struct end`
`struct end` creates the constant `u/i`

Projekt FancyForth: Tags auf dem Return–Stack

Jörg Völker

Nachdem mit dem FlexiFloat–Format auf dem Daten–Stack der Wortschatz für Arithmetik ordentlich entschlackt wurde, geht es heute dem Return–Stack an den Kragen. Entwicklungsziel: mehr Programmierkomfort, Betriebssicherheit und Fehlertoleranz.

Forth dürfte so ziemlich die einzige (?) Hochsprache sein, bei der der Programmierer direkten und ungeschützten Zugriff auf den Return–Stack hat und das eigenverantwortliche Arbeiten auf dem Return–Stack auch gar nicht vermeiden kann — denken wir nur an `DO LOOP` oder `>R R>` und solche Konstrukte. Das führt zu großer Flexibilität und ermöglicht teils abenteuerliche Programmiertricks, ist aber in der Alltags–Programmierung ein ständiges Risiko — der Absturz lauert überall. Und dabei erfolgt der Absturz in der Regel auch noch kommentarlos, was die Fehlersuche nicht gerade vereinfacht.

Tatsächlich ist das der Grund, warum ich bezüglich Forth zum Späteinsteiger wurde: Ich hatte zwar mehrere Versionen schon in meiner Homecomputer–Zeit, aber es blieb bei einem kurzen Kennenlernen. Das Nachladen von Kassette nach jedem Crash war mir viel zu nervig und langsam. So kam es zu gut 20 Jahren Abstinenz und erst mit *Holon11* und danach *Swift–X* zu neuer Zusammenarbeit.

Tags überall

Aber wie lassen sich Abstürze so weit wie möglich abfangen? Ganz einfach, indem man die Daten auf dem Return–Stack mit einem „Tag“[1], einer Markierung, versieht. Das ist grundsätzlich keine neue Idee, Tags für Adressdaten auf dem Datenstack z. B. gab es unter dem hochtrabenden Stichwort „Compiler Security“ schon im Fig–Forth. Da ging es darum, strukturelle Fehler schon zur Übersetzungszeit zu erkennen.

Hier nun sollen Tags zum Einsatz kommen, um Rücksprungadressen auf dem Return–Stack eindeutig zu kennzeichnen, damit `EXIT` nicht mehr auf andere Daten reinfällt und das System kommentarlos ins Nirwana verzweigt. Zwangsläufig müssen dann auch andere Elemente, die den Return–Stack belegen können, entsprechend gekennzeichnet werden. Zum Glück sind das in Summe gerade einmal drei:

1. Return–Adressen, von `:` („colon“) erzeugt, werden von `EXIT` verbraucht.
2. Daten, von `>R` („to–r“) erzeugt, werden von `R>` (r–from“) verbraucht.
3. Der Schleifenzähler `I` wird von `DO` erzeugt und von `LOOP` verbraucht.

Ich habe die Liste experimentell noch um eine *lokale Variable* erweitert, aber das ist unausgegoren und muss seine Nützlichkeit noch beweisen.

[1] „tag“ (engl.): Etikett, Schildchen, Marke, Anhänger

Auf einem 8–Bit–Prozessor kann der Tag also problemlos in einem Byte kodiert werden. Bei 16– und 32–Bit–Systemen ist es in der Regel nicht möglich oder sinnvoll, den Stack in 8–Bit–Schritten zu inkrementieren, da hat also so ein Tag schnell mal 16 oder gar 32 Bit. Solche Systeme haben aber auch immer mehr RAM zur Verfügung, so ist das vielleicht etwas unbefriedigend, aber eigentlich auch kein Problem.

Welche Primitives müssen nun neu kodiert werden? Die Liste umfasst alles, was auf den Return–Stack zugreift, siehe oben, also kommen noch `R@` und weitere `DO ... LOOP` Varianten dazu, sofern vorhanden.

An dieser Stelle stellt sich die Frage, wie man mit einem „Mismatch“ umgeht. Also z. B. was macht `EXIT`, wenn es auf keine gültige Return–Adresse trifft? Die naheliegendste Antwort ist natürlich: eine Fehlermeldung! Tatsächlich ist das aber nicht die einzige Möglichkeit. Und jetzt wird es interessant.

Neue Möglichkeiten

Mit den Tags auf dem Return–Stack kann die Funktionalität von Forth ein ganzes Stück erweitert werden. Der Trick besteht darin, dass z. B. `EXIT` nicht passiv das aktuelle Element auf dem Stack prüft und ggf. mit einer Fehlermeldung ablehnt, sondern aktiv nach einer gültigen Return–Adresse sucht. Wenn auch `I` und `LOOP` und `R>` sich so verhalten und sich ihre zugehörigen Daten auf dem Stack selber suchen, ergeben sich ganz neue Möglichkeiten und eine für Forth völlig untypische Fehlertoleranz. Einige gerade für Anfänger unverständliche Restriktionen fallen einfach weg. Bei dem Wort „Suchen“ bekommen natürlich alle Compiler–Bauer und Effizienz–Junkies sofort eine Krise, aber ich gehe vor dem Hintergrund massiv gestiegener Rechenleistung selbst im 8–Bit–Bereich immer mehr dazu über, Forth in Richtung Interpreter weiter zu denken und die Effizienz in Sachen Geschwindigkeit einfach mal hintenanzustellen.

Ein „intelligentes“ `EXIT`

`EXIT` kann nun auch die Rolle einer Müllabfuhr übernehmen und alles, was nach der Return–Adresse steht, automatisch vom Stack entfernen. Ein vergessenes `R> DROP` führt dann eben nicht mehr zum Absturz, und `EXIT` kann in einer beliebig tief verschachtelten Loop verwendet werden, `UNLOOP` wird überflüssig. Die gegenseitigen Abhängigkeiten von `>R` und `R>` sowie `DO ... I ... LOOP` und `EXIT` lösen sich weitgehend in Luft auf.

Auch sehr nützlich: Ein Stack–Trace

Die Tags erlauben es auch, nach einem Fehler den Return–Stack detailliert zu interpretieren und die Aufrufreihenfolge der beteiligten Worte sogar ggf. mit den Werten der beteiligten Schleifenzähler anzuzeigen — purer Luxus im Vergleich zu einem kommentarlosen Crash.

... und noch mehr

`R>` sucht in dieser Welt nur bis zur Return–Adresse, d. h. innerhalb des Forth–Wortes. Wird kein passender Wert auf dem Stack gefunden, erfolgt eine Fehlermeldung. Bei `I` kann man die Funktionalität noch erweitern: Lässt man es zu, dass ein Schleifenindex auch über Wortgrenzen gesucht wird, dann ist auch folgendes Konstrukt möglich:

```
: anzeigen ( -- )
  I . ;

: demo ( -- )
  10 0 DO anzeigen LOOP ;
```

Wie gesagt, es ist möglich. Ob so etwas auch sinnvoll ist, kann man natürlich diskutieren. Spannender wird diese Funktionalität, wenn man auf dem Return–Stack auch noch einen *Local Value* unterbringt, auf dem dann nachfolgende Unterprogramme ebenfalls zugreifen können. Quasi eine Alternative zur Parameter–Übergabe über den Stack.

FancyForth Fazit

All dies ist in meinem aktuellen Forth eingebaut, und das vorläufige Fazit sieht so aus:

- Zweifellos sinnvoll ist die Verwendung von Tags zur Vermeidung von Abstürzen, für Fehlermeldungen und Stacktraces.
- Die zusätzlichen Möglichkeiten in der Kombination von `>R R>` und `DO LOOP` usw. finde ich selber schick, aber widersprechen natürlich dem Standard. Hier gefällt mir vor allem, dass auch der eine oder andere meiner Flüchtigkeitsfehler abgefangen wird.
- Der Zugriff auf Schleifenzähler über Wortgrenzen hinweg war einfach zu ergänzen und hat erst einmal mehr experimentellen Charakter. Das gilt auch für den Local Value.
- Auch positiv: Es werden keine grundsätzlich neuen Worte eingeführt, sondern nur die Funktionalität bestehender Worte erweitert. Mit einer Ausnahme: Im Compiler ist dann doch der direkte Zugriff auf Return–Adressen nötig, dazu habe ich `RTAG>` und `RCELL>` neu eingeführt.
- Solange nur Fehler angezeigt werden, ist der Aufwand nicht so groß. Nimmt man die Suche und das intelligente `EXIT` dazu, ist dagegen schon mehr zu programmieren. Die derzeitige 32–Bit–Version mit Local Value auf dem *STM8* hat ca. 500 Bytes.

Wie bei mir (leider) üblich, habe ich hier keine Forth–Quellen, sondern die Testimplementierung erfolgte direkt in den Assembler–Primitives. Sorry :-)

Forth Sequenz	Klassisch	Fancy
>R ;	Absturz	OK, funktioniert
R> ;	Absturz	Fehlermeldung
DO EXIT LOOP	Absturz	OK, funktioniert
>R DO R@ LOOP	Fehlfunktion	OK, funktioniert
DO >R I R> LOOP	Fehlfunktion	OK, funktioniert
DO >R LOOP	Absturz	Fehlermeldung
DO R> LOOP	Absturz	Fehlermeldung
: WORD I ; DO WORD LOOP	Fehlfunktion	OK, funktioniert (!)

Abbildung 1: Ein Vergleich mit einer klassischen Implementierung.

Tester für R2R–Netzwerke

Rafael Deliano

R2R–Widerstandsnetzwerke (Abb. 1)[1] *sind als 8–Bit–D/A–Wandler nützlich, weil schaltungstechnisch einfach verwendbar und billig. Sie sind allerdings bezüglich Genauigkeit eher vage spezifiziert. Für die Anwendung in Prüfgeräten ist es daher nötig, sie zu testen.*

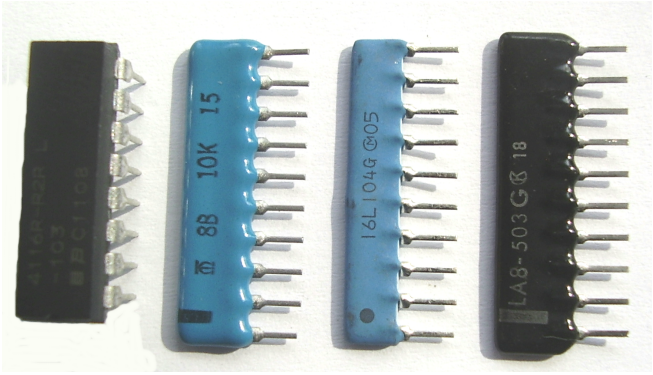


Abbildung 1: R2R–Netzwerke (von li. nach re.: a,b,c,d)

	Form	Name	Typ	4R=
a	DIL	Bourns	4116R–R2R–103 L	40K
b	SIL	Murata	6L104G 05	200K
c	SIL	blau	—	20K
d	SIL	schwarz	—	100K

Tabelle 1: Getestete Muster

Als Test ist es üblich, alle Codes auszugeben und die resultierende Rampe mit höher auflösendem A/D–Wandler zu messen (Abb. 2).

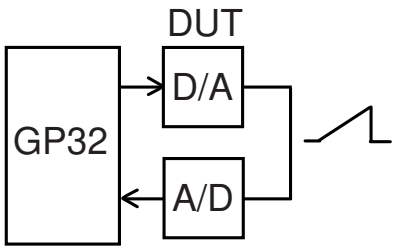


Abbildung 2: Test D/A–Wandler

Array

Als 10–Pin–SIL oder 14–Pin–DIP handelsüblich. Das Pinout, zumindest bei SIL, ist bei den Herstellern einheitlich (Abb. 3). Bei DIL vereinfachen die zusätzlichen Kontakte die Verwendung als z. B. 6–Bit–D/A und eventuell auch den Abgleich per Laser.

In der Definition des Widerstands sind sich die Anbieter nicht einig. Es empfiehlt sich Messung zwischen D0 und GND, d. h. 4R (Tab. 1). Das so definierte R entspricht dann dem Ausgangswiderstand (Abb. 4). Der Innenwiderstand R lag hier abhängig vom Typ bei 5k . . . 50k (Tab. 1). Sie sollten ausreichend hochohmig sein, sonst wirkt sich der Innenwiderstand der FETs im Port des Controllers auch als Fehlerquelle aus.

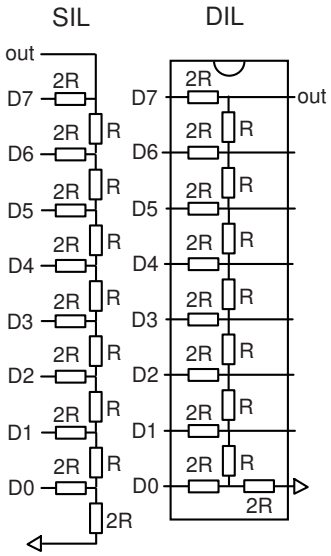


Abbildung 3: Innenbeschaltung

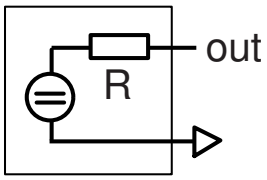


Abbildung 4: Innenwiderstand

Schaltung

Hier wurde als A/D–Wandler ein 24 Bit *LTC2400* verwendet. Es gibt ihn nicht als DIP, der SO8 muss mit den Kondensatoren auf einen kleinen Adapter verbaut werden (Abb. 5). Erste Tests ergaben, dass er bei hochohmigen Arrays Messfehler erzeugt. Die Schaltung wurde deshalb um einen OP ergänzt (Abb. 6). Dessen negative –0,43 V Versorgung erzeugen Optokoppler, die als „Solarzellen“ beschaltet sind. Anders als Schaltregler liefern sie saubere Spannung. Sie sorgen auch für üppigen Strom auf 5 V. Der ist nötig, damit man die positive +5,6 V Versorgung per Spannungsteiler ausreichend niederohmig liefern kann, ohne dadurch die 5 V unbeabsichtigt hochzuziehen.

[1] R/2R Series Thick Film Networks

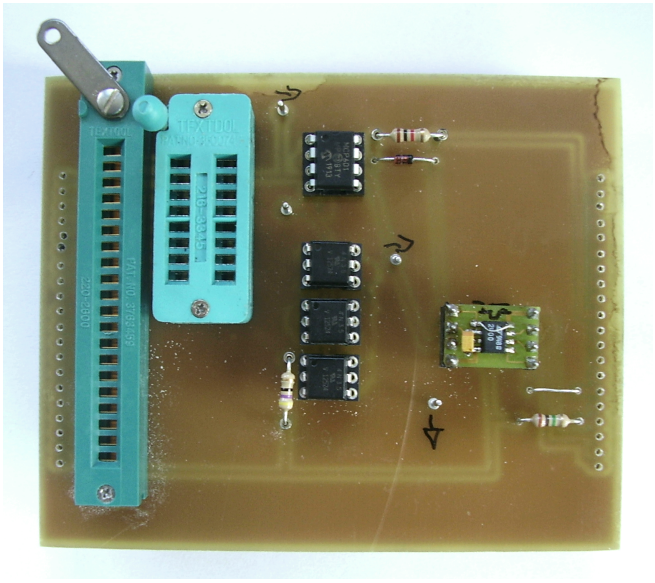


Abbildung 5: Breadboard

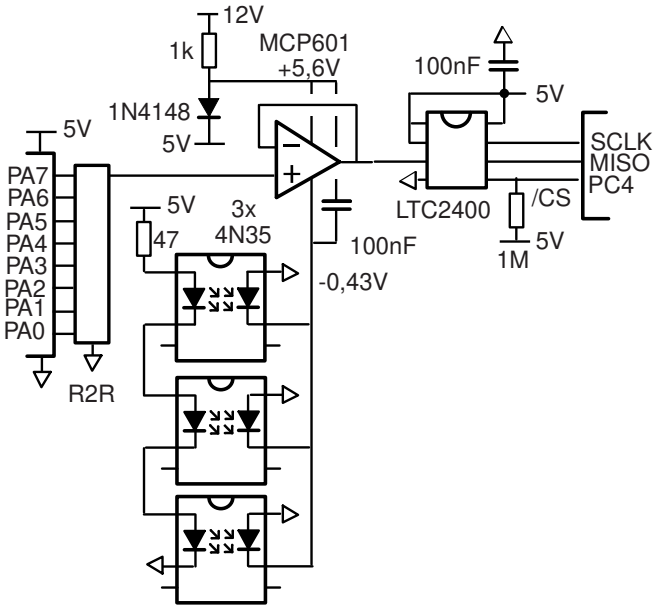


Abbildung 6: Schaltung

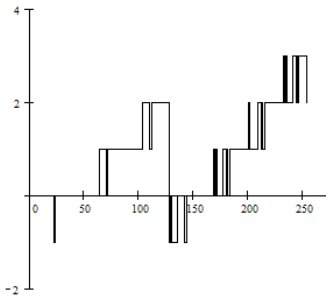


Abbildung 7: Fehlerplot a

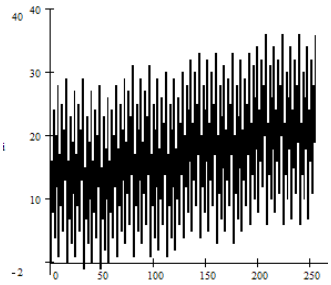


Abbildung 8: Fehlerplot b

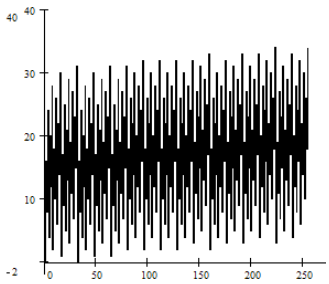


Abbildung 9: Fehlerplot c

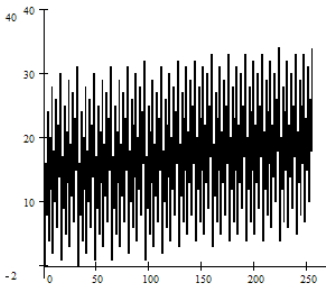


Abbildung 10: Fehlerplot d

Messungen

Der *Fehlerplot* ist die Differenz zwischen D/A und A/D (Abb. 7 bis 10). Hier um 4 Bit auf 12 Bit skaliert, d. h. 1 = 1/16 LSB. Alle SILs bleiben ca. in ±16 d. h. ±1 LSB. Erfreulicher wäre $\pm 0,5\,LSB$ gewesen. Aber die Datenblätter sprechen nur von 2 %. Der DIP wurde offensichtlich mit Lasertrimmer bearbeitet und ist erstaunlich gut (Abb. 7). Tempco wird von Murata mit $\pm 200\,ppm/°C$ angegeben. Dickfilm erreicht nicht die Werte guter ICs.

Speedup

Der A/D–Wandler ist zwar sehr genau, aber mit ca. 165 msec langsam. 265 Messungen dauern 44 sec. Um tütenweise Bauteile zu testen, muss der Ablauf schneller sein. Dafür sind ohnehin nur die Spitzenwerte des Fehlers von Interesse. Ein simpler Ansatz war hier, nur zwei kleine Abschnitte am Anfang und Ende zu messen (Abb. 11), da dort vermutlich die Extremwerte liegen. Das erwies sich nur als näherungsweise richtig, war aber gut genug. Als

weitere Vereinfachung erhielt der Fehler einen Offset von 1000, damit man immer mit vorzeichenlosen positiven 16–Bit–Zahlen arbeiten kann. Die Fehlerverteilung war keine Gaußkurve, sondern fast alle Widerstands–Arrays verhielten sich identisch, außer den Ausreißern, die man nicht verwenden will und wegen denen man testet (Abb. 12). Diese Abbildung zeigt in der oberen roten Linie den maximalen Fehler eines Bauteils, in der unteren grünen Linie den minimalen Fehler. Auf der X–Achse ist die laufende Bauteilnummer aufgetragen, in dem Bild wurden somit die Tests von 80 Arrays vom Typ c (s. Tab. 1) aufgezeichnet. Vier Teile konnten ausgesondert werden wegen zu großer Ungenauigkeit ihrer Widerstände.

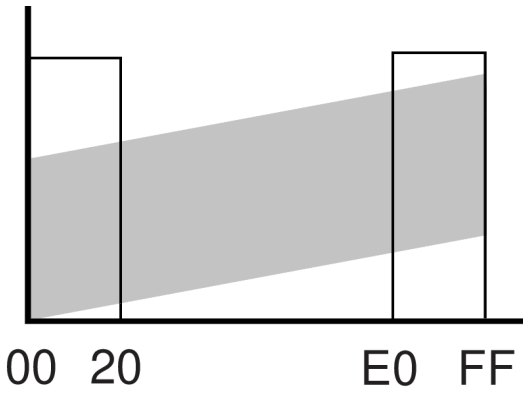


Abbildung 11: Fenster

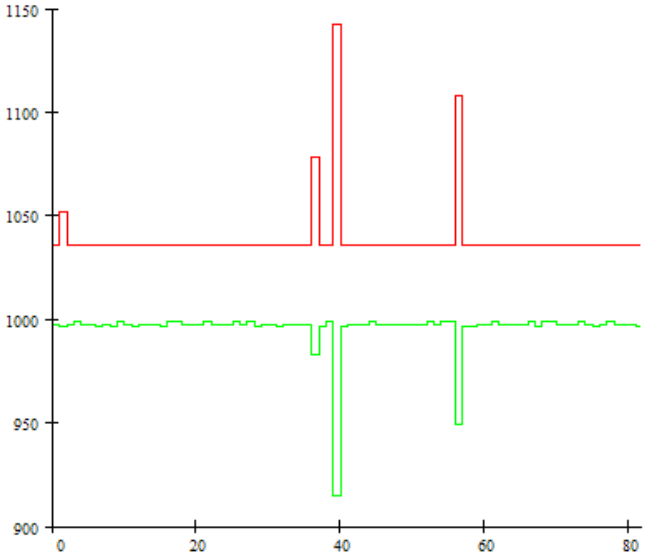


Abbildung 12: Maximale und minimale Fehler von 80 Arrays des Typs c.

Links

https://de.wikipedia.org/wiki/R2R-Netzwerk

Anmerkung des Redakteurs: Neugierig geworden, wo so etwas wie R2Rs denn tatsächlich verbaut werden, stieß ich auf *teure Audioanwendungen* und kam aus dem Staunen gar nicht mehr heraus. Da werden tatsächlich jede Menge hochwertige DACs diskret aufgebaut.

> „Discrete means a ton of tiny resistors all individually mapped out with specific engineering requested processing parameters to convert the digital signal to analog in the exact way Musician Audio wants it to be done. It is more time consuming than delta–sigma but the offshoot is generally always a more natural sound that is free of artifacts or ringing, especially in the higher frequencies. [Musician Audio Pegasus R2R DAC Review, 2021]“

Schauts euch an, wer hätte das gedacht? Ich bin begeistert! mk

Listing

```
<| \ LTC2400

\ INIT             \ init IO-Pins
\ INIT-SPI         \ init SPI for A/D

\ (SPI-C!@)        \ in:  akku = data
\                  \ out: akku = data

\ ----------------------------------------
\ LTC2400 24 Bit dual A/D

\ 76543210 ISTAT
\ ^         /EOC   0
\  ^        0
\   ^       SIGN   1 = +
\    ^      EXR    1 = overrange

\ read only: MISO not connected

1 ZVARIABLE ISTAT

:CODE A/D@                    \ (  --- UD1 )
              /CS-A/D MBC,
       DEX, DEX, DEX, DEX,
' (SPI-C!@)           JSR,
                ISTAT STA,    \ store flags
                0F #. AND,    \ mask flags
                  0,X STA,    \ MSB first
' (SPI-C!@)           JSR,
                 1 ,X STA,
' (SPI-C!@)           JSR,
                 2 ,X STA,
' (SPI-C!@)           JSR,
                 3 ,X STA,     \ LSB last
              /CS-A/D MBS,
                      RTS,
CODE;

: TEST-A/D    \ (  ---  )
  INIT INIT-SPI
  BEGIN
    CR A/D@ NH. NH.  \ print 32 bit
  AGAIN ;
|>

\ nanoFORTH auf MC68HC908GP32
```

Simplify your AmForth–Life

Erich Wälde

Since the AmForth project has lost its original author and now, that I have accepted the fact, that I cannot really live up to the role as the maintainer of this project, I have thought about possible ways for me to continue at least with my own homebrew data acquisition system. This text covers a number of decisions I made to simplify my AmForth–Life.

Introduction

I have been running a data acquisition system based on AVR microcontrollers equipped with AmForth, RS485 transceivers, sensors and whatever is needed: A multitasker and multi–processor communication and a little bit of magic glue, this works happily with AmForth Version 4.6 since 2011.

In 2020 I decided to port the setup to the latest version of AmForth 6.8, and to add a few changes to make the communication more robust. For reasons still unknown I never managed to reach a stable state. The communication on the RS485 cable would get stuck after some time for no apparent reason. By the end of the summer and after many hours I simply gave up.

However, a tinkerers mind will not rest, so occasionally I would still try to think of why, and how to find out. And recently I picked up this project again with a rough plan on how to proceed.

The mere fact, that the old version of the program works fairly reliably[1] inspired the question, what had changed in AmForth since then. Lots of things of course, but there is a fundamental change: AmForth gained support for MSP430 in version 5.6, and support for ARM and RISC–V32 later. I remember that I had to include a lot more Forth code files than before, where close to everything existed as precompiled .asm files. And I remember discussions with Matthias, about how to proceed in detail, like e.g. the MSP430 lacking proper EEPROM. So it occured to me that this fundamental rearrangement of puzzle pieces might have created undesired side effects somewhere. So one option forward would be to stay with version 4.6 or at least with a version before 5.6 for possibly unjustified simplicity.

Another thing that always has bothered me a little, was the dependence on `wine/avrasm2.exe`, which I never liked. So I went back to my old directories to see, up to which version I could assemble AmForth using `avra` instead. And while collecting details, I found that `avra` [1] has seen new releases in 2019/2020, including support for my beloved *ATmega644P*. So I gave it a new try. I could assemble version 5.0. It failed on 5.1 for the use of the `.DD` directive. I had a glimpse at the source code, I found the place, where `.DW` was handled, but it did not look like a half hour adventure to add `.DD`. But anyway, using AmForth 5.0 together with `avra` became a nice option forward!

[1] Not flawless, mind you!

Another thing that I did not really like were the Python tools associated with AmForth. Sorry folks, but I just don't like Python. I don't use `amforth-shell.py`, because it does things behind my neck without really telling me. I used an old version of `amforth-upload.py`, because of some niggle I had with the current version. I do not remember, what it was. I do remember that I used the current version when documenting the clockworks stuff. And I did not like it. The old version is on Python 2 and I'm in no position to port this to Python 3. But I am old enough to have learned *Perl* (version 4 and 5!). And just to find out, if I could, I wrote a slim, stupid `upload_fs` script. It is a bit slower than the Python thing, it ditches a character in the screen output here and there, but it works. And it blends together well with my other Perl scripts `unfold_fs` (to recursively resolve `#include` directives) and `trim_fs` (to remove comments and some white space for smaller upload size). Call me a Perl die hard, that's ok.

I made two more decisions upfront: First, I decided to write any interrupt service routines (ISR) in assembly. I did have my share of problems with Forth ISRs, the most important one was the uncovering of a race condition, 1 bit by 1 instruction cycle small, the details of which are given at [2]. The fix was released with version 6.5 in 2017. And second, every step towards a working sensor station would have to run at least 10^6 seconds (12 days) straight on the cable with the data collector contacting it every 2 minutes — no failures allowed. If this cannot be achieved, ditch the whole project pretty much immediately.

Work Environment

So I set out along this path. I had converted the Subversion repository into a Git one some time ago ([3], [4]). So I made a clone of the latter, checked out the branch `releases/5.0` and created a new branch from this point called *am50forth*. This branch shall accumulate any changes, picked from later releases and the like, in order to make my data acquisition system work. Nothing else. This is not going to be a full blown project. However, I created an *offsite backup* of sorts for my projects at `sourcehut.org`. So you will find my fork of AmForth 5.0 here [5]. This includes the aforementioned Perl scripts. You will find the new firmware at [6] alongside the design files of the printed circuit boards, that I use. Feel free to roam all of this for your inspiration.

Then I added a copy of the avra project mentioned before and compiled and installed `avra` :

```
$ bin/avra --version
AVRA: advanced AVR macro assembler \
 (version 1.4.2)
```

At this point I decided to put the newfangled *RISC–V64* machine I happen to have into action for this work environment, just in order to increase the chance for obscure bugs — I will hate it at some point, I'm sure. But so far everything works.

To assemble AmForth, a bunch of definition files are needed. They come with `avrasm2.exe`, of course and were copied into a separate directory.

Next I created a directory tree meant to collect all the files, which comprise the new firmware for my ATmega644P controllers. I included the mentioned Perl scripts and my `makefile` driven workflow came to life.

`make` will assemble the AmForth firmware and produce two `.hex` files as usual. `make install` will feed them to the burner and thus onto the connected controller. `make marker` will then upload a fixed set of Forth files that I always want. And `make upload` will upload the project code. There are also `make` targets to set the fuses or read the content of the controller back to files. That is how I like it.

Experiment 1 — an empty sensor node

The ATmega644P controller is driven by a 11.0592 MHz crystal, which allows a 115200 baud serial interface. A second crystal of 32768 Hz drives the clock ticks.

The first iteration of the new firmware will include the following features:

- A clock tick twice per second.
- The multitasker would run all the work in the background.
- The background job counts up on time and calls periodic jobs.
- The foreground job is the well known command loop.
- The uptime count serves as sensor value to be reported.
- Data are reported on request only.
- Data are send as a plain ASCII text string, including a *fletcher16* checksum for every record.

This needs some tricks around `emit`, but I would not consider them clever. At the start of a record the variables to calculate the fletcher16 checksum are reset. Then `emit` is replaced by a wrapper, which will update the variables and then emit the character nonetheless. At the end of the record the checksum is finalized, formatted, and emitted.

The controller is connected with a USB to serial TTL cable. A small system (*Olinuxino Lime2*) is running a collector script to request data reports every 2 minutes. This is the same code that runs the existing data collection.

This version ran 12 days on first attempt. So this was encouraging.

Experiment 2 — add handling of a RS485 transceiver

This modification involves changes to the low level communication code. When sending a byte off the controller, be sure to assert the write pin on the attached RS485 transceiver. Also be sure to release this pin once the byte was sent. The *transfer complete interrupt* will handle this. As mentioned, all these changes are made in assembly, not in Forth.

This version ran 6 days, then I spoiled it while messing around on my desk. The second attempt succeeded and has passed 1.3 million seconds while I write this. So far everything looks good.

Experiment 3 — add multi processor communication

This change is in progress. It will allow several nodes on the cable connection. The controller acts deaf until a control byte is received. If this happens to contain that controller's station ID, the firmware will respond to the following 7 bit communication.

Conclusions

The Forth code mentioned here is not new. I already wrote about this in our Forth Magazin *"Vierte Dimension"*, but in German.

Simplifying the code base may prove useful to advance my home–brew stuff. The experiment continues.

References

[1] `https://github.com/Ro5bert/avra/`

[2] The 1 Bit by 1 Instruction Cycle race condition — Forth Tagung 2017, Kalkar
`https://forth-ev.de/wiki/events:tagung-2017`
(Talk about this in German)
VD 2017/3 :: Clock Works 3 (E.Wälde) — Auf der Suche nach der verlorenen Zeit
VD 2017/4 :: Clock Works 4 (E.Wälde) — Des Rätsels Lösung : Fixed in AmForth releases 6.5 and 6.6

[3] `https://git.sr.ht/~amforth/code-tree/`

[4] `https://git.sr.ht/~amforth/code/`

[5] `https://git.sr.ht/~ew/hbv3_am50forth` personal fork of AmForth 5.0

[6] `https://git.sr.ht/~ew/hausbus-v3` personal repository with the code for the new firmware.

Forth–Gesellschaft e.V. — Ordentliche Mitgliederversammlung 08.05.2022

Wolfgang Strauß

Versammlungsleiter: Anton Ertl

Protokollant: Wolfgang Strauß

Teilnehmer:
Direktorium: Ulrich Hoffmann, Bernd Paysan, Gerald Wodni
Insgesamt sind 13 stimmberechtigte Mitglieder in der Versammlung.

Sitzungsdatum: Sonntag, 08.05.2022

Sitzungsbeginn: 9:05 Uhr

Sitzungsende: 11:40 Uhr

Sitzungsort: pandemiebedingt online per Videokonferenz

Begrüßung der anwesenden Mitglieder

Gerald Wodni begrüßt im Namen des Direktoriums die anwesenden Mitglieder.

Wahl des Schriftführers

Wolfgang Strauß wird zum Schriftführer gewählt.

Wahl des Versammlungsleiters

Anton Ertl wird zum Versammlungsleiter gewählt. Er stellt fest, dass die Versammlung fristgerecht einberufen wurde. Der Verein hat momentan 90 Mitglieder, von denen 13 anwesend sind. Damit ist die Versammlung beschlussfähig (mehr als 10 Prozent der Mitglieder sind anwesend).

Ergänzungen zur Tagesordnung

Es gibt keine Ergänzungen zur Tagesordnung.

Bericht des Direktoriums

Bericht der Verwaltung (Carsten Strotmann)

Mitgliederentwicklung

In 2022 haben bis dato drei Mitglieder den Verein verlassen (zwei davon durch Ableben), ein Mitglied konnte neu gewonnen werden.

Momentan hat der Verein 90 Mitglieder. Der Aufnahmeantrag eines weiteren Interessenten liegt vor.

Finanzen

Carsten Strotmann hat am 01.01.2022 die Verwaltung (Forth–Büro) übernommen. Da der Amtsinhaber während des Geschäftsjahres 2021, Ewald Rieger, aus persönlichen Gründen nicht in der Versammlung sein kann, erläutert Carsten Strotmann im Detail die Einnahmen und Ausgaben, getrennt nach Verein und Zweckbetrieb (Vereinszeitschrift Vierte Dimension). Zum Ende des Wirtschaftsjahres 2021 ergibt sich ein Vermögen von 8.984,07 EUR, das ist ein Zuwachs von 1.139,55 EUR zum Vorjahr.

Bericht des Kassenprüfers

Wolfgang Strauß hat die Kasse des Jahres 2021 am 06.05.2022 geprüft. Carsten Strotmann hat dazu alle Belege eingescannt und so für die Online–Einsicht verfügbar gemacht. Wolfgang Strauß bescheinigt Ewald Rieger eine vorbildliche Buchhaltung mit vollständigen Belegen und guter Nachvollziehbarkeit.

Der Kassenprüfer empfiehlt die Entlastung der Verwaltung ohne Einschränkung. Die Versammlung entlastet die Verwaltung einstimmig.

Rund um das Forth–Magazin (Ulrich Hoffmann)

Die Vereinszeitschrift Vierte Dimension erscheint weiterhin mit im Schnitt vier Ausgaben pro Jahr; sie ist die letzte auf Papier gedruckte Veröffentlichung in Sachen Forth weltweit. Ulrich Hoffmann betont auch dieses Jahr wieder die Wichtigkeit des Magazins für den Vereinszusammenhalt.

Ulrich Hoffmann dankt Michael Kalus und Wolfgang Strauß für ihre kontinuierliche Arbeit an der Zeitschrift.

Um das Erstellen der Vierten Dimension zu erleichtern und zu vereinheitlichen, wurden die erforderlichen Werkzeuge in einem Docker–Container zusammengestellt. Die Nutzung dieser Technologie wurde bei der Erstellung der letzten Hefte erprobt und funktioniert gut. Damit ist die Mitarbeit am Magazin ohne großen Installationsaufwand auch für neue Interessenten möglich.

Internet–Präsenz (Gerald Wodni)

Gerald Wodni gibt eine Übersicht über die Server des Vereins und erläutert seine Pläne bezüglich Migration, Updates und die Verwaltung der verschiedenen Dienste unserer Infrastruktur. Durch Automatismen und Skripte sollen auch die Mitglieder in die Lage versetzt werden, an der Konfiguration mitzuarbeiten.

Gerald Wodni möchte das Einstellen von Meldungen auf unserer Website einfacher gestalten. Im Moment sind zu viele Schritte nötig, um einen Text einzupflegen.

Gerald Wodni erwähnt den Dienst „Mattermost“ und wirbt für die Nutzung als Kommunikationsplattform für den Verein.

Klaus Schleisiek meldet sich zu Wort. Er beklagt die geringen bis gar nicht vorhandenen Rückmeldungen auf Artikel in der Vereinszeitschrift VD. Er schlägt vor, die Artikel auf der Internetseite zu veröffentlichen. Ulrich Hoffmann gibt zu bedenken, dass Internet und Printmedium zwei unterschiedliche Medien sind. Martin hat die Idee, mit dem Versand der VD einen Hinweis auf die Artikel nebst Kommentarfunktion online zu stellen. Im Anschluss werden noch andere Möglichkeiten diskutiert, unter anderem, Dienste von GitHub zu nutzen.

Bezüglich der Internet–Präsenz fragt Carsten Strotmann, ob der Strato–Server noch gebraucht wird. Bernd Paysan erklärt, dass der E–Mail–Server vorher umgezogen werden muss. Carsten Strotmann wird den Umzug durchführen.

Außendarstellung und Projekte (Bernd Paysan)

Bernd Paysan erwähnt die stark eingeschränkte Darstellung des Vereins nach außen, bedingt durch die Pandemie. Die Frage sei, ob man wieder in präsenz geht. Die Pandemielage sei weiterhin nicht vorhersagbar. Jede Ansammlung von Menschenmengen könne zum Superspreader werden.

Klaus Schleisiek erklärt, auch die Ausrichtung des CCC dieses Jahr ist ungewiss.

Es wird diskutiert, ob, und wenn ja, wie, an der Maker Faire Hannover teilgenommen werden kann.

Bernd Paysan bietet jeden Donnerstag ab 20:00 Uhr einen Forth–Chat auf der von ihm entwickelten Plattform net2o an.

Wolfgang Strauß bietet jeden Montag ab 20:30 Uhr eine Online–Forth–Runde per Videokonferenz an.

Projekte des Vereins werden genannt: Ein neues Forth–Buch (Carsten Strotmann), Projekt Feuerstein (Wolfgang Strauß), Gforth 1.0 (Anton Ertl, Bernd Paysan, Gerald Wodni), VolksForth (Carsten Strotmann, Philip Zembrod).

Entlastung des Direktoriums

Anton Ertl stellt den Antrag, das Direktorium zu entlasten. Der Antrag wird mit 11 Ja–Stimmen, 0 Nein–Stimmen und 2 Enthaltungen angenommen.

Das Direktorium ist damit entlastet.

Wahl des Direktoriums

Als Kandidaten treten erneut an: Ulrich Hoffmann, Bernd Paysan und Gerald Wodni.

Alle Kandidaten werden mit 12 Ja–Stimmen, 0 Nein–Stimmen und einer Enthaltung wiedergewählt.

Alle Gewählten nehmen die Wahl an.

Projekte

VolksForth

Dank der Arbeit von Philip Zembrod und Carsten Strotmann erfährt das VolksForth neue Aufmerksamkeit im Verein.

Klaus Schleisiek regt an, einen Docker–Container zu erstellen, um die Zugänglichkeit des Projektes für Interessenten zu erhöhen.

Philip Zembrod zeigt in einer kurzen Präsentation eine VolksForth–Testsuite in der DOSBox.

Ulrich Hoffmann schlägt vor, ein Arbeitstreffen zum Projekt VolksForth zu organisieren. Es soll herausgearbeitet werden, was das Besondere/Essentielle an VolksForth ist. Philip Zembrod kümmert sich um einen Termin.

Klaus Kohl–Schöpe erwähnt seinen Bestand an Büchern über VolksForth sowie Dokumentation im Word5–Format.

Carsten Strotmann hat ebenfalls Material und kann alte Dateiformate einlesen/umwandeln.

Ein neues Forth–Buch

Carsten Strotmann möchte ein modernes Forth–Buch ins Leben rufen. Er wirbt um Mitarbeit. Ein erstes Treffen soll stattfinden, mögliche Termine werden noch bekanntgegeben.

Verschiedenes

Martin Bitter stellt den Antrag, eine Änderung/Entfernung der 10%–Mindestanwesenheitsklausel im Abschnitt über die Beschlussfähigkeit in der Vereinssatzung durchzuführen.

Die anschließende Diskussion ergibt, dass der Aufwand für eine Änderung der Satzung den Nutzen bei weitem übersteigt.

Martin Bitter zieht daraufhin seinen Antrag zurück.

Der neue Hüter des „Swap“

An dieser Stelle wurde die Sitzung unterbrochen. Der Tradition folgend hatte der *Drachenrat* am Abend zuvor getagt.

Der neue Drachenhüter ist Philip Zembrod.

Wolfgang Strauß hielt die Laudatio.

Der Drache wird auch dieses Jahr wieder per Kurier an seine neue Wirkungsstätte reisen.

Fortsetzung Verschiedenes

Gerald Wodni schlägt für die nächste Forthtagung einen Termin vor Pfingsten vor. Er wirbt auch für einen Besuch der Euroforth, die online und/oder in Rom stattfinden wird.

Klaus Schleisiek erwähnt die Aktivitäten der Silicon Valley Forth Interest Group, die sich einmal im Monat online trifft. Er empfiehlt auch den Besuch der ZOOM–Videokonferenzen der Forth2020–Gruppe, die von Peter Forth organisiert werden.

Schluss

Die Jahresversammlung endet um 11:40 Uhr.

Wesel, den 28.05.2022

gez. Wolfgang Strauß

Abbildung 1: Das obligatorische Gruppenfoto auf den Stufen des Versammlungsortes — nur mehr ein Screenshot.

Abbildung 2: Aber im Sommer dann wieder mit Treppe ...

Stapel überprüfen

Willem Ouwerkerk

Manchmal treten beim Ausführen eines neuen Programms seltsame Fehler auf. In einem Moment funktioniert alles, wie es soll, im nächsten Moment friert dein Programm ein oder stürzt sichtbar ab! Dies kann natürlich ein Programmierfehler in einem Programmteil sein, der selten aufgerufen wird. Es kann aber auch ein Stapelüberlauf vorliegen! Besonders auf Forth–Systemen mit flachen Stacks.

Im Folgenden werden zwei einfache Hilfsmittel vorgestellt, die bei der Erkennung solcher Probleme helfen. Bitte beachten: Es kann auch hilfreich sein, die Faktorisierung des zu untersuchenden Programms zu durchleuchten.

A) Maximale Tiefe des Stacks erreicht?

Füge eine kleine Code–Routine ein, z. B. im Forthwort ; (Semikolon), die den Returnstackzeiger speichert, sobald er einen Referenzwert überschreitet. Wir initialisieren diesen Referenzwert auf das untere Ende des Returnstacks und können damit den Stapelüberlauf detektieren.

Hier die Liste der Kommandos mit ihren Stack–Effekten und den Funktionsbeschreibungen.

(R) (--) Maximale Tiefe des Rückgabestapels beibehalten.

START (--) Initialisierung des Referenzwertes RD

.RD (--) Druckt die maximal aufgezeichnete Tiefe des Rückgabestapels und den verbleibenden Platz. Ist der Abstand negativ, ist ein Überlauf aufgetreten.

; (--) Fügt (R) zu jeder folgenden Definition hinzu.

Ein noForth–Codebeispiel dazu

```
value RD        \ Returnstack Depth

code (R) ( -- )
  \ Save RP in RD when RD is larger than RP
  adr rd & rp cmp >? if,
    rp adr rd & mov
    then, next
end-code

: .RD ( -- )
  \ Number base local in decimal
  base @ >r decimal
  \ Print R-stack depth
  r0 rd - 2/ .
  \ Print remaining R-stack space
  rd s0 - dup 2/ .
  \ Attention to R-stack overflow
  0< if ." !! " then
  \ Restore original number base
  r> base ! ;

: START ( -- )  r0 to rd ; start

: ; ( -- ) postpone (r) postpone ; ; immediate
```

B) Inhalts des Stapels anzeigen

Wir können den Inhalt von einem oder mehreren Stapeln speichern, wenn eine bestimmte Bedingung eintritt. Das kann das Überschreiten eines Grenzwertes sein, oder ein Tastendruck. Die Daten sollten dann irgendwohin sicher gespeichert werden, von wo sie gelesen werden können, ohne dein Programm zu stören.

Es folgt ein Beispiel aus *noForth* mit 14 Elementen auf dem Returnstack. Der Interpreter wird in einem Catch–Frame gestartet und fordert eine Eingabe. Wir drucken nur das Wort, an dem wir gerade arbeiten, und nicht das Wort, das soeben ausgeführt wird.

```
3FE = QUIT
3FC = 03AE ( Datenstapelzeiger )
3FA = 03FC ( Rückgabe-Stapelzeiger )
3F8 = CATCH
3F6 = INTERPRET
3F4 = BL-WORD
3F2 = BL-
3F0 = REFILL
3EE = 0380 ( Ende der TIB )
3EC = C! ( Abgeleitet von dem gespeicherten PC )
3EA = 000A ( Statusregister von MSP430 )
```

Und hier wieder die dazu verwendeten Kommandos:

DEBUG!+ (--) Datenwort speichern und Zeiger erhöhen.

DEBUG (--) Die Unterbrechungsroutine, die die Aktion ausführt.

EMPTY (--) Löschen der gespeicherten Daten.

>CFA (a -- cfa) Finde die CFA der Definition an der Adresse 'a'.

>NFA (cfa -- nfa) Umwandlung der CFA in die NFA.

.NAME (x --) Druckt den zur Adresse 'x' gehörenden Namen oder die Nummer.

RAW (--) Drucken der gespeicherten Daten in Rohform.

SHOW (--) Anzeige der gespeicherten Daten in lesbarer Form.

Im Listing seht ihr, wie zunächst der Returnstack–Pointer gespeichert wird, um dann eine feste Anzahl der Returnstack–Werte in den INFO–Flash des MSP430G2553 zu retten. Dann wird dasselbe mit dem Datenstack gemacht.

Dazu braucht es aber noch eine kleine Code–Routine, die eine Zelle in den Flash–Speicher schreibt, *ohne* den Returnstack zu benutzen:

```
\ Free INFO flash is 1000 to 107F
  ( = 128 Bytes )
routine DEBUG!+ ( -- )
A500 # 012C & mov \ FCTL3
A540 # 0128 & mov \ FCTL1
zz moon ) mov \ Save ZZ to address in MOON
#2 moon add \ To next cell
A500 # 0128 & mov \ FCTL1
A510 # 012C & mov \ FCTL3
rp )+ pc mov ( ret )
end-code
```

Resümee

Der Returnstack vom noForth wächst nach unten in den Datenstack hinein (Abb. 1). Das kann hässliche Effekte erzeugen, wenn der Returnstack zu groß wird. Durch die Routine `(R)` wird der Überlauf abgeschnitten und mit `.RD` kann man eine Fehlermeldung an passender Stelle in seine Definitionen einbauen. `SHOW` hilft dann dabei, die Ursache des Überlaufs zu identifizieren.

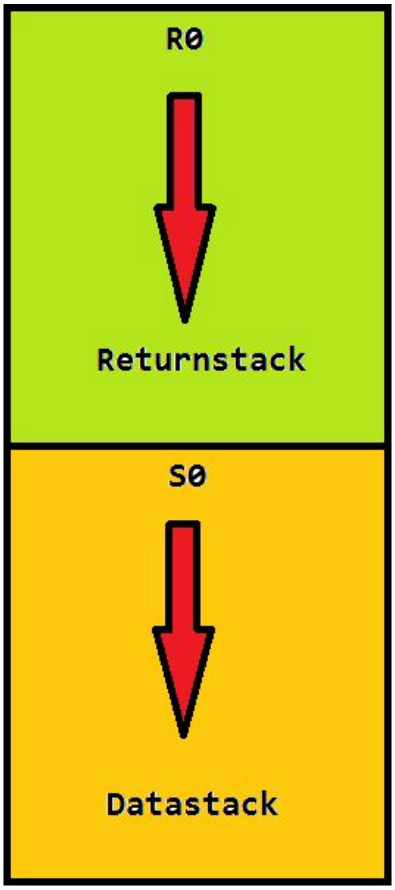


Abbildung 1: Wachstumsrichtungen der Stacks

Listing

```
1   (* Freeze stack content
2
3   This example is used on the mesh network
4   to discover some strange behaviour of a node.
5
6   0)show
7   R-stack = B  ( 11 elements, no problem )
8   A C! 380 REFILL BL- BL-WORD INTERPRET
9      CATCH 3FC 3AE QUIT
10  D-stack = 3
11  25 CAAA CAB4  OK.0
12
13  0)show
14  R-stack = 2F ( 47 elements, stack overflow! )
15  D MS F9F READ-MODE BUSY 0 7FC0 WRITE-DTX? XEMIT?
16     0 7FFC XEMIT >NODE >NODES 400 >OTHERS INFO
17     *INFO (HANDLER) CATCH
18  D-stack = 4
19  0 0 302 1
20  *)
21
22  v: inside also  definitions
23
24  \ Free flash is 1000 to 107F ( = 128 Bytes )
25  \ We use input: P2.7 on the Egel-kit MSP430G2553
26  \ XX = counter  YY = address  ZZ = data
27  routine DEBUG!+ ( -- )
28      A500 # 012C & mov          \ FCTL3
29      A540 # 0128 & mov          \ FCTL1
30      zz  moon ) mov
31           \ Save content of ZZ to address in MOON
32      #2 moon add                \ To next cell
33      A500 # 0128 & mov          \ FCTL1
34      A510 # 012C & mov          \ FCTL3
35      rp )+ pc mov ( ret )
36  end-code
37
38  \ 390 is start of datastack,
39  \ 400 start of returnstack.
40  \ The returnstack grows toward the datastack.
41  routine DEBUG   ( -- )
42      moon push
43      #1 29 & .b bis           \ Led on
44      80 # 2B & .b bic         \ Interrupt done!
45      1000 # moon mov          \ Set start address
46      moon ) zz mov            \ Read first cell
47      #-1 zz cmp  =? if,
48          \ Store data when it's empty
49          rp zz mov    \ First r-stack pointer
50          debug!+ # call
51          rp yy mov    \ And last 52 elements on it
52          34 # xx mov
53          begin,
54              yy )+ zz mov
55              debug!+ # call
56              #1 xx sub
57          0=? until,
58          sp zz mov    \ Second data stack pointer
59          debug!+ # call
60          sp yy mov    \ And last 10 elements on it
61          dm 10 # xx mov
62          begin,
63              yy )+ zz mov
64              debug!+ # call
65              #1 xx sub
66          0=? until,
67      then,
68      rp )+ moon MOV
69      reti
70  END-CODE
71  debug   FFE6 vec!   \ P2 vector
72
73  code INT-OFF  ( -- )   #8 sr bic ,  next  end-code
74  code INT-ON   ( -- )   #8 sr bis ,  next  end-code
```

```
: START       ( -- )
    \ Activate hardware debug interrupt
    startnode         \ Start the node handler too
    led-on  int-off   \ Show startup on a led
    FF 2E *bic        \ Enable P2 completely
    80 2A *bic        \ P2.7 = input
    80 2F *bis        \ With pull-up
    80 29 *bis
    80 2C *bis        \ Interrupt from high to low
    80 2D *bis        \ and active
    80 2B *bic        \ Clear interrupt
    led-off  int-on ;

\ From Albert Nijhof's decompiler:
: CHAR?         ( c -- f )
    ch !  [ ch ~ 1+ ] literal  within ;

: >NFA          ( a -- nfa )
    dup 1 and 0= if             \ even?
        1- dup c@ FF = +  false
          \ skip alignment char
        begin
        over c@ char? while
        true /string  20 over < until
          \ walk backwards through name
        then
        ?dup if
-v:         over c@ 3F and = if \ count ok? \ c
v:          over c@ 7F and = if \ count ok? \ v
                dup 1 and ?exit \ nfa odd -> ok
            then
        then
    then ;


\ Find CFA of the routine where address 'a'
\ is a part of!   This runs backwards from
\ address 'a' to find a valid CFA
: >CFA          ( a -- cfa )
    begin
        dup @ over cell+ <>
               \ Not CFA of code def?
        over @ [ ' . @ ] literal <> and
               \ No CFA of colon def?
        over @ [ ' value @ ] literal <> and
               \ No VALUE VARIABLE or RAM CREATE
        over @ [ ' constant @ ] literal <> and
               \ No CONSTANT
        over @ [ ' 'commands @ ] literal <> and
               \ No ROM CREATE
        over @ [ ' does> @ ] literal <> and
               \ No DOES>
    while
        cell-              \ To previous address
    repeat
    dup cell- @ [ ' literal @ ]
    literal =    \ LITERAL?
    if  cell-  then ;

: .NAME         ( x -- )
    dup C036 chere within if
        \ 'x' is an address in noForth name space?
        >cfa >nfa count 1F and type space
        \ Yes, print name
    else
        \ No, is it NOOP or NEXT routine, etc?
        dup origin C036 within
        if  ." Noop/Etc. "  drop exit  then
        u.                 \ No, it is a number
    then ;

value BR
: BREAK         ( +n -- )
    \ New line after every 10 elements
    br 0A mod 0= if  cr  then  incr br ;


v: forth definitions
: SHOW          ( -- )   \ Show all data readable
    1000 @ -1 = ?exit    \ Empty, do nothing
    cr  ." R-stack = "  1000 @+
    r0 swap - 2/ 1- >r  r@ u.  0 to br
    cell+  ( Skip MOON register )
    33 r@ umin for  break  @+ .name  next
    33 r> 33 umin -  2* +  ( Skip invalid data )
    cr  ." D-stack = " @+ s0 swap - 2/ >r  r@ u.
    cr  dm 10 r> umin for  @+ u.  next  drop ;

: RAW           ( -- )      \ Show data as is
    1000 @ -1 = ?exit       \ Empty, do nothing
    cr  ." RP = " 1000 @+ u.  0 to br
    34 for  break  @+ u.  next
    cr  ." SP = " @+ u.
    cr  dm 10 for  @+ u.  next  drop ;

: EMPTY         ( -- )      \ Erase all data
    1000 recycle  1040 recycle ;

v: fresh
' start  to app
shield DEBUG\  freeze

\ End
```

Calculemus!

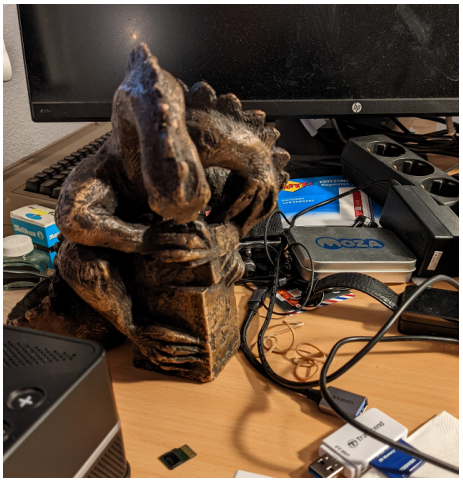
Im 17. Jahrhundert entwickelte Leibniz die Universalsprache *Characteristica Universalis*, um rationale Gedanken auszudrücken. Und *Calculus Ratiocinator* zur Berechnung von Schlussfolgerungen. Er dachte, man könne das so weit treiben, dass jede Meinungsverschiedenheit statt durch einen Kampf oder Ähnliches entschieden werden könne, indem man sie in symbolischer Form aufschreibt und dann berechnet. Sein berühmter Ausruf war „Calculemus!“, lasst uns berechnen. Das ist also die Idee hinter formalen Beweisen: logische und mathematische Argumentation auf eine präzise Rechnung zu reduzieren. Man fängt mit einer sehr kleinen Zahl linguistischer Grundelemente an, Symbolen und grundsätzlichen Regeln, um Aussagen zu treffen und abzuleiten und mit ein paar Axiomen. Das ist ein Axiomensystem. Das Bemerkenswerte daran ist, dass man das von einem kleinen Ausgangspunkt zu immer komplizierteren Aussagen hin aufbauen kann. Und so kann ein Computer einem dabei helfen, einen präzisen, detaillierten Beweis auf die Grundprinzipien zurückzuführen. [Aus: *Die Kunst, Orangen zu stapeln*; Deutschlandfunk; über den Mathematiker Thomas Hales, 05.04.2015, zur Keplerschen Vermutung.]

Irgendwie forthig, oder? mk

Unboxing Swappy

Philip Zembrod

Unser SWAPPY hat in seiner Reisehöhle wohlbehalten zu mir gefunden und fühlt sich offenbar wohl hier. Seht selbst.



Swappy reist neuerdings in einer als DHL–Paket getarnten Reisehöhle. In deren Eingang liegt — natürlich — eine Fußmatte, unter die Swappy den Höhlenschlüssel zu legen pflegt. Er hat extra eine Vertiefung dafür fräsen und sogar Haltemagnete anbringen lassen. Als neuer Drachenhüter war ich begeistert, nicht erst im Keller den Torx–20 holen zu müssen.

Drinnen gibt es dann einen Drachensitz mit extra starkem Sicherheitsgurt, dessen Gurtschlösser man auch aufschließen muss. Und natürlich, in Tuch eingeschlagen, die Truhe mit dem Drachenschatz. Kaum ausgestiegen, hat Swappy dann natürlich erst einmal nichts Besseres zu tun als — Swap!

Anschließend hat Swappy sich dann auf meinem Schreibtisch niedergelassen, und hat sich über mich gewundert. Warum ich TypeScript debuggen würde, statt VolksForth zu schreiben, wollte er wissen. Er meinte auch, ich solle mal aufräumen. Hatte ich eigentlich vor kurzem gemacht …

Forth–Gruppen regional

Bitte erkundigt euch bei den Veranstaltern, ob die Treffen stattfinden. Das kann je nach Pandemie–Lage variieren.

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@net2o.de`
Treffen: Jeden 4. Donnerstag im Monat um 19:00 auf `http://public.senfcall.de/forth-muenchen`, Passwort `over+swap`.

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://www.meetup.com/Essen-Forth-Meetup/`

Dienste der Forth–Gesellschaft

Nextcloud `https://cloud.forth-ev.de`

GitHub `https://github.com/forth-ev`

Twitch `https://www.twitch.tv/4ther`

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 58 46) – 98 04 00 8**$_p$
`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

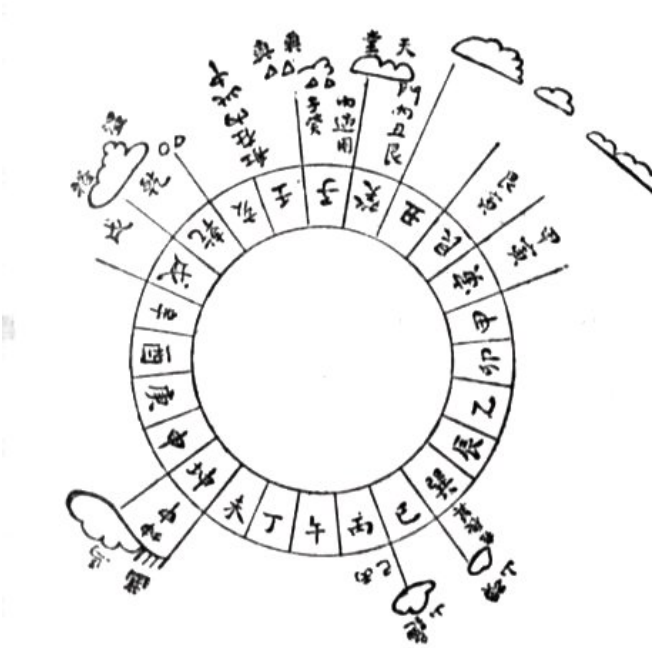
Donnerstags ab 20:00 Uhr
Forth–Chat net2o `forth@bernd` mit dem Key `keysearch kQusJ`, voller Key:
`kQusJzA;7*?t=uy@X}1GWr!+0qqp_Cn176t4(dQ*`

Montags ab 20:30 Uhr
Forth–Abend
Videotreffen (nicht nur) für Forthanfänger
Info und Teilnahmelink: E–Mail an `wost@ewost.de`

Jeder 2. Samstag im Monat
ZOOM–Treffen der Forth2020 Facebook–Gruppe
Infos zur Teilnahme: `www.forth2020.org`

Forth–Sommertreffen (in persona) 15.–17. Juli 2022 im Linux Hotel in Essen–Horst
Infos und Teilnahme siehe Rückseite dieses Heftes

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.

Einladung zum

Forth–Sommertreffen 2022 vom 15. bis 17. Juli im Linux Hotel in Essen–Horst

Das Treffen findet in der Villa Vogelsang statt, Antonienallee 1, 45279 Essen–Horst. Das Gebäude liegt in einem Park mit Aussicht auf die Ruhr. Idylle pur!

Anreise

Essen ist mit der Bahn, Essen–Horst mit der S–Bahn erreichbar; per Auto über die A40 (Essen–Kray oder Essen–Frillendorf). Flughäfen gibt es mindestens drei zur Auswahl: Düsseldorf, Dortmund, Köln. Die liegen aber nicht „um die Ecke“. Und ja, man kann auch zu Fuß oder mit dem Fahrrad kommen — ist erwiesenermaßen machbar!

Anmeldung

`https://sommer.theforth.net/`

Programm (kann sich noch ändern, aber im Prinzip locker, und immer mit Forth dabei. :)

Donnerstag

ab 13:00	Frühankommer, Lobby

Freitag

vormittags	Frühankommer Ausflug
ab 13:00	Ankunft, Lobby
18:00	Gemeinsames Abendessen
20:30	Workshops

Samstag

09:00	Frühstück
10:00	Ausflug
12:30	Mittagessen
14:00	Ausflug
18:00	Gemeinsames Abendessen
20:30	Workshops

Sonntag

09:00	Frühstück
10:00	Spaziergang zur Burg
12:30	Ende, Abreise