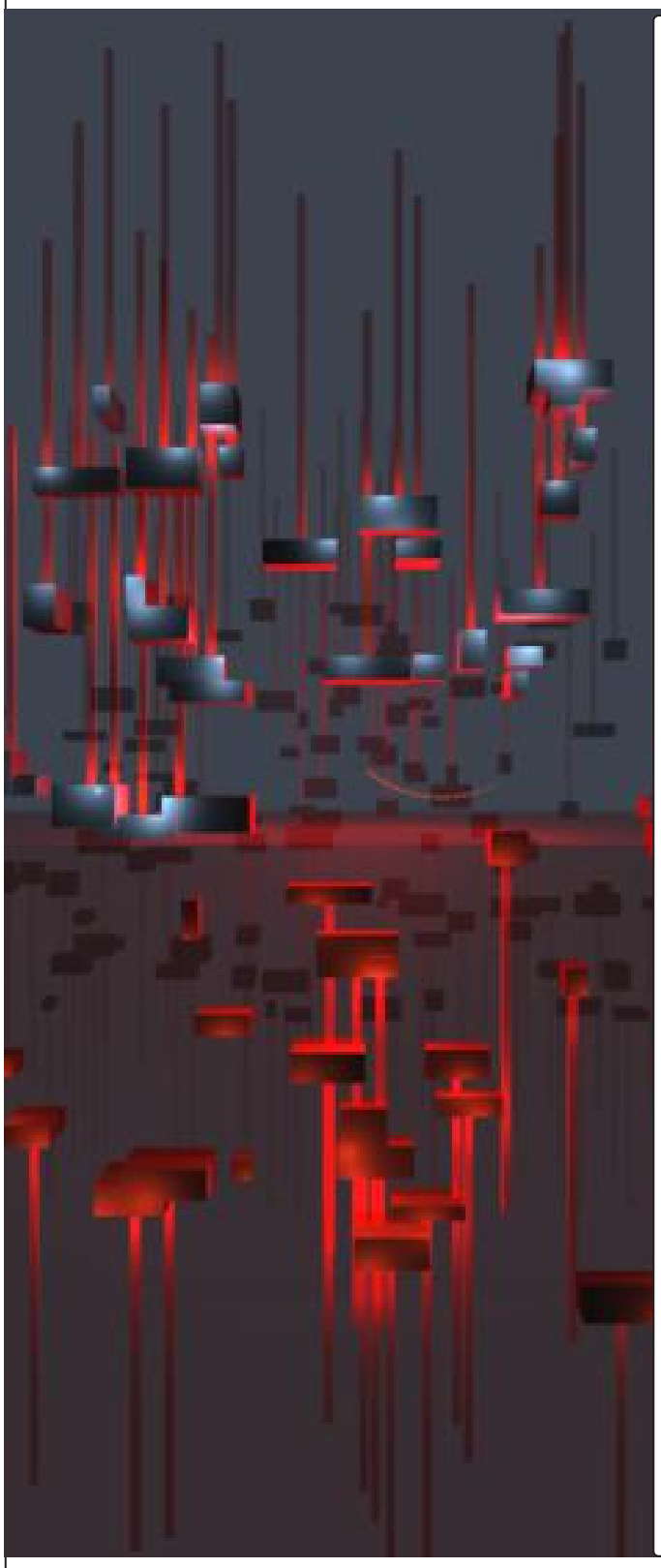
Vierte Dimension

4/2021

Das Forth–Magazin

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



In dieser Ausgabe:

Titelbild: Floating Points Art AUTOR mk

Quelle: Internet, selbst abgewandelt

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 1030
48481 Neuenkirchen
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,

un–gleichzeitig und un–persönlich verlief auch dieses 2. Jahr ohne eine leibhaftige Forthtagung des Vereins, für mich eine Un–Zeit. Wie gerne bin ich doch zu den Forthtagungsorten gereist! Jedes Jahr gab es einen anderen Tagungsort im Lande — Reisen bildet.

So blieb denn nichts anderes übrig, als die Forthtagung in Form einer Videokonferenz abzuhalten, einschließlich einer Mitgliederversammlung der Forthgesellschaft, inklusive Tagung des Drachenrates: So gibt es nun einen neuen Drachenträger: Wolfgang Strauss — herzlichen Glückwunsch!

In den dreiköpfigen Vorstand wurden wiedergewählt Ulrich Hoffmann und Bernd Paysan und wir begrüßen Gerald Wodni als neues Vorstandsmitglied. Carsten Strotmann ist aus dem Vorstand ausgeschieden und hat stattdessen das Forthbüro von Ewald Rieger übernommen. Soweit zum vereinsinternen Stoffwechsel, im Heft mehr dazu.

Gerald Wodni hat inzwischen die Mitschnitte der Videokonferenz in den *4ther–Kanal* bei YouTube hochgeladen. Das ist die schöne Seite der neuen Ungleichzeitigkeit, die Themen können weitere Kreise ziehen, als die leibhaftigen Treffen es vermochten.

Thomas Göppel und Willem Ouwerkerk haben eine ganze Gruppe Forth–Enthusiasten für eine gemeinsame Anstrengung begeistern können. So wächst jetzt seit Mai auf GitHub das Team–Projekt *Forth Works* heran. Wer schon alles dabei ist, lest ihr im Heft.

Danach hat es sich in dem Heft diesmal so ergeben, dass viel gerechnet wurde. Rafael Deliano jongliert *Sinus–Tabellen* in Perfektion auch für kleinste Rechner — cool. Und Jörg Völker spricht mir aus der Seele mit seinem Vorschlag, die Forth–Arithmetik zu entkrampfen durch ein geschickt gewähltes *Floating–Point–Format*. Da bin ich sehr gespannt, wie sich das in den Standard einfügen lassen wird. Ich hätte nichts dagegen, würde das die einfache Basis, und all das klassische Zeugs optional. Matthias Koch demonstriert geradezu spielerisch hohe Forth–Kunst. Er zeigt, wie es mit der klassischen Forth–Arithmetik gelingt, *PNM–Bilder* zu malen. Dieses Ausgabeformat kann gut genutzt werden, um Grafiken aller Art zu erzeugen — gewusst wie.

Und dann gibt es noch einen ersten Eindruck von SWAPs neuer Residenz für ein Jahr. Wenigstens er kann reisen ...

Was die nächste *Forth–Tagung* angeht, sie wird wieder als Videokonferenz stattfinden, denn für Präsenztreffen gibt es auf absehbare Zeit einfach keine Planungssicherheit. So sehen wir uns voraussichtlich im Frühjahr 2022 an den Bildschirmen. Haltet die Augen offen, denn die Ankündigung kommt unter Umständen plötzlich auf unserer Website `www.forth-ev.de`, aber womöglich nicht rechtzeitig für unser nächstes Magazin!

Bis dahin, 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-2021-04`

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

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

Errata zum Heft 4d2021–03

In das vorherige Heft haben sich trotz aller Sorgfalt kleine Fehler eingeschlichen — ihr habt's bemerkt und Bescheid gesagt. Danke.

S. 4

Da unten im Editorial ist mir die Neuschöpfung „Bildschrim“ gelungen. Gemeint war natürlich die elektrisch angesteuerte Anzeige ohne bewegliche Teile zur optischen Signalisierung von veränderlichen Informationen wie Bildern oder Zeichen. Wobei die Begriffe Bildschirm, Monitor, Screen und Display im Deutschen oft durcheinander gehen, obschon für die beiden letztgenannten englischer Herkunft es gleichwohl sprachliche Unterschiede hat. So ist der Bildschirm eigentlich direkt im Gerät verbaut, so wie in einem Laptop. Monitore hingegen werden die Bildschirme genannt, die als einzeln stehende Peripheriegeräte eingesetzt werden. Und die Projektionsfläche eines Projektors, etwa eine Leinwand, ist im Englischen ein „screen“, was wir gerne mit „Schirm“ übersetzen. Und im Deutschen meinen wir alle Anzeigegeräte im weiteren Sinne, wenn wir „Display“ sagen, also in etwa „Anzeige“, zum Beispiel Flip–Dot–Displays.

Dabei hat das Wort „screen“ eine interessante Entwicklung hinter sich:

> „… ‘upright piece of furniture providing protection from heat of a fire, drafts, etc.,’ probably from a shortened variant of Old North French ‘escren’, Old French ‘escran’, a “fire-screen”, perhaps from Middle Dutch ‘scherm’ “screen, cover, shield,” or Frankish ‘skrank’, a “barrier,” from Proto–Germanic ‘skerm’ (source also of Old High German ‘skirm’, ‘skerm’, a “protection” — from root ‘sker–’ “to cut.”
>
> Meaning ‘net–wire frame used in windows and doors’ is recorded from 1859. Meaning ‘flat vertical surface for reception of projected images’ is from 1810, originally in reference to magic lantern shows; later of movies. Transferred sense of ‘cinema world collectively’ is attested from 1914; hence ‘screen test’ (1918), etc. ‘Screen saver’ first attested 1990. ‘Screen printing’ recorded from 1918.“ (`https://www.etymonline.com/word/screen`)

Alles klar?

S. 24

Dort ist die Abbildung 1, Plot einer Ortskurve, so dünn gedruckt worden, dass man „das Ei“ neben dem Kreis kaum noch sehen konnte. Das ist sehr bedauerlich. Auch alle Schrift im Heft wirkte grau und etwas dünn, statt schwarz — Erich Wälde hat das nicht gefallen, danke für den Hinweis! Im PDF des Heftes ist das nicht so zu sehen gewesen, da war alles gut zu erkennen. Rätselhaft!

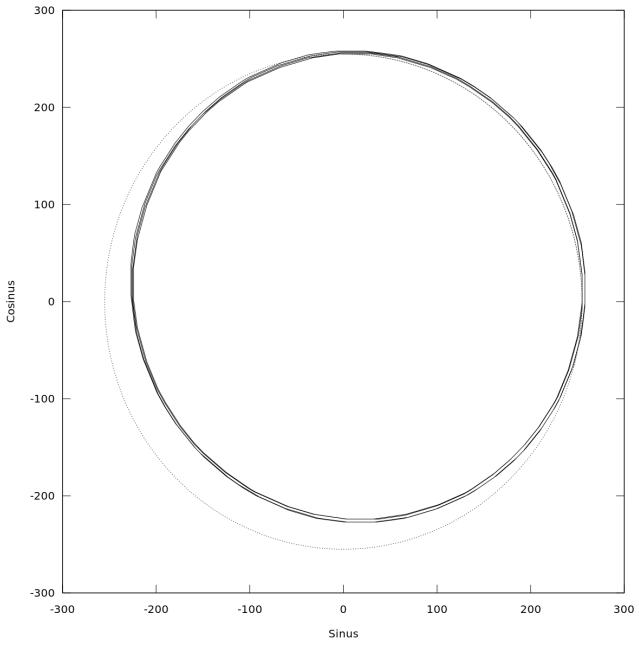


Abbildung 1: Das „Ei“ der Ortskurve

Was war passiert? Bernd Paysan hat genau hingeschaut:

> „Das sieht für mich allerdings nicht nach einem physischen Toner–Problem aus, sondern der Text ist grau gerastert (also definitiv Software). Das zeigt die Makro–Aufnahme sehr deutlich. Das Problem wird umso heftiger, je dünner die Striche sind, … Bei dem Comic–Sans–Text von xkcd ist das Schwarz tatsächlich Schwarz, und nur die vom Anti–Aliasing produzierten grauen Pixel sind gerastert. Das ist nicht das erste Mal, dass wir dieses Problem haben, möglicherweise ist das [beim Drucker] so eingestellt, um Toner zu sparen. Im PDF ist der Text so schwarz wie die Nacht (also Tonerwert #000000), an uns liegt es nicht. Alle eingebetteten Schriften sind Typ1 oder Truetype, also Vektorschriften.“

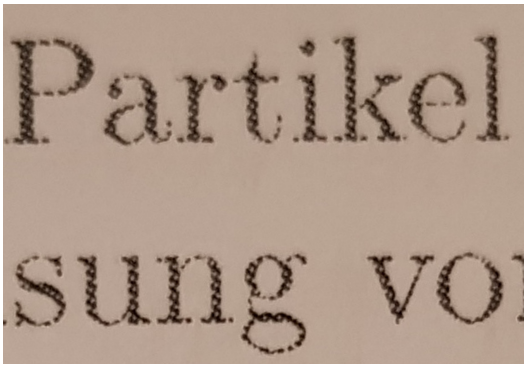


Abbildung 2: Vergrößerung der grau gerasterten Schrift aus Heft 4d2021–03

Wir geloben Besserung und werden mit dem Drucker unseres Magazins reden müssen. Falls ihr Seiten aus dem Heft in schwarzer Schrift haben müsst, holt euch einfach das PDF aus unserem Archiv und druckt es nach, das wird richtig. mka

gforth.org

Da ist sie nun, die eigene Seite für *Gforth*. Dort findet ihr die Links zu gleich zwei Source–Codes zum Herunterladen: alt und stabil und *Bleeding Edge*, letzteres ist der wöchentliche Snapshot. Zum Zeitpunkt der Drucklegung des Heftes war Version 0.7.3 die stabile Fassung, ein ziemlich alter, aber bewährter Release. Bevor man Gforth damit selbst baut, sollte „See `INSTALL.md` for building instructions“ gelesen worden sein.

Für Debian sind die Pakete des letzten Snapshots als Binaries schon dort. Was dort noch fehlt, sind Docker– und Flatpak–Images. Obschon es die Docker–Images schon gibt und Bernd auch schon Flatpak–Images funktionsfertig hat. Nur wie sie verteilt werden können, ist noch nicht entschieden.

Supported Systems

Eine Übersicht darüber ist ebenfalls auf der Homepage.

> „Gforth runs under GNU, BSD, and similar systems, MS Windows and MacOS X and should not be hard to port to other systems supported by GCC. Version 0.7.9_20160714 has been tested successfully on the following platforms:
>
> GNU/Linux — amd64 arm64 armel armhf i386 mips mipsel powerpc
>
> Android/Linux — amd64 arm64 arm i386 mips
>
> Gforth EC(embedded) — r8c 4stack misc 8086
>
> Windows — amd64 i386
>
> MacOS — amd64 i386“

Manual

Über die Menü–Leiste kommt man an das *Gforth–Manual* der Autoren: Bernd Paysan, Anton Ertl und Gerald Wodni; Copyright © 1995 . . . 2020 Free Software Foundation, Inc.

Das Manual ist sehr ausführlich. Es beschreibt in 15 umfangreichen Kapiteln und 4 Anhängen alles zum Gforth. Der vollständige `Word Index` ist ins Manual verlinkt, so dass man die Erklärung eines jeden Forth–Wortes sofort finden kann.

Das 5. Kapitel ist das umfangreichste. Alle Forth–Worte sind darin in Funktionsgruppen untergliedert und ausführlich beschrieben, so dass man sich heraussuchen kann, was man braucht. Diese sehr umfangreiche Sammlung geht dabei weit über die einfachen Standard–Forth–Worte hinaus. Gforth ist ein sehr komfortables, „fettes“ Forthsystem, zu viel, um alle Funktionsgruppen und Untergruppen hier aufzulisten. Geht auf den Link und seht selbst.

Das 3. Kapitel ist ein Forth–Tutorial. Hier wird in Forth und speziell Gforth eingeführt. Dargestellt werden alle *Konzepte* von Forth: Nach Starting Gforth, der Syntax und dem Stack kommen: Arithmetics, Stack Manipulation, Dateien für den Forth–Source Code nutzen, Comments, Colon Definitions, Decompilation, Stack–Effect Comments, Data–Types, Factoring, Designing the stack effect, Local Variables[1], Conditional execution, Flags and Comparisons, General Loops, Counted loops, Recursion, Leaving definitions or loops, Return Stack, Memory, Characters and Strings, Alignment, Floating Point, Files handling, Interpretation and Compilation Semantics and Immediacy, Execution Tokens, Exceptions, Defining Words, Arrays and Records, POSTPONE, Literal, Advanced macros, Compilation Tokens, Wordlists and Search Order.

In den weiteren Kapiteln findet man eine Einführung ins Standard–Forth, die Fehlermeldungen und zwei Tools: *Report the words used, sorted by wordset* und *Stack depth changes during interpretation.* Der Stack–Checker soll in der nächsten Gforth–Version enthalten sein, teilte Anton Ertl auf der diesjährigen Forthtagung mit.

Dann wird der Frage nachgegangen: „Should I use Gforth extensions?“ und beschrieben, wie man Gforth in C–Programme integriert, Emacs mit Gforth verheiratet, was es mit den Image–Files des Gforth auf sich hat, wie die portable Forth–Engine funktioniert, inklusive der Anleitung, Primitives in C einzubinden und die automatische Generation davon. Und schließlich noch für diejenigen, die Forth selbst machen wollen, das Kapitel zum Cross–Compiler für einen Forth–Kernel.

Das Manual lässt keine Wünsche offen, finde ich. Fehlt eigentlich nur noch das Buch dazu, auf Papier, old–fashioned. Zum Reinschreiben seiner Notizen an den Rand.

Gforth 1.0 ?

Auf der Mitgliederversammlung diesen Jahres hat Bernd Paysan angekündigt, dass Gforth 1.0 nun bald kommen wird. In seinem Vortrag ging er näher auf die Veränderungen in der Header–Struktur ein, was die wesentlichste Veränderung sei. Damit würden Schwierigkeiten bei der Generierung des ganzen Forth–Systems beseitigt. Wer selbst schon Forth–Systeme implementiert hat, und seien es auch nur kleinere auf MCUs, wird die Probleme kennen, die mit den Wortkennzeichnungen wie `immediate`, `compile-only`, `defer` oder Zuschreibungen wie `is`, `to` und `does>` verbunden sind. Und wie tückisch Wortlisten sein können. All das wurde gründlich neu gedacht. Ausprobieren kann man das, was bald Gforth 1.0 wird, schon jetzt mit jenem oben genannten *Bleeding–Edge–Snapshot.* Dafür ist der ja da. Da wird nichts im Verborgenen entwickelt. Und ich hoffe dann natürlich auf eine eingehende Beschreibung der neuen Innereien im User–Manual. mka

`https://gforth.org`

`https://git.savannah.gnu.org/git/gforth/`

`hub.docker.com/u/forthy42`

[1] Den Locals kann man sowohl Variablen– wie auch Value–Charakter zuweisen.

Blick über den Tellerrand

UXN, TAL, varvara

Beim Rumstöbern in den dunkleren, staubigen Ecken der elektronisch gestützten Spinnweben bin ich über das Wort UXN gestolpert. „UXN is a stack–machine with 32 instructions“ ist da zu lesen [1]. Ein Emulator, mir deucht. Der Quellcode wohnt auf `sourcehut.org` [2]. Da wird die Erklärung sofort technischer: „An assembler and emulator for the UXN stack–machine, written in ANSI C.“ Aha. Dann machen wir doch mal ein Experiment, oder? Mein PC läuft mit Debian GNU/Linux. Flugs `libsdl2-dev` installiert. Das zieht natürlich ein hübsches Grüppchen an Abhängigkeiten nach sich.

Dann in einem neuen Verzeichnis den Quellcode herunterladen. Das Skript `build.sh` ist übersichtlich. Ein Aufruf ist nach kurzer Zeit fertig und zeigt die Applikation `piano.rom` in einem separaten Fenster. Und sogar die Tonausgabe funktioniert auf meinem Rechner. Faszinierend!

```
$ sudo apt install clang-format libsdl2-dev \
    build-essential
$ mkdir ~/Projekte/39_uxn
$ cd ~/Projekte/39_uxn
$ git clone https://git.sr.ht/~rabbits/uxn
$ cd uxn
$ ./build.sh
```

Soweit war das also nicht speziell schwierig. Und jetzt? Hmmm. Stöbern in der Doku fördert dieses zutage:

```
$ ./bin/uxnasm projects/examples/demos/life.tal \
    bin/life.rom
$ ./bin/uxnemu bin/life.rom
```

Es erscheint ein Fenster mit Conways Game of Life — zwar relativ winzig auf meinem Bildschirm, aber immerhin. D. h., das Programm wohnt in `life.tal`. Reingucken. Und ja, das ist ja schon *lesbar*. Auch wenn ich jetzt sofort nixx versteh. Aber so Fragmente wie dieses

```
@run-cell ( x y neighbours state -- )
    #00 = ,&dead JCN
    &alive
            DUP #02 < ,&dies JCN
            DUP #03 > ,&dies JCN
            &lives POP ,save-cell JSR RTN
            &dies POP POP2 RTN
    &dead
            DUP #03 = ,&birth JCN POP POP2 RTN
            &birth POP ,save-cell JSR RTN

RTN
```

sehen doch ziemlich *stack based* aus, oder? In `projects/examples` wohnen noch mehr Dateien, da kann man stöbern.

Die Referenz für die asm–Sprache *tal*, die auf der uxn–Maschine läuft, findet sich hier [3]. Auf uxn kann eine weitere Schicht namens *varvara* laufen [4]. Sieht aus wie 'ne Art Mini–Betriebssystem. Und das sieht so aus, als würde das oben schon erwähnte Fenster damit gemacht — inklusive Anschluss an Lautsprecher, Maus etc.

Varvara läuft auch auf richtiger Hardware, z. B. der Nintendo DS, Gameboy Advance und ESP32.

SEJO hat ein Tutorial angefangen [5]. NEAUOIRE macht auch was mit richtiger Hardware [6].

Wenn man in dieser Ecke der elektronischen Spinnweben weiter sucht, dann findet man heraus, dass der Erfinder 100RABBITS auch auf das Pseudonym DEVINE LU LINVEGA hört. Er benutzt *plan9* auf einem *RaspberryPi* als seinen Hauptrechner, und er wohnt auf einem Segelboot. Schon diese Kombination von Puzzlesteinchen ist so unwahrscheinlich, dass es wahrscheinlich interessant ist. Viel Spaß beim Stöbern.

Links

[1] `https://wiki.xxiivv.com/site/uxn.html`

[2] `https://sr.ht/~rabbits/uxn/`

[3] `https://wiki.xxiivv.com/site/uxntal.html`

[4] `https://wiki.xxiivv.com/site/varvara.html`

[5] `https://compudanzas.net/uxn_tutorial.html`

[6] `https://llllllll.co/t/uxn-virtual-computer/46103`

Projekt Gemini

Letztes Jahr bin ich über merkwürdige Zufälle und Raumzeit–Verwerfungen an ein Projekt namens Gemini geraten. Gemini ist ein Protokoll, um Dateien über das Netzwerk zur Verfügung zu stellen (wie *gopher* oder *http*). Das Protokoll ist radikal simpel gehalten und bewusst nicht erweiterbar gestaltet. Die einzige *moderne* Kante an dem Ding ist die Verwendung von *transport layer security*, also verschlüsselten Verbindungen. Das plaziert Gemini zwischen Gopher einerseits und HTTP1.0 andererseits.

Dazu gibt es die Definition eines Textformats, so eine Art radikal abgespecktes *Markdown*. Dabei muss man allerdings lernen, dass dieses Format zeilenorientiert arbeitet.

Das führt dazu, dass ein Link auf einer separaten Zeile steht, die mit => anfängt.

Die Beschränkungen im Protokoll und im Textformat sorgen immer mal für hitzige Debatten auf der Mailing–Liste [2]. Meines Erachtens werden die meisten dieser Debatten davon entzündet, dass jemand sein Lieblingsfeature aus der HTTP/HTML–Welt retten will, anstatt zu sehen, was innerhalb der Beschränkungen möglich ist.

Genug Geschwafel. Die Projektseite wohnt hier [1] (ja, ist ein wenig verwaist, was daran liegt, dass der Gründer Solderpunk sich weitgehend zurückgezogen hat). Es gibt Server und Browser/Clients in etlichen Sprachen. Ich verwende *emacs/elpher* [3] und gelegentlich *Lagrange* [4]. Es gibt einen in Javascript realisierten Browser im Browser namens Wobbly [5]. Damit kann man ohne Installation von Extra–Software schon mal gucken gehen. Eine nicht–Nerd–taugliche Einführung namens *geminiquickst.art* ist sowohl via http [6] als auch via gemini [7] verfügbar. Einer von mehreren *feed aggregators* ist Antenna [8].

Warum ist das interessant? Weil die Programmier– und Ausdrucksmöglichkeiten so stark eingeschränkt sind, treffen sich hier die Minimalisten. Es ist z. B. möglich, den TLS–Teil vom Server komplett abzutrennen und `relayd/nginx` oder ähnlich vor einen Server zu schalten, der nur via `stdin` und `stdout` operiert [9]. So etwas ist auch auf winzigen Maschinen und Mikrocontrollern denkbar. Allerdings hatte ich noch nicht die Nerven, einen minimalen Server in Forth zu schreiben. Wo das hinführen könnte und wie man unter diesen Einschränkungen auch so etwas wie Netz–Applikationen realisieren kann, beschreibt Solderpunk hier [10]. Viel Spaß beim Stöbern!

Links

[1] `https://gemini.circumlunar.space/`

[2] `https://lists.orbitalfox.eu/listinfo/gemini`

[3] `https://thelambdalab.xyz/elpher/`

[4] `https://gmi.skyjake.fi/lagrange/`

[5] `https://warmedal.se/~wobbly/`

[6] `https://geminiquickst.art/`

[7] `gemini://geminiquickst.art/`

[8] `gemini://warmedal.se/~antenna/`

[9] `https://tildegit.org/solene/vger`

[10] `gemini://gemini.circumlunar.space/users/solderpunk/gemlog/a-vision-for-gemini-applications.gmi`

Erich Wälde

Vintage — Das erste Forth als Listing auf GitHub

Ken Boak hat das aus einem Ausdruck rekonstruierte *erste Forth* auf GitHub eingestellt:

`https://github.com/monsonite/1968-FORTH`

> „Original Forth for IBM 1130 written by Charles Moore in 1968.
>
> This implementation consists of 2 parts, a 645 line file written in IBM 1130 assembly language, and a 235 line text file listing, dumped from the IBM 1130 disk, but originally created as a deck of punched cards.
>
> These listings came to light sometime around 2011, and in March 2018 attracted the interest of Carl Claunch, an IBM 1130 restoration enthusiast. Most of the detective work concerning the operation of this Forth has been performed and documented by Carl Claunch.
>
> Please see `FORTH-68_notes.txt` and '`notes on FORTH assem code.pdf`' for Carl's invaluable documentation.
>
> Part of the difficulty in understanding the listings, was the use of the IBM proprietary EBCDIC (Extended Binary Coded Decimal Interchange Code) on the IBM 1130. Charles Moore created his own character coding to make the use of hexadecimal numbers easier and to facilitate alphabetical dictionary searches. Neither of these character coding schemes are compatible with ascii. . . .“

Um sich vorzustellen, was das damals für eine Programmierarbeit gewesen ist, hier die einführende Beschreibung des IBM–Assemblers. Ein Assembler, der eine Macro–Library hatte und `names` kannte — ein Konzept, das Charles Moore wohl zu seinem Forth inspiriert haben könnte.

> „The 1130 Assembler language permits the programmer to write (code) source programs in a symbolic language that is more meaningful and easier to use than the binary machine language. The symbolic language provides the programmer with mnemonic operation codes, special characters, and other necessary symbols. The use of symbolic labels (names) makes a program independent of actual machine locations. Unique mnemonic operation codes are included to relieve the programmer of coding the machine–language instruction modifications.
>
> Macro instructions are included, which (in conjunction with the program loaders) automatically provide linkage to the IBM supplied subroutines. The subroutines provided are listed in the Subroutine Library, which is described later.
>
> The source program, punched in either cards or paper tape, is assembled into machine language by

> the 1130 Assembler. The object program is punched into the first 20 columns of the source card (by the card assembler) during the second pass of the two–pass assembler. This deck is termed the 'list deck'. The paper tape assembler punches the object program during the second pass of the source program.
>
> Before the object program can be loaded into the CPU for execution, it must be acted upon by the Compressor Program. This program 'compresses' the object information from several list–deck cards into one card. This deck, known as the Compressed Binary Object Program deck, can be loaded with the Relocatable Loader or it can be converted into core–image format by the Core–image Converter Program. The core–image format deck can be loaded by the Core–image Loader. Either the Relocating Loader or the Core–image Converter Program will select (and supply the necessary linkage for) the subroutines used by the object program. [IBM 1130 System Summary; sixth edition; P.9]"

IBM warb damals damit, dass sich das 1130–System am „Operator" orientiert habe. Nur ein Minimum an Training und Erfahrung sei nötig, damit Ingenieure individuelle Projekte und Forscher Probleme lösen können. IBM befreie den Benutzer von detaillierten Programmierkenntnissen, weil er mit vertrauten Sprachen arbeiten könne, eben jenem Assembler und — Fortran. Das kompakte System biete alles:

> „... the IBM 1131 Central Processing Unit (CPU) with core storage capacity ranging from 4,096 to 32,768 sixteen–bit words. The core storage cycle time is 2.2 or 3.6 microseconds to access a full word of storage."

Dabei war die CPU allein so groß wie ein Schreibtisch. Erlaubt euch mal den Spaß und schaut Bilder davon im Internet an, ich finde es beeindruckend. In Anbetracht solcher Beschränkungen wird mir die Notwendigkeit von FORTH tatsächlich einleuchtender. Der Gigabyte–RAM– und Terabyte–Disk–Space– bzw. SSD–gewohnte Python–Programmierer von heute kann sich das eigentlich nicht mehr vorstellen, oder? mk

Prof. Dr. Karl Meinzer, DJ4ZC, zum DARC–Ehrenmitglied ernannt

Frug mich der Editor, dem ein gewisser Jürgen geschrieben hatte, was es denn damit auf sich habe, und warum das was in der Vierten Dimension, also hier, verloren hätte. Tja. Keine Ahnung. Aber ich muss ja nicht alles wissen, um *etwas* zu wissen.

Der DARC ist der „Deutsche Amateur–Radio–Club e.V." Den kenn ich. Der hat's mit dem Thema Amateurfunk, also elektro–magnetische Wellen machen zur Verständigung der Leute auf dem Planeten, und zur Erforschung der Möglichkeiten. Und DJ4ZC ist das persönliche Rufzeichen von Herrn Meinzer. Und der wurde jetzt zum Ehrenmitglied ernannt. Soso. Aha. Na, dann wird er ja wohl was gemacht haben dafür. Die komplette Meldung kann man auf der Webseite vom DARC [1] lesen. Dort ist aber lediglich erwähnt:

> „Sein Berufsleben verbrachte er an der Uni Marburg, hier besonders im Entwicklungslabor für Elektronik bis zu seiner Pensionierung im Jahr 2005. Die Räume des ZEL waren zugleich Sitz der AMSAT–DL."

Schon wieder so 'ne Abkürzung. Die AMSAT ist eine Vereinigung, die sich um Amateurfunkzeugs an Bord von Satelliten kümmert. Herr Meinzer hat also mit denen zusammengearbeitet.

Wenn man weiter gräbt und die Links in Jürgens E–Mail verwendet, dann gelangt man zu einem Artikel [3] über *IPS*. Und dort steht: „IPS — Interpreter for Process Structures developed by Prof. Dr. Karl Meinzer, DJ4ZC."

Ah, jetzt! Man könnte fast schon ahnen, dass das IPS womöglich ein forthiges Ding ist. Und das bestätigt sich dann auch. Den zugehörigen Code gibt es auf dem berühmten *GitHub* [4]. Und in der Dokumentation [5] stehen dann so Sachen wie

> „The programming system IPS described in this book uses a general approach somewhat similar to FORTH (Moore [1]), but was designed to provide more demanding user interaction mechanisms and to use the typical low–cost peripherals of microcomputers."

`SWAP` und `DUP` findet sich da. `DROP` heißt `DEL`, und `ROT` heißt `RTU`. So auf die Schnelle. Viel tiefer bin ich da nicht eingedrungen. Für Leute, die ihr eigenes Forth schreiben, ist das wahrscheinlich eine Fundgrube.

Geneigte Leser ahnen schon, der oben erwähnte Jürgen hat eine Edition dieser Dokumentation fabriziert, die bei einem bekannten Händler für Kindle–Lesegeräte erhältlich ist. Aber da kann ich nicht mitlesen. Erich Wälde, DL7TUX

Links

[1] DARC Webseite `https://www.darc.de/home` unter Meldungen vom 13.11.2021.

[2] AMSAT Webseite `https://www.amsat.org/`

[3] `https://amsat-dl.org/en/ips-high-level-programming-of-small-systems-for-the-amsat-space-projects`

[4] IPS Quellen: `https://github.com/amsat-dl/IPS`

[5] IPS Dokumentation: `https://github.com/amsat-dl/IPS/tree/master/Documentation`

Top Programming Languages 2021

Die Sinnhaftigkeit mag zwar bezweifelt werden, scheint aber genug Unterhaltungswert zu haben, dass alle Jahre wieder in der Zeitschrift ein Artikel erscheint. Diesmal im Newsletter *Embedded Software Engineering*. In dem Beitrag wird eine Liste aus `spectrum.ieee.org` verwendet. Hier ist sie, wenn man nur „embedded“ selektiert:

1. Python (100.0)[2]
2. C (94.7)
3. C++ (92.4)
4. C# (82.4)
5. Arduino (68.4)
6. Rust (63.1)
7. Assembly (62.8)
8. Verilog (40.3)
9. Ada (38.8)
10. VHDL (38.5)
11. D (36.6)
12. LabView (35.8)
13. Elixir (29.2)
14. TCL (27.6)
15. Erlang (18.3)
16. Forth (18.2)
17. LadderLogic (14.3)

D. h., Forth taucht schon noch in Statistiken auf. Man findet es aber schneller, wenn man solche Listen von unten her liest.

MfG JRD

`https://www.embedded-software-engineering.de/die-am-haeufigsten-eingesetzten-embedded-programmiersprachen-2021-a-1077382/`

`https://spectrum.ieee.org/top-programming-languages/#/index/2021/0/0/0/1/1/50/1/50/1/50/1/30/1/30/1/20/1/20/1/5/1/50/1/100/1/50/`

Zitate

„Forth really doesn't need corporate support to prosper. It's an individual thing rather than an collektive. ... (und gefragt, ob er Forth auch für sich verwendet:) Yes of course. I have a wonderful version of colorforth, which I use almost every day. I'm not doing anything impressive with it, just messing around with computers. Basically I'm retired, I spent a lot of time exercising, trying to stay alive, making food. (It is a) very pleasent live, and I certainly don't want to be under any kind of pressure to produce things. So I'm basically not. I have a number of applications in colorforth that I enjoy, they are (probably) not portable. Forth to me ... is a personal language and I'm doing it my way, and I will be happy to share ideas, but no one will be interested in my forth code.“

Quelle: Chuck Moore and Dutch Fig–Talk, Zoom–Meeting 9. Oktober 2021; Recordingtime 01:14:31 ff.

Abbildung 3: Screenshot of Chuck Moore at that Zoom meeting 2021

[2] Ranking, mit dem Score in Klammern dahinter. „Rankings are created by weighting and combining 11 metrics from eight sources: CareerBuilder, GitHub, Google, Hacker News, the IEEE, Reddit, Stack Overflow, and Twitter.“ [spectrum.ieee.org]

Project Forth Works — Embrace The Difference

Willem Ouwerkerk et al.

This project started after two lectures for the Forth2020 group in May 2021, by Thomas Göppel *and* Willem Ouwerkerk. *Both highlighted the need to share more code within the Forth community. To try out this idea, a workgroup was formed consisting of:* Martin Bitter, Thomas Göppel, Ulrich Hoffman *and* Willem Ouwerkerk. *Later* Jeroen Hoekstra *was also asked to join. We are now well underway with the project.*

What this project is all about . . .

Forth would benefit greatly from an active community sharing sources and solutions. But our problem lies in the saying:

> "When you have seen one Forth, you have seen one Forth."

With this project we would like to embrace our differences. And start sharing, despite having all the different Forth dialects.

Core idea

The core idea of this project is to use a simple, generic version of Forth to spread ideas, algorithms, protocols, applications and hardware drivers. This generic version of Forth guarantees that it can be understood and used by (almost) everybody in the Forth community.

`https://github.com/project-forth-works/project-forth-works.github.io/blob/main/minimalforth.md`

In addition, other implementations can be added in any dialect, version or standard of Forth, and covering any CPU. Having multiple solutions in different dialects of Forth is really seen as a benefit in this project!

Circular buffer in RAM				
Ram	X0	X1	X2	X3
Index	0	1	2	3

Name	Other data structures
PTR	Pointer to oldest data
#SIZE	Constant with length of buffer

Figure 1: Example of structured description.

Structure

Without some structure this project is doomed to fail. That is why a standard structure for each topic is envisioned. The idea is to start with a description of the idea/algorithm, clarify that idea or algorithm in pseudo code and add an example implementation in *Generic Forth*. Links to more detailed information, a description of any pitfalls encountered, etc. are also very welcome. And we're really serious in embracing the differences.

The example in *Generic Forth* is always the starting point, but examples in any Forth dialect and for any other CPU can be added to the original idea. The more, the better. And different solutions for the same problem are also very welcome.

The project can also be a place to ask for help when you are stuck or stranded somewhere. Say you have trouble interfacing to a certain peripheral device. The project could already contain a solution. Or help you find somebody who has a solution. Or help you find somebody with whom you could partner up to try to find a solution together.

Participate

If you want to participate, the two main things you can do are:

1) Add code to an existing topic

Add something like a different implementation to an already existing topic. This can be an alternative way of solving the problem. Or it can be a more specific implementation in a different dialect, or a solution for a different CPU.

A short description of the difference of a new implementation plus the source are all that is need. But you can add interesting facts if you like to do so. For instance benchmarks or maybe there are known limitations to your solution, maybe you know who invented your algorithm. Et cetera, et cetera. Lets learn and have fun!

2) Initiate a new topic

You can initiate a new topic. Upload any project or algorithm worth sharing in the appropriate category, and describe it as detailed as is useful. How to do that is explained below.

How to describe a new topic?

1. Describe the problem that is solved
2. Give some possible usage examples (if not self–evident)
3. Give a detailed description of the algorithm solving the problem
4. Then a more logical decription in pseudo code

5. Add a Generic Forth code example, with the focus on clarity rather than efficiency or aesthetics
6. Add any background documentation and describe encountered pitfalls etc.
7. Add an implementation (or more than one) in your preferred Forth dialect/CPU
8. When it is an unfinished project, also describe as much as is possible of the problem(s) you encountered

To make it more concrete, on the template page there is a detailed example of how we believe a topic should be worked out and published.

`https://github.com/project-forth-works/project-forth-works.github.io/tree/main/Template`

Visit the Project Forth Works at GitHub

There are many well known words that we did not (yet) select for the Generic Forth word set. Look at the link below for some implementation examples of these words:

`https://project-forth-works.github.io/well-known-words.txt`

```
: NIP  ( x y -- y )     swap drop ;
: TUCK ( x y -- y x y ) swap over ;
```

Figure 2: Two well known words.

Changes

In the meantime the composition of the group has changed: Martin and Thomas have taken a step backward and Albert Nijhof has joined the core group.

Further ideas — What could you contribute?

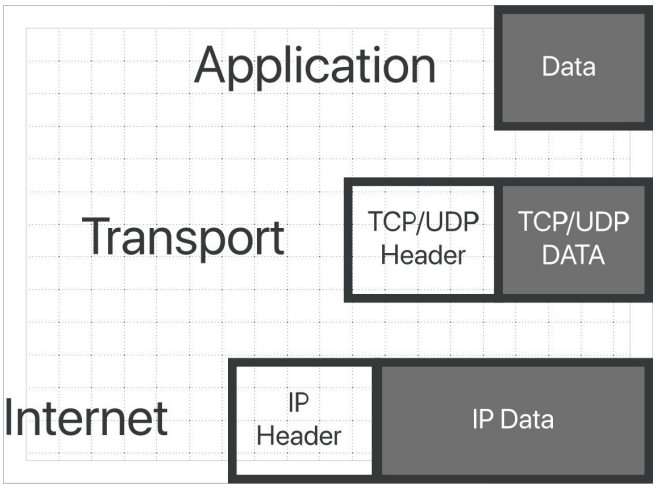


Figure 3: Topics to contribute.

If you are interested please contact us (Tab.1)

Willem Ouwerkerk	sp332236@telfort.nl
Jeroen Hoekstra	canterstate@yahoo.com
Ulrich Hoffmann	uho@xlerb.de
Albert Nijhof	anij@hccnet.nl

Table 1: Contact

DUMP as an example for Project Forth Works

DUMP Idea

A DUMP utility is a software tool that allows to inspect main memory and display it in a user readable form. Typically output is composed of several lines that have the structure: Display bytes starting in memory at a given address in hex (or other radix). Behind them is the ASCII representation of these bytes. The DUMP utility is normally called DUMP and is invoked with parameters which specify what memory to inspect, often start address and length.

DUMP Implementation

One way to implement DUMP is to iterate in a loop line by line through memory. Say you want to display 16 bytes in each line, then this loop will start at the start address and increment the address by 16 in each loop iteration. (If you want a different number of bytes in each line, adjust accordingly).

When you display memory in a single line you first output the current address and then have two loops that run one after the other iterating both from 0 to 15. The first loop outputs bytes with two hexadecimal digits (or in decimal, or whatever you intend) and the second loop ouputs the indivual bytes as ASCII characters.

As some characters might control the output in a special way (so called control characters such as 07 bell, 0A linefeed, 0C formfeed) it is wise to just ouput a period instead of the actual character, in order to get a well formatted display.

Have a look here for the real example:

`https://github.com/project-forth-works/project-forth-works/tree/main/System-Software/dump`

Pseudo code for the DUMP implementation

```
Function: dump-line ( address -- )
  output address (possibly right aligned)
  output ":" and some space

  LOOP i from 0 to 15:
     output byte in memory at (address+i) as two hexdecimal digits
     (or in another radix if desired)
     output some space

  LOOP i from 0 to 15:
     output byte in memory at (address+i) as an ASCII character,
     "." if that character is a control character (byte<32)

Function: dump ( address length -- )
   WHILE length is positive:
        dump-line at address
        increase address by 16
        decrease length by 16
```

DUMP in Generic Forth

For a definition of `BOUNDS` see the list with well known words.

```
hex
: .BYTE ( c -- )   0 <# # # #> type space ;    \ Print hex byte
: .ADDR ( x -- )   0 <# # # # # #> type ;      \ Print 16-bit hex address
\ : .ADDR ( x -- ) 0 <# # # # # # # #> type ;  \ Print 24-bit hex address

: PEMIT ( c -- )   dup 7F < and BL max emit ;  \ Protected EMIT
: PTYPE ( a u -- ) bounds do i c@ pemit loop ; \ Protected TYPE

: DUMP-LINE ( a u -- )
    over .addr ." : "               \ Print addres in hex
    2dup bounds do i c@ .byte loop  \ Dump one line in hex
    [char] | emit ptype ." | " ;    \ Print 16 bytes in visible ASCII

: DUMP ( a u -- )
    hex 10 /  0 do
        cr dup 10 dump-line
        10 +  key? if leave then    \ Adjust address, test any key to stop
    loop  drop ;


4400 40 dump
4400: 36 45 16 B3 01 20 30 46 36 90 00 90 07 34 07 93 |6E    0F6    4  |
4410: 37 44 03 20 36 80 01 88 05 56 00 4F 36 90 00 70 |7D  6    V O6  p|
4420: 06 34 06 11 24 83 84 47 00 00 07 46 00 4F 36 80 | 4  $  G   F O6 |
4430: 01 78 05 56 00 4F 00 00 83 84 4E 4F 4F 50 00 44 | x V O    NOOP D|

```

Link

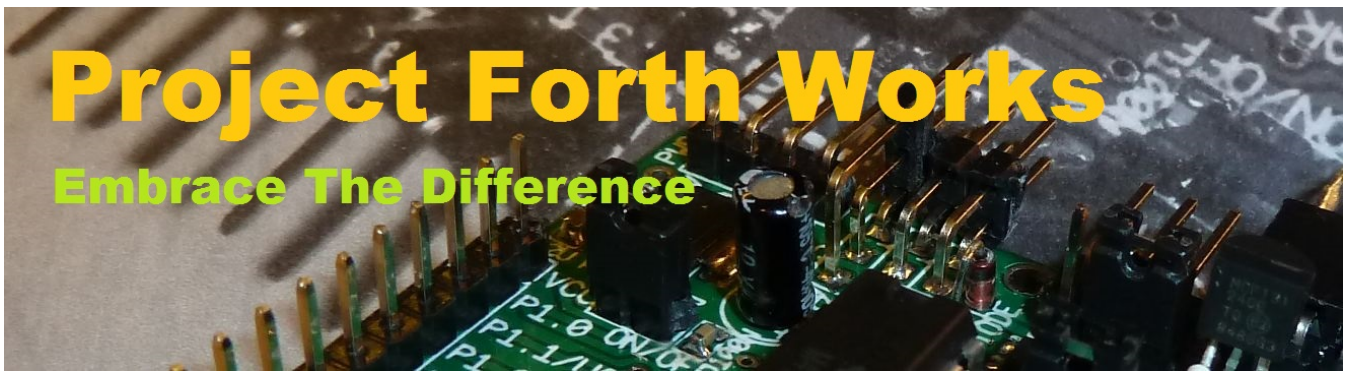


Figure 4: `https://github.com/project-forth-works/project-forth-works`

Vintage: Der „Micro–Ordinateur“ Hector

Rafael Deliano

Der englische Jupiter Ace von 1982 bis 1983 war nicht der einzige Homecomputer mit FORTH im ROM.

Der US–Hersteller des *Interact Family Computer Model One* (Abb. 1) ging 1979 sofort nach Markteinführung pleite. Der 2 MHz Intel 8080A mit 8 bzw. 16 kB[1] RAM und 2 kB ROM wurde mit einem sehr frugalen BASIC ausgeliefert, das von Band geladen werden musste. Das eingebaute Laufwerk hatte, um Kosten zu sparen, den Löschkopf nicht verdrahtet: also Cassetten extern löschen.

Abbildung 1: Der „Interact“

Der Lagerbestand wurde dann von *MicroVideo* in den USA noch bis 1980 abverkauft. Einige Verbesserungen wie 32 kB RAM Speicherkarte und leistungsfähigeres Microsoft BASIC halfen nur bedingt. Auch für $ 200 konkurrierte die Maschine nicht gut gegen die etablierte Z80–Konkurrenz *Tandy TRS* „trash“[2] 80.

Für den Anwender waren Homecomputer Spielkonsolen. Für die Kaufentscheidung war das Angebot an Spielen relevant. BASIC war nur für eine Minderheit von Benutzern von Interesse. Die dritte Alternative, die immer angeboten wurde, war Assembler. Sie richtete sich eher nicht an den Endkunden, sondern an die Softwareentwickler.

Victor Lambda

Im Januar 1980 erwarb dann *Lambda Systems* von Michel Henric in Toulouse die Rechte für Interact, ging aber im Juli 1981 pleite. Abgesehen vom Netzteil war die nötige Modifikation gewesen, das Video auf PAL umzustellen. Das Gerät ist nun in Frankreich am Markt eingeführt und wird von *Micronique* übernommen. Diese Firma wurde 1975 von M. René Benech gegründet, kein Start–Up der Goldgräberzeit[3]. Die Firma verfügte über eine eigene Fertigung und technisches Know–How, und existiert heute noch.

Das neue, nun in Frankreich gefertigte Gerät wurde rasch vom „VICTOR Lambda I“ 1981 (16 kB RAM) zum „VICTOR Lambda II“ (48 kB RAM) ausgebaut und 1983 als „VICTOR 2HR“[4] mit besserer Grafik überarbeitet. Es waren aber immer noch 8080 Geräte mit schlechter Tastatur.

Zur Popularität trug bei, dass Software und Dokumentation in französischer Sprache gehalten waren. Das war umgekehrt aber auch ein Grund, warum das Gerät im übrigen Europa unbekannt und unverkäuflich war.

Hector

1983 erfolgte die Umbenennung in *Hector*, um Verwechslung mit dem amerikanischen Hersteller Victor Technologies zu vermeiden. Diese Firma von Chuck Peddles bot damals kurzzeitig ein Konkurrenzprodukt zum IBM–PC an.

Außer einer besseren Tastatur gab es nun eine Z80–CPU (Tab. 1). Die kleinen Stückzahlen bei laufenden Entwicklungskosten schlugen sich auf den Preis nieder. Anders als Interact entschied man sich bewusst für einen hohen Preis und versuchte technisch an der Spitze mitzuhalten. Abgesehen von besserem Video also auch mehr Speicher. Doch Programme von Cassette mit 1500 Baud zu laden wird jedoch bei 64 kB RAM langsam unakzeptabel.

Tabelle 1: Varianten

	HECTOR 1	HECTOR 2HR	HECTOR HRX
CPU	Z80 1,7MHz	Z80 5MHz	Z80 5MHz
ROM	4kB	4kB	16kB
RAM	16kB	48kB	64kB
BASIC	4,5kB	32kB	32kB

Der 2HR hatte als Alternative zu BASIC bereits FORTH oder Assembler vom Band dabei.

Hector HRX

Der hatte nun FORTH in 16 kB ROM[5]. Es lag in der Memory–Map im Bereich des 16 kB Write–Only–Video–RAMs. Das war wesentlich angenehmer als Laden von Cassette. FORTH als „novelty item“ wurde damals in der Presse durchweg positiv besprochen. Doch da es anders als BASIC keine etablierten Benutzer gab, verbesserte es die Verkaufszahlen aber nicht.

[1] Nach der alten Interpretation: 1 kB (Kilobyte) = 1024 Byte; heute 1 KiB

[2] „trash“ war die liebevolle Aussprache vieler Anwender für „TRS“.

[3] Die „Goldgräberzeit“ war ab 1977. Vorher war der Mikroprozessor dem breiten Publikum unbekannt. *Interact* und *Lambda Systems* waren unterfinanzierte Neugründungen, die am Consumermarkt die „schnelle Mark“ machen wollten.

[4] HR = Haute Resolution

[5] real natürlich EPROM

Abbildung 2: Der „Hector HRX“

Die konventionelle Alternative war wieder ein 38 kB Microsoft BASIC vom Band. Als Optionen kamen nun auch Pascal, Cobol und Fortran. Für Programmiersprachen von Softwarefirmen in ROMs wären Lizenzgebühren fällig geworden. FORTH war die billige Alternative für das ROM–Experiment.

Mit dem Cassetten–Recorder wurde man allerdings nicht wie Apple][oder CP/M als Business–Kleincomputer wahrgenommen. Deshalb kam als klotzige Neuerung eine externe Baugruppe „DISC 2“ mit zwei Disketten–Laufwerken dazu, mit RAM und CPU, die als CP/M 2.2 System gedacht war. Hector wäre dabei nur das Terminal gewesen.

Hector MX80

Der HRX hatte nur die üblichen 22 Zeilen x 40 Zeichen Text, die man auf kleinem PAL–Fernseher noch gut darstellen konnte. Das reicht für die BASIC–Programmierung. Drucker, wie der 1980 eingeführte Epson MX80, wurden billiger, die Computer–Bildschirme besser. Nützliche Anwendung war nun die Textverarbeitung, die aber 80 Zeichen erforderte. Also wurde das Video noch einmal entsprechend aufgebohrt.

Das 64 kB ROM enthielt nun BASIC 3X, HRX–Forth, Logo und einen Editor/Assembler. Die Firma zielte auf staatliche Bildungseinrichtungen, fand aber nur eine sehr geringe Verbreitung. Die Produktion wurde dann 1985 eingestellt. In dem Jahr kamen mit *Amiga* und *Atari ST* die 16 Bit Home–Computer auf den Markt. Und die Disketten lösten die Cassetten endgültig ab.

Schaut euch dazu mal die Webseite der Retro–Anwender [1] an, die ist zwar wenig komfortabel, aber inhaltsreich.

CYBER 310

Zu den interessanteren Zusatzprodukten für den Bereich Bildung gehörte ein Roboterarm, der von *Cyber Robotics Ltd* in England hergestellt wurde (Abb. 3). Ab 1982 wurde von David N. Sands dafür ROBOFORTH angeboten. Seine Firma existiert heute noch [2]. Ihr könnt euch auch mal seine ROBOFORTH Videos auf YouTube ansehen.

Aber das ist bereits eine andere Geschichte.

Abbildung 3: Der „CYBER 310“

Links

[1] `hectorvictor.free.fr/`

[2] `www.strobotics.com/`

Interpolierte Sinus–Tabelle

Rafael Deliano

Ein altes Thema, für das es viele Lösungen gibt ...

Einen 12–Bit–Sinus mit 256 Stützstellen konnte man im Zeitalter der PMOS–ROMs aus drei 256x4–Bit–Speichern direkt aufbauen [1] (Abb. 1). *National Semiconductor* lieferte damals MM4220BM/MM5220BM Memory–Chips (ROM) als „Sine Look–Up Table“, die aber mit ihrer 7–Bit–Adresse nur 8–Bit–Daten ausgaben. Das ROM MM4220BN/MM5220BN war die „Arctangent Look–Up Table“. In den meisten Anwendungen benötigte `ATAN` weniger Genauigkeit als `SINUS`. Für bessere Auflösung des Sinus, d. h. 12 Bit (Abb. 2), war damals die Zahl der ROMs für direkte Implementierung unpraktikabel.

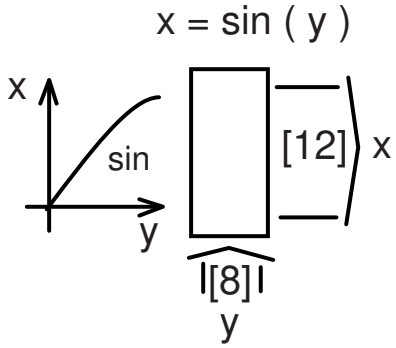


Abbildung 1: Kleine ROMs

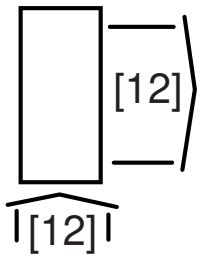


Abbildung 2: Riesige ROMs

Interpolation

So griff man in die Trickkiste bzw. seine Formelsammlung und teilt das Datenwort y in zwei Teilworte — siehe Excurs weiter unten. Man hat dann zwei ROMs und einen Addierer. Damit verkleinert sich das erste ROM auf drei Speicher–ICs. Für das zweite ROM genügte durch vergröberte Auflösung ein Speicher (Abb. 4). Der 12–Bit–Addierer aus drei kaskadierbaren handelsüblichen 4–Bit–MSI–ICs[1] war billig. Die benötigen ROMs und Addierer lieferte National ehedem als Bauteilsatz [2]. Dessen Datenblatt enthält die Schaltung und einen Vorschlag, wie man die cos/sin–Umschaltung macht. Die Auflösung entsprach nicht exakt 12 Bit, kam aber sehr nahe.

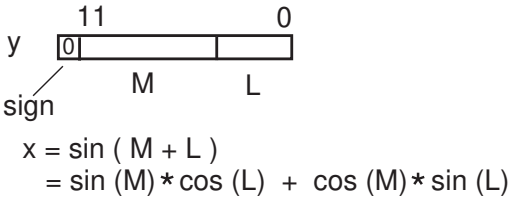


Abbildung 3: Formel

Diese algebraische Darstellung aus [1] (Abb. 3) ist allerdings wenig anschaulich. Intuitiver und auch universeller ist die Annahme, dass es ein Verfahren mit Grob– und Fein–Schritten ist. Die sind in der Elektronik wie auch Mechanik (Abb. 5) üblich.

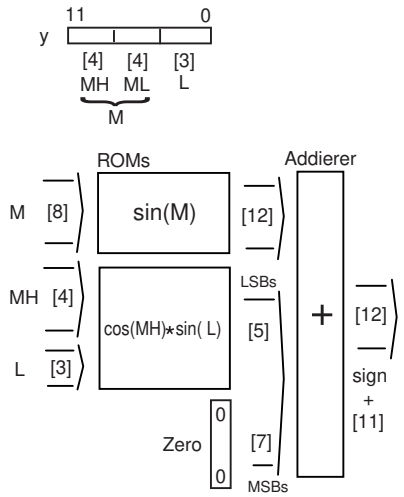


Abbildung 4: Implementierte Variante

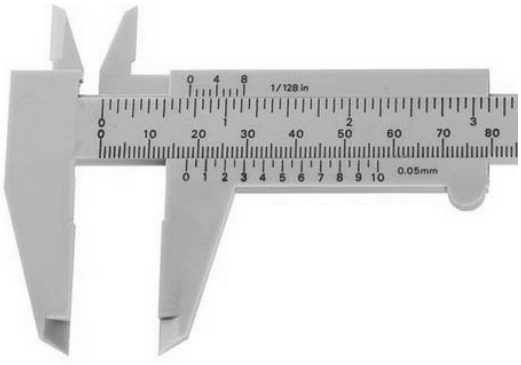


Abbildung 5: Schiebelehre

[1] MSI „Medium Scale Integration“, billige TTL–Logik. ROMs sind LSI „Large Scale Integration“, also teuer.

Implementierung

Als Hilfsmittel erzeugt man sich in FORTH zunächst einmal eine Sinus–Referenz mit einer höheren 15–Bit–Auflösung und 2048 Abtastpunkten. Aus dieser wird dann die reduzierte `SIN(M)`–Tabelle erzeugt. Die Differenz entspricht dem Fehler, geplottet ergibt das Abb. 6.

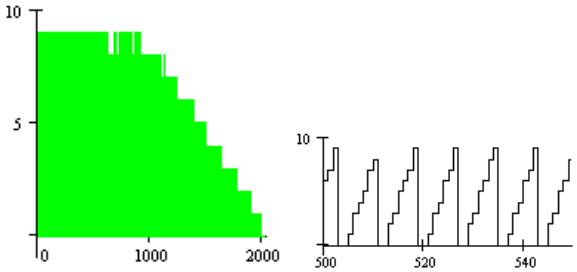


Abbildung 6: Restfehler `SIN(M)` und Feinstruktur

Der Fehler ist im unteren Bereich hoch, weil der Verlauf des Sinus dort steil ist. Die Werte liegen im Bereich `00`... `11` LSB[2], können also bereits mit 4 Bit korrigiert werden. Man sieht, dass die Feinstruktur mit Rampen von 8 Samples zyklisch wiederkehrt. Das ergibt sich daraus, dass die untersten 3 Bits ignoriert wurden. Die zweite Tabelle `COS*SIN-TAB` enthält nun diese untersten Bits für die Feinstruktur, aber auch die obersten 4 Bits. Mit diesen verschiebt sie ein 16–Samples–breites Fenster. Dafür fehlen ihr in der Mitte Bits. Dies bewirkt, dass nicht alle Werte einer Rampe exakt korrigiert werden können. Möglich ist aber als Näherung der Mittelwert von 16 Samples eines Fensters. Wie erwünscht ähnelt ein Plot dieser Tabelle (Abb. 7) dem Fehlersignal. Man könnte den Gesamtfehler (Abb. 8) durch Retuschieren der Tabellen absehbar noch verbessern. Darauf wird hier aber verzichtet, weil es die Darstellung unnötig verkompliziert hätte.

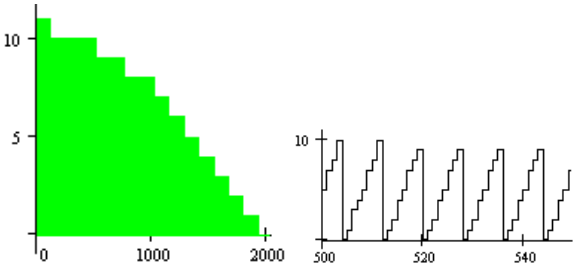


Abbildung 7: Inhalt `COS*SIN-TAB` und Feinstruktur

[2] least significant bit

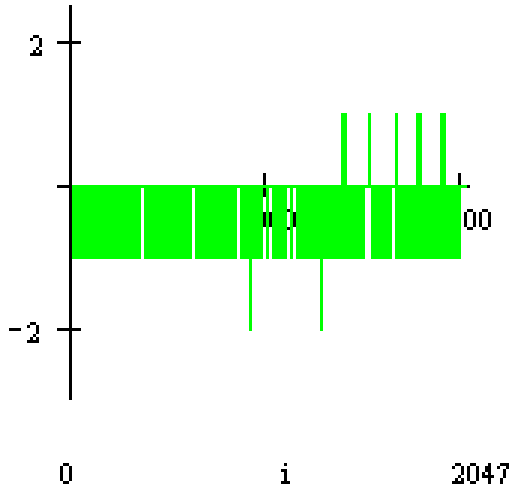


Abbildung 8: Gesamtfehler

Anwendung

Das Verfahren ist nicht auf den Sinus allein beschränkt. Viele Sensoren haben heute Controller (Abb. 9) integriert, die die Bauteilstreuung mit Korrekturwerten aus dem Flash kompensieren sollen. Für schnelle Sensoren muss das über Tabellen im Speicher erfolgen. Der Speicher sollte aber klein bleiben, damit man billige Controller verwenden kann.

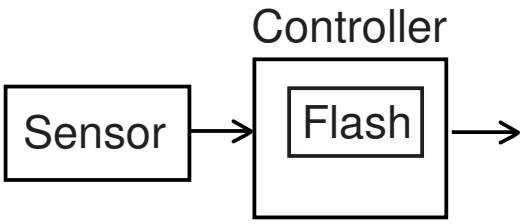


Abbildung 9: Sensorlinearisierung

Quellen

[1] MOS Brief 10, Trig Function Generators, National Semiconductor, MOS Integrated Circuits, 1972.

[2] SK0003, sine/cosine look–up table kit

Listing

```
1   \ ROM-SIN
2
3   DECIMAL
4   TABLE SIN-TAB
5   0 , 25 , 50 , 75 , 101 , 126 , 151 ,
6     ... 32767 , 32767 , 32767 ,
7   HEX
8
9   \ 15 Bit Reference
10  : SIN  ( UN1 --- UN2 )
11    \ UN1 = 0000 ... 07FF ; UN2 = 0000 ... 7FFF
12    1<SHIFT SIN-TAB + @ ;
13
14  \ ------------------
15
```

```
16  DECIMAL
17  TABLE SIN(M)-TAB
18  0 , 12 , 25 , 37 , 50 ,
19    ... 2047 , 2047 , 2047 , 2047 ,
20  HEX
21
22  : SIN(M)  ( UN1 --- UN2 )
23    \ UN1 = 0000 ... 07FF ; UN2 = 0000 ... 7FFF
24    1SHIFT> 1SHIFT> 1SHIFT> 1<SHIFT SIN(M)-TAB + @ ;
25
26  TABLE COS*SIN-TAB
27  00 C, 01 C, 03 C, ...  00 C, 00 C, 00 C,
28
29  : COS*SIN^  ( UN1 --- UN2 )
30    DUP B%         111 AND SWAP
31        B% 11110000000 AND 4SHIFT> OR
32    COS*SIN-TAB + C@ ;
33
34  : SIN"  ( UN1 --- UN2 )
35    \ 12 Bit ROM-SIN  approximation
36    DUP SIN^ SWAP COS*SIN^ + ;
37
38  \ ----------------------------------
39  \ make SIN(M)-TAB
40
41  : MAKE-ROM1  (  --- )
42    FF 00 DO
43    I 4<SHIFT SIN-TAB + @ 4SHIFT>
44    ND. ." , " \ print out
45    LOOP ;
46
47  \ ----------------------------------
48  \ make COS*SIN-TAB
49
50  $HEAP CONSTANT COS*SIN-TAB  \ 128 Byte RAM
51
52  : MAKE-ROM2   ( UN1 --- UN2 )
53    COS*SIN-TAB 80 00 FILL   \ erase RAM to 00
54    7FF 00 DO
55      I SIN 4SHIFT> I SIN^ -   \ calc error
56      I DUP B%         111 AND SWAP
57      B% 11110000000 AND 4SHIFT> OR
58      COS*SIN-TAB + +C!    \ accumulate error in RAM
59    LOOP
60    7F 00 DO                \ print out RAM
61    COS*SIN-TAB I + C@ 4SHIFT> CH. ." C, "
62    LOOP ;
63
64
```

Excurs zur interpolierten Sinus–Tabelle

Da ich zunächst rätselte, was Rafael Deliano da gezaubert hat, fragte ich Jens Storjohann, und er hat es mir, dem Mathe–Dummie, dankenswerterweise erklärt. Ihr hingegen wusstet das bestimmt alles schon.

Wenn man eine Funktion in einem Computersystem mehrfach berechnen will, gibt es zwei extreme Methoden. Die erste ist, ein Programm zu schreiben, das den Funktionswert für ein gerade geliefertes Argument stets neu berechnet und danach den berechneten Wert vergisst. Die andere extreme Methode besteht darin, vorab die Funktionswerte für alle möglicherweise aufrufbaren Argumente zu berechnen und in Tabellen zu speichern. Ersteres kostet Zeit, letzteres Speicherplatz.

Die Sinus– und die Cosinus–Funktionen erlauben einen Kompromiss dieser Methoden.

Es gilt:

$$\sin(x+y) = \sin(x)\cos(y) + \cos(x)\sin(y)$$
$$\cos(x+y) = \cos(x)\cos(y) - \sin(x)\sin(y)$$

Eine Anwendung in einem willkürlichen Zahlenbeispiel liefert:

$$\sin(0.11) = \sin(0.1 + 0.01) = \sin(0.10)\cos(0.01) + \cos(0.10)\sin(0.01)$$

Man sieht ein, dass man die Werte von `sin` und `cos` jeweils in einem *groben* und einem *feinen Raster* speichern muss.

Will man beispielsweise die Werte von `sin` und `cos` für Argumente von Null Grad bis 90 Grad in Schritten von 1 Grad haben, dann braucht man eine Funktionstabelle von `sin` und `cos` für die Argumente in groben Schritten:

`(0, 10, 20, 30, 40, 50, 60, 70, 80, 90).`

Und zusätzlich braucht man Werte in einem feinen Raster:

`(0, 1, 2, 3, 4, 5, 6, 7, 8, 9).`

Damit bekommt man bereits eine Auflösung von 1 Grad mit nur 10+10 = 20 Tabelleneinträgen, wenn man sich außerdem noch an seine Schulmathematik erinnert, wonach $\cos(x) = \sin(x+90)$ gilt, die Cosinus–Funktion also dem um 90 Grad verschobenen Sinus entspricht. Eine weitere Vereinfachung ergibt sich dadurch, dass für genügend kleine Werte, wie sie in Delianos Anwendung gegeben sind, der cos(y) davon 0,99998 . . . 1,0 ist. Die Näherungsformel reduziert sich dann zu $\sin(x+y) = \sin(x) + \cos(x)\sin(y)$.

In der Form ist das auch in der Abb. 4 seines Beitrages dargestellt.

Für die benutzten Formeln kann man *jede* Formelsammlung zu Rate ziehen. Im Internet gibt es eine umfangreiche Sammlung (PDF, 470 Seiten) aus dem Buch von Milton Abramowitz und Irene A. Stegun, Handbook of Mathematical Functions.

mk

FlexiFloat — ein Floating–Point für Forth

Jörg Völker

Das Projekt „FancyForth“ habe ich im Corona–Sommer 2020 gestartet. Es will Forth „entschärfen“. Denn Speicher und Rechenleistung sind heutzutage auch in kleinen MCUs reichlich vorhanden, sodass die Optimierung auf Geschwindigkeit und Speicherbedarf inzwischen oft unnötig ist. Der STM8 z. B. ist ein preiswerter 8–Bit–Prozessor, 10x schneller als die Z80– oder 6502–CPU es noch waren. Deshalb habe ich nun andere Entwicklungsziele: Programmierkomfort, Betriebssicherheit und Fehlertoleranz. Ein Aspekt davon wird hier näher beschrieben: die Arithmetik.

Ganzzahlarithmetik nervt!

Denn sie hat einen kleinen Werteumfang, einen Überlauf ohne Fehlermeldung, immer wieder diese lästige Notwendigkeit, zu skalieren, und braucht einen ganzen Zoo spezieller Operatoren.[1] Neben den gängigen Basics `= + - * / MOD` braucht man ja auch noch dieses ganze Gewusel hier: `*/ */MOD /MOD D+ D- S>D D>S M* M*/ M+ M- M/ UM/MOD UM*`

Die Projekterfahrung in meiner Firma[2] zeigte, dass sich Ganzzahlarithmetik gerade auf einem 16–Bit–Forth schwer pflegen lässt — ändert man in einem Algorithmus einen Faktor, muss regelmäßig die gesamte Arithmetik neu überprüft werden. Auf der einen Seite drohen Überläufe, bei denen auch noch schlagartig das Vorzeichen wechselt, auf der anderen Seite kann die Auflösung knapp werden. Doppeltgenaue Werte und insbesondere der Mix mit einfachgenauen Werten (mal vorzeichenbehaftet, mal vorzeichenlos) auf dem Stack machen die Sache dann erst richtig kompliziert. Forth behandelt ganze Zahlen wie eine Maschinensprache, es kümmert sich um nichts. Überläufe von vorzeichenlosen Werten, hier im vereinfachten 4–Bit–Zahlenkreis zwischen 15 und 0 führen genauso wenig zu einer Fehlermeldung wie Überläufe von Werten im Zweierkomplement, hier zwischen –8 und +7 (Abb. 1). Für manche Algorithmen ist das durchaus praktisch, in den allermeisten Fällen aber eher gefährlich und eine unangenehme Quelle für Fehler.

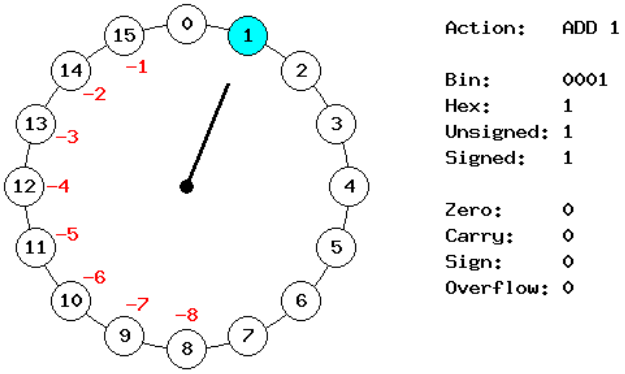


Abbildung 1: Zahlenkreis (Beispiel 4 Bit)

Damit wir uns nicht falsch verstehen: Das alles macht viel Sinn und Forth ist sehr gut durchdacht — aber aus der Sicht der Zeit um 1970! Mich quält diese Problematik schon lange und ich halte diese Form der Arithmetik schon länger schlicht für nicht mehr zeitgemäß — ich möchte mich damit einfach nicht mehr herumärgern. Zeit ist Geld und Fehlersuche macht keinen Spaß. Das muss anders gehen. Kann man diese Zahlenformate nicht irgendwie vereinheitlichen?

Alternativen?

Dabei gibt es zur Ganzzahl–Arithmetik durchaus Alternativen. Das hier sind die drei gängigen:

- Festpunkt–Arithmetik (32 Bit)
- Sättigungsarithmetik (16 oder 32 Bit)
- Floating–Point (32 Bit)

Über alle drei Varianten habe ich viel nachgedacht. Ein erster Schritt ist sicherlich der grundsätzliche Wechsel auf ein 32–Bit–Forth — ja, auch auf einem 8–Bit–Prozessor! Heutige Rechenleistungen machen das problemlos möglich. Bei der Festpunktarithmetik konnte ich mich allerdings nie entscheiden: Nimmt man 16.16 — oder doch das attraktivere 22.10–Format, bei dem 10 Bit genau 3 dezimalen Nachkommastellen entsprechen? Dann muss man entweder Ganz– und Festpunktzahlen auf dem Stack doch wieder auseinanderhalten oder bei der Adressierung schieben … Hmm.

Sättigungsarithmetik wird nicht ohne Grund bei vielen Signalprozessoren verwendet und wäre für meine Regel– und Steuerungsanwendungen in Kombination mit Festpunkt eigentlich gut geeignet. Der Wertebereich bleibt natürlich weiter eingeschränkt.

Aber Floating–Point? Der erste Gedanke: Das ist zu langsam!

Eigentlich hatte ich das deshalb erst gar nicht in Betracht gezogen. Bis ich dann testhalber verschiedene Varianten einfach mal codiert habe. Mit einem verblüffenden Ergebnis: Der STM8 kann 32–Bit–Werte zwar nicht effizient addieren, die 24–Bit–Mantisse eines 32–Bit–Floating–Point–Formats — das ist jetzt sehr prozessorspezifisch — aber schon! Da Addition und Subtraktion auch Bestandteile der Multiplikation und Division sind, ist das ein alles entscheidender Punkt.

[1] Rechnen mit ganzen Zahlen in Forth; Michael Kalus, „Vierte Dimension“, 35. Jahrgang, Heft 3/2019, S. 10–15

[2] tematik GmbH, in Wedel bei Hamburg

Na schön, dann eben Floating–Point — hab ich noch nie programmiert, aber warum eigentlich nicht ... Mit der derzeit forthtypischen Art, das auf einen Extra–Stack zu verlagern, kann ich mich allerdings mal wieder überhaupt nicht anfreunden. Zwei Stacks sind definitiv genug. Ich will vereinfachen, statt die Komplexität wieder zu steigern.

Bleibt ein Problem: Auf dem Stack wird viel mit Adressen gearbeitet. Adressen kann man im üblichen IEEE754 Floating–Point–Format zwar auch ablegen, aber das macht kein Mensch.[3] Auf diesen kleinen Prozessoren wäre das durch die nötige Schieberei auch entsetzlich ineffizient. Logische Werte und bitweise Verknüpfungen sind in dem Format gar nicht möglich. Also doch wieder zwei verschiedene Formate auf dem Stack, oder? Aber dann werde ich die ganzen F–Worte[4] nicht los (Abb. 2), und ich will doch vereinfachen.

```
F! F@ D>F F>D FDROP FDUP FOVER FROT FWAP
F+ F- F* F/ FABS FMAX FMIN FNEGATE F0<
F0= F< FVARIABLE
*/ */MOD /MOD D+ D- S>D D>S M* M*/ M+ M-
M/ UM/MOD UM* D+ D- D< D= D0= D>S S>D
DABS DNEGATE U. U< U> UM* UM/MOD
```

Abbildung 2: Der Zoo der Floating–Point–Arithmetik– und Double–Worte im Forth–Standard

FlexiFloating–Point, kurz: FlexiFloat

Schaut man sich das übliche IEEE754–Format genauer an, dann wird man den Verdacht nicht los, dass hier Vollblut–Mathematiker am Werk waren. Offensichtlich mit dem Ziel, aus 32 Bit das Maximum an Auflösung herauszukitzeln. Selbst aus meiner praktischen Sicht eher exotische Dinge wie *Infinity* und *NaN*[5] werden abgedeckt. Die Mantisse ist stets positiv und linksbündig normiert und — Trick — das höchstwertigste Bit deshalb immer eins, also wird es erst gar nicht gespeichert.

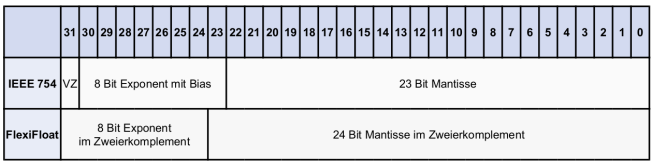


Abbildung 3: Unterschiede zum IEEE 754

Ich sehe aber keinen Vorteil darin, mich an diesen Standard zu halten. So entstand *FlexiFloat*, ein Format, das auf das letzte Quäntchen an Auflösung verzichtet, dafür aber unserem Forth auf den Leib geschneidert ist.

Dieses Flexi–Format ist praktisch eine Synthese aus Integer und Floating–Point. Ist das Exponent–Byte Null, entspricht die Mantisse einem 24–Bit–Ganzzahl–Wert im Zweierkomplement, für Adressierungen aller Art und ein wenig Kompatibilität zum Standard. `True` und `False`, alle Logik– und Schiebeoperationen funktionieren wie gewohnt, es fühlt sich halt an wie ein 24–Bit–Ganzzahl–Forth. Zahlenüberläufe oder Divisionen mit gebrochenen Ergebnissen wechseln nun aber automatisch in die Floating–Point–Darstellung mit einem Exponenten ungleich Null. Umfangreiche Fehlerüberprüfung wird plötzlich ganz einfach! Der Versuch, einen Floating–Point–Wert zur Adressierung zu verwenden, führt genauso zur Fehlermeldung wie Floating–Points bei logischen Verknüpfungen oder als Parameter für `DO...LOOP`. Der einfache, schnelle Test des Exponenten auf Null, mehr braucht es nicht.

Die Unterschiede zeigt Abb. 3 und die Tab. 1 und Tab. 2.

Wert	IEEE 754	FlexiFloat
1	3F800000	00000001
10	41200000	0000000A
–10	C1200000	00FFFFF6
100	42C80000	00000064
1.000	447A0000	000003E8
1.000.000	49742400	000F4240
+0	00000000	00000000
–0	80000000	00000000
Infinity	7F800000	—
–Infinity	FF800000	—
NaN	7FFFFFFF	—

Tabelle 2: Gegenüberstellung der Repräsentation typischer Werte

Kompatibilität

Die Vorteile liegen auf der Hand: Ein voll ausgebautes 16–Bit–Forth mit einfach– und doppeltgenauen Ganzzahlen und Floating–Point auf einem Extra–Stack braucht alleine an die 50 Worte, nur um diese unterschiedlichen Formate alle abzudecken und ineinander umzuwandeln (Abb. 2). Das ist extrem umständlich und fehlerträchtig und ganz nebenbei Forth–Interessenten und Neulingen im Jahr 2021 sicherlich auch nicht leicht zu vermitteln.

Verzichtet man auf die doppeltgenauen 32–Bit–Werte und gibt sich mit 24 Bit im FlexiFloat–Einheits–Format auf dem Stack zufrieden, braucht man diese 50 Worte nicht mehr. Der Wortschatz wird dadurch sehr übersichtlich: `+ - / * ABS MAX MIN NEGATE 0= > < =` funktionieren jetzt für ganze Zahlen und Floating–Point gleichermaßen. Das behaupte ich jedenfalls und muss zugeben, das ist ganz schön radikal. So ein radikaler Schritt hat sicher auch Nachteile, die Kompatibilität in der Mathematik zum Standard ist weitgehend dahin, die Ein– und Ausgabe belegt den Forth–Dot „.“ sinnvollerweise mit einer anderen Funktionalität usw. Trotzdem: Endlich funktioniert mein Forth so, wie ich es von meinem HP–Taschenrechner

[3] Fast kein Mensch, wie ich gelernt habe. Danke an Bernd Paysan für diesen Hinweis bei meinem Online–Vortrag: Es gibt GPUs, die das in der Hardware unterstützen.

[4] F wie Float. :)

[5] Not a Number

IEEE 754	FlexiFloat
Mantisse linksbündig normalisiert	Mantisse rechtsbündig nicht normalisiert
Mantisse immer positiv	Mantisse im Zweierkomplement
Exponent mit Offset	Exponent im Zweierkomplement
Ineffizient für Adressen	Pointer für Adressarithmetik geeignet
Keine logischen Werte	Kann True oder False sein
Optimiert auf Auflösung / Mathematische Anwendungen	Optimiert für Forth / Embedded Systems

Tabelle 1: Gegenüberstellung IEEE 754 vs. FlexiFloat

gewohnt bin, der hat schließlich auch nur eine Taste für „+“ und nicht drei!

Was ist nun so Flexi an diesem FlexiFloat?

Zum einen natürlich die Flexibilität, mit der sich das Format unterschiedlichen Ansprüchen nach schneller Adressarithmetik einerseits und präziser Floating–Point–Darstellung andererseits anpassen kann, ohne dass sich der arme Programmierer darum kümmern muss. Aber es gibt noch einen Trick: Die Anzahl der Nachkomma–Bits, die bei Division und Multiplikation berechnet werden, kann eingeschränkt werden. Bei der Division ergibt sich dadurch sogar ein Geschwindigkeitsgewinn. Das geht soweit, dass man sowohl beim Aufruf als auch global die Anzahl der Nachkomma–Bits auf `0` setzen kann — mit dem Ergebnis, dass sich das FancyForth in ein (fast) reines 24–Bit–Ganzzahlsystem verwandelt. Das ist so ein wenig wie eine einstellbare Festpunktarithmetik.

ACCURACY Dieser Value bestimmt bei `*` und `/` global die Anzahl der Nachkommabits.

FLEXI/ `( r1 r2 acc -- rem quo )` entspricht dem `/MOD` bei acc=0.

FLEXI* `( r1 r2 acc -- r3 )` entspricht bei acc=0 einem `*` wie bei Ganzzahlarithmetik, bei acc=127 voller Auflösung.

Kein Vorteil ohne Nachteil

Dieses Floating–Point–Paket wurde speziell für den STM8 vollständig in Assembler geschrieben, und damit sind wir bei einem weiteren Nachteil, über den man offen sprechen muss: Die traditionellen Forth–Varianten, die ich kenne, versuchen immer mit einem Minimum an prozessorspezifischen Assembler–Teilen auszukommen[6], auch auf Kosten der Geschwindigkeit. Deshalb ist traditionelles Forth so portabel, aber eben nicht schnell, wie oft fälschlicherweise behauptet wird. Ein Floating–Point in Forth zu schreiben, führt daher in der Regel zu einer miserablen Performance, wenn man nicht gerade einen Forth–Prozessor besitzt. Dieses STM8–FlexiFloat–Paket ist so STM8–spezifisch und auf Direct–Threaded–Code angepasst, dass es schon eine Herausforderung ist, es auf andere Systeme zu portieren. Speziell auf 32–Bit–Maschinen stellt sich sofort die Frage, ob 24 Bit für Adressarithmetik überhaupt ausreicht — beim Coldfire V1 ist das der Fall, das dürfte aber die Ausnahme sein. Allerdings könnte man das Grundprinzip ja auch auf 48 oder 64 Bit umsetzen.

[6] Gerne auch „primitives“ genannt.

Performance

Nun habe ich gleich am Anfang argumentiert, Geschwindigkeit sei nicht so wichtig. Beim Schreiben der ersten Arithmetik–Routinen sind dann aber gleich die Pferde mit mir durchgegangen und ich habe doch wieder angefangen, zu optimieren, das hat halt Spaß gemacht, ich tüftel gerne in Assembler herum. Die Erkenntnisse lassen sich vielleicht auch auf andere Systeme übertragen. So wird für die Division erst eine umfangreiche Fallunterscheidung durchgeführt — wann immer möglich kommt zunächst eine 16/16–Division als eigener Prozessor–Befehl zum Einsatz und erst die Nachkommastellen werden mit dem üblichen Schieben/Subtrahieren bestimmt. Bei der Multiplikation werden ggf. die Faktoren getauscht, dann im ersten Schritt die 8x8–Multiplikation des Prozessors verwendet und danach erst auf Schieben/Addieren gewechselt, um das Ergebnis in einem Arbeitsgang in die Mantisse einzupassen. Die ganze Vorzeichenverwaltung findet ebenfalls schon auf der Assembler–Ebene statt, das ist in Forth zu ineffizient.

Ein Beispiel

Die einfache Sinus–Approximation mit einer Taylor–Reihe (s. Listing) benutzt alle Grundrechenarten und kommt auf etwa 38 kFlops. Je nach Sichtweise ist das viel oder wenig, für ein 8–Bit–System finde ich es jedenfalls nicht schlecht.

Ausblick

FlexiFloat ist nur eine Komponente in meinem Bemühen, Forth immer weiter zu entschärfen und zu entgiften. Dazu gehört fast untrennbar auch ein minimalistisches *OOP*, um Variablenzugriffe zu vereinheitlichen, und seit neuestem auch eine Laufzeit–Überwachung des Return–Stacks über „Tags“, um dort Fehler zu erkennen und Abstürze möglichst abzufangen. Aber das ist ein anderer Bericht.

Links

`https://www.servonaut.de`

Listing

```
1   \ Taylor Reihe
2
3   : sin ( n -- n )
4     dup dup * >r          \ x
5     dup r@ *              \ x^3
6     dup r@ *              \ x^5
7     dup r@ *              \ x^7
8     dup r> *              \ x^9
9        362880 /
10    swap 5040 / -
11    swap  120 / +
12    swap    6 / - + ;
13
14    3.141593 constant pi  OK 0
15    pi 2 / sin . 1.0000035  OK 0
16  \ Mit der Abweichung kann ich leben. :)
17
```

Nachtrag

Warum STM8, warum nicht gleich ein ARM oder vergleichbarer 32–Bit–Prozessor? Die Frage ist berechtigt. Das hat alles, wie so oft, historische Gründe. Wir feiern gerade 20 Jahre Servonaut, der Markenname, unter dem wir Modellbauelektronik vermarkten. Seit Anfang an ist Forth dabei, unser erstes Produkt hatte noch einen 68HC11 und war in *Holon11* programmiert. Dann kam der Bedarf für einen kleineren und billigeren Chip und der Wechsel zum 68HC08. Für den konnte ich kein mich überzeugendes Forth finden, also habe ich erstmals eine Version selbst geschrieben.[7] Auf den HC08 folgte der S08, aber diese Chips wurden immer teurer. 2017 erfolgte dann der Wechsel bei Neuentwicklungen hin zum STM8. Der Prozessor war von der Struktur fast identisch, die Anpassung von unserem hausinternen Forth deshalb einfach, und dabei waren die Chips auch noch deutlich schneller und nur halb so teuer!

Das war keine glückliche Entscheidung, wie sich leider heute herausstellt. Wir stecken mitten in der Chip–Krise, die natürlich nicht nur die Automobil–Industrie betrifft, sondern auch jedes KMU[8], das irgendwie mit Elektronik zu tun hat. Wir verarbeiten rund 800 Controller jeden Monat. Ein existenzielles Problem. Alles ist auf den Kopf gestellt. Zum Glück haben wir derzeit, ebenfalls durch die historische Entwicklung bedingt, verschiedene Prozessorfamilien im Einsatz. Neben dem S08 von *NXP* (ehemals Freescale, davor Motorola) ist das noch der *Coldfire* 32 Bit, der STM8 von *ST* und in einigen Fällen auch noch AVR von *Microchip* (ehemals Atmel).

Die verschiedenen Hersteller und Prozessorfamilien sind ganz unterschiedlich von der Krise betroffen. Während bei ST die Liefersituation bei allen Controllern schon seit einem halben Jahr katastrophal ist (alles steht auf Lead Time 53 Wochen), ist die Lage beim S08 und Coldfire etwas entspannter. Katastrophal heißt: Ware ist praktisch nur noch bei (i. d. R. chinesischen) Chip–Brokern zu bekommen, zu astronomischen Preisen zwischen Faktor 4 und Faktor 20. Entspannter heißt: Man findet ab und zu Bestände bei seriösen Distributoren zum im Schnitt doppelten Preis im Vergleich zum Vorjahr. Um die Lage noch etwas zu verschärfen, hat ST jetzt die zwei STM8–Unterfamilien STM8AL und STM8AF einfach mal komplett abgekündigt. Das macht für ST in dieser Situation absolut Sinn, denn das Lieferprogramm war und ist überladen und unübersichtlich genug. Aber werden das die Anwender auch so sehen? Bei *TI* sieht es da ganz anders aus, zwar ist auch da nicht alles lieferbar, TI bietet aber immer noch Ware über den eigenen Shop an, zu verträglichen Preisen. So wird der Spekulation effektiv die Grundlage entzogen. Ich jedenfalls werde keine ST–Produkte mehr in Neuentwicklungen einsetzen. Das konnte ich vor zwei Jahren, als *FancyForth* für den STM8 gestartet wurde, aber nicht ahnen …

Einer schöner als der andere. LKW–Modelle im Maßstab 1:14. Nicht wenige davon sind mit Elektronik aus Wedel ausgestattet und tragen Forth–Systeme mit sich herum.

(Farbig noch schöner: `https://wiki.forth-ev.de/doku.php/vd-archiv`)

[7] J. Völker, „T4 Embedded–Forth–Experiment“; Forthmagazin „Vierte Dimension“, Heft 4d2007–02, S. 26–30.

[8] Kleines oder mittleres Unternehmen.

Mandalas oder: Verrückte Rotationen auf Ganzzahlen

Matthias Koch

In dem Artikel „Angenehmes Blinken II“ wurde ein Algorithmus vorgestellt, der Kreise zeichnet und dabei ein wenig eiert. Das Interessante daran ist, dass jedem Punkt eindeutig ein Nachfolger und ein Vorgänger zugeordnet wird, und die Kurven nach mehr oder weniger vielen Schritten geschlossen sind. Das bedeutet, dass jeder Punkt in der Ebene zu genau einem „Zyklus“ gehört, und mit der Iterationsvorschrift alle anderen Punkte dieses Zyklus gefunden werden können. Aber nicht nur Kreise können so entstehen!

Was anderes als Kreise malen

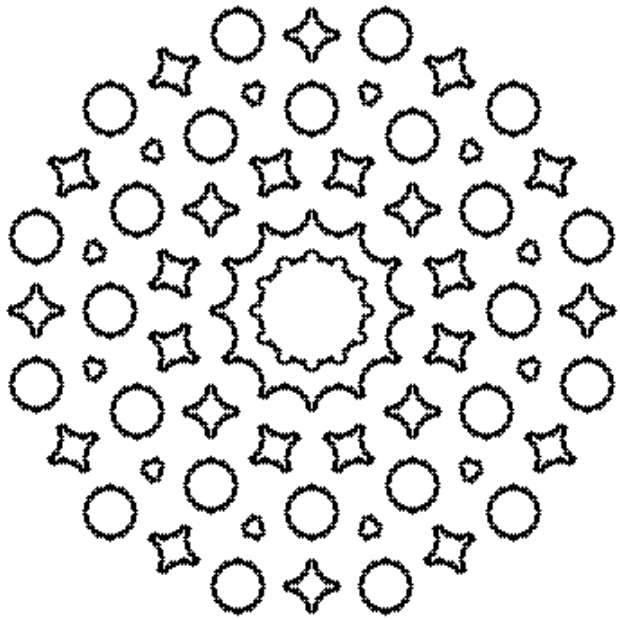


Abbildung 1: Mandala mit einem Drehwinkel von 1/12 des Vollkreises. Zu einem Zyklus gehören jeweils 12 Stellen pro Umdrehung, die nacheinander jeweils einen weiteren Pixel bekommen, zum Beispiel die 12 Kringel im äußersten Ring. Die äußeren 12 Sterne gehören zu einem anderen Zyklus, hängen aber ebenfalls zusammen.

Werden die Parameter so gewählt, dass der Iterationsschritt eine Drehung um einen bestimmten Winkel ausführt, dann gibt es ganz andere Muster. Sind die Winkel 0 Grad oder 90 Grad, ist es gleich klar: Bei Null Grad wird nicht gedreht, da wäre der Ausgangspunkt ja sein eigener Nachfolger. Und bei 90 Grad entstehen Quadrate, alle Zyklen (wenn sie nicht gerade vom Ursprung ausgehen) haben also die Länge 4.

Doch was passiert, wenn um Winkel gedreht wird, die eben nicht in das Ganzzahl–X–Y–Gitter passen, aber trotzdem geschlossene Zyklen vorliegen, die Drehungen eindeutig umkehrbar sind und alle Koordinaten ganzzahlig bleiben? Dann passieren interessante Dinge, die bisweilen wie *Mandalas* aussehen.[1] Und um genau diese soll es jetzt gehen!

Iterationsvorschrift

Zunächst einmal die Iterationsvorschrift:

$a = \frac{cos(\theta)-1}{sin(\theta)}$

$b = sin(\theta)$

und:

$(x, y) \rightarrow (x'', y')$

$x' = x + round(a * y)$

$y' = y + round(b * x')$

$x'' = x' + round(a * y')$

Ohne die Rundung würde es sich bei dieser Iterationsvorschrift einfach um eine Drehung des Punktes um den Winkel θ handeln, doch mit der Rundung beginnt es interessant zu werden: Der Punkt wird so gut es geht um den gewünschten Winkel gedreht und landet dabei auf einer ganzzahligen Koordinate, die so in etwa der Drehung um den Winkel θ entspricht. Ganz besonders hängt das Ergebnis im Detail davon ab, wie sich diese Rundung der Zahl verhält — ich habe sie so implementiert, dass 0,5 addiert wird und dann die Nachkommastellen abgeschnitten werden. Das führt zu besonders schönen Ergebnissen, die in diesem Artikel gezeigt werden. Wer neugierig ist, mag jedoch auch andere Rundungsvarianten erproben. Ich gebe allerdings zu, dass ich noch nicht wirklich weiß, wie genau die verwendete Rundungsvorschrift die entstehenden Zyklen beeinflusst.

Erstaunlich lange Zyklen

Gehen wir nun alle Punkte in der Ebene durch und untersuchen die Länge der Zyklen, dann sind es sehr viele kurze Zyklen, die einfach nur den Punkt herumdrehen und dann nach einigen wenigen Umdrehungen wieder am Ausgangspunkt ankommen. Es tauchen aber überraschenderweise auch einige sehr lange Zyklen auf, und die sind wirklich interessant geformt, wobei die allerschönsten

[1] Die Mandala–Bilder hier im Heft sind extra für den Druck auf Papier in schwarz und fett geplottet worden. Wenn ihr das am Bildschirm probiert, werden sie farbig auf schwarzem Hintergrund sein. Lasst euch überraschen wie schön die sind! :)

Muster entstehen, wenn die Drehung um 1/3, 1/6, 1/8 oder 1/12 des Vollkreises gewählt wird.

Praktisches Beispiel

Der hier abgedruckte Quelltext ist für *Gforth* gedacht und schreibt das Ergebnis als *PNM–Bild*. Dieses Format, die „portable any map“, ist ein ganz simples Bitmap–Grafikformat, welches zwar nicht sehr gebräuchlich ist, aber dafür in wenigen Zeilen Quelltext implementiert werden kann und sich somit prima als Austauschformat eignet. Die Bitmap so einer PNM–Datei ist mit Forth leicht im RAM zu erzeugen und kann dann mit einem passenden Header davor direkt in eine Datei übertragen werden. Diese Grafikdatei wiederum kann dann, zum Beispiel mit den Werkzeugen *Imagemagick,* in ein beliebteres Format umgewandelt werden — schon steht Experimenten mit Grafik in Forth nichts mehr im Wege. In Linux braucht es nur diese zwei Zeilen, um das schönste Mandala hinzuzaubern:

```
gforth Mandala.fs
convert Mandala.pnm Mandala.png
```

Fließkomma wird in dem Beispiel übrigens nur für die Bestimmung der Konstanten `a` und `b` verwendet, alles andere geschieht in Ganzzahlen und kann somit auch für Grafikdemos auf Mikrocontrollern verwendet werden, wenn die Konstanten vorab berechnet und in den Quelltext eingefügt werden.

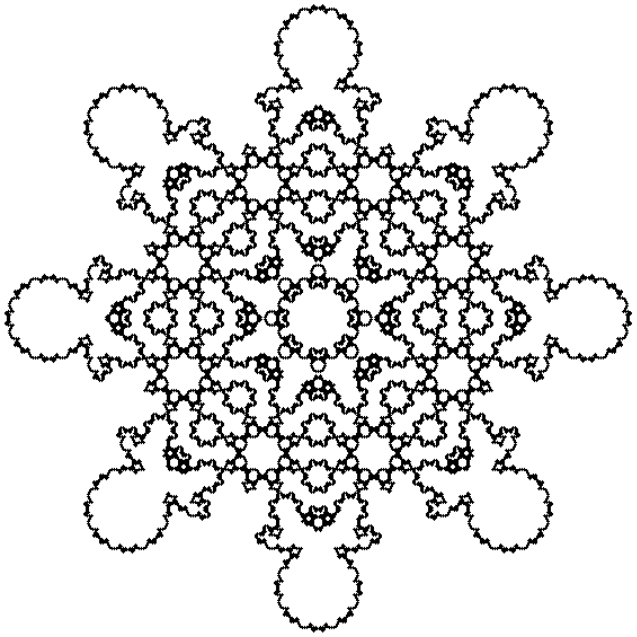


Abbildung 2: Mandala mit einem Drehwinkel von 1/8 des Vollkreises. Es treten sehr kompliziert geformte Zyklen auf, die teilweise nahe beieinander liegen und deshalb in diesem fett gedruckten Schwarzweißbild nicht mehr unterschieden werden können. Haben sich hier etwa Lebkuchenmännchen versteckt?

Exkurs zu Fixkomma–Zahlen

Für diejenigen, die noch nie „Ganzzahlen mit Nachkommastellen“ verwendet haben, dazu noch eine kleine Notiz:

Die Idee ist, eine normale Zahl mit Nachkommastellen mit einer Zweierpotenz zu multiplizieren, so dass einige Nachkommastellen in den unteren Bits der Zahl plaziert werden können. Falls wir 4 Bits Nachkommastellen in einer insgesamt 8 Bit breiten Zahl haben möchten, so würde die Zahl `3` als `%00110000` dargestellt werden, und $0,5 = 2^{-1}$ würde zu `%00001000` werden. Analog $1,25 = 2^0 + 2^{-2}$ = `%00010100`. Der kleinste Wert `%00000001` wäre $2^{-4} = 0,0625$. Alles klar?

Zuerst muss also entschieden werden, wie viele Nachkommastellenbits benötigt werden. `a` sei eine „normale“ reelle Zahl mit Nachkommastellen, `f` sei die Fixkomma–Schubweite (die eben genannte Zweierpotenz zum Multiplizieren der Nachkommastellen in den Ganzzahlbereich), dann ist $A * f \approx a$, wobei `A` eine Ganzzahl ist.

Addition und Subtraktion

Bei der Addition und Subtraktion passt alles gut, diese Fixkomma–Zahlen lassen sich wie normale Ganzzahlen addieren und subtrahieren, wobei auch das Zweierkomplement wie üblich funktioniert. Es gilt:

$$a \pm b = A * f \pm B * f = (A \pm B) * f$$

Multiplikation

Für die Multiplikation ist aber zu beachten, dass das Ergebnis doppelt so lang sein wird wie die beiden Fixkomma–Faktoren, daher ist auf eine ausreichend Bit–Breite für das Ergebnis zu achten! Anschließend muss das Ergebnis noch einmal durch `f` geteilt werden, was elegant durch einen arithmetischen Schub nach rechts um die gewählte Zahl der Nachkommastellenbits erledigt wird.

$$a * b = A * f * B * f = A * B * f^2$$

Division

Bei der Division schließlich würde die direkte Division der beiden Fixkommazahlen die Nachkommastellen eliminieren.

$$\frac{a}{b} = \frac{A*f}{B*f} = \frac{A}{B}$$

Das kann nützlich sein, aber wenn Nachkommastellen im Ergebnis gewünscht sind, ist es also nötig, vorher den Dividenden um die gewünschte Zahl an Nachkommastellenbits nach links zu schieben, damit das Ergebnis Nachkommastellen behält.

$$f * \frac{a}{b} = \frac{A*f^2}{B*f} = f * \frac{A}{B}$$

Praktische Verwendung

Damit stehen all die Grundrechenarbeiten zur Verfügung. Es ist ein bisschen Mühe, das für längere Rechnungen auszuknobeln, aber im Endeffekt definiert man sich einmal

eine Konstante wie `fractionalbits`, dröselt die Rechnung auf Papier auf und schiebt immer mal wieder an strategischen Stellen. Das ist der ganze Zauber.

Werden nun nur wenige Nachkommastellen benötigt, kann es gut sein, dass die gesamten Rechnungen in die Zellenbreite eures Forth–Systems passen und gar keine doppeltlangen Zwischenergebnisse nötig sind. Trick im Kopf behalten, wenn es mal rasant gehen muss!

Ausgabeformat

Die Ausgabe solcher Zahlen funktioniert so, dass zuerst alle Nachkommastellen durch einen arithmetischen Schub nach rechts abgetrennt werden und so der Teil vor dem Komma ganz normal ausgegeben werden kann.

Anschließend werden die Nachkommastellenbits maskiert und die einzelnen Stellen werden durch eine Multiplikation mit der aktuellen Zahlenbasis gewonnen, die die jeweils nächste Nachkommastelle in den „Ganzzahlbereich“ befördert, sodass auch der Nachkommateil „normal“ gedruckt werden kann.

Listing

```
1
2   \ Mandalas für Gforth
3   \ Matthias Koch, Dezember 2021
4
5   \ ------------------------------------------------------------
6   \   Grafikspeicher und Zeichenroutinen definieren
7   \ ------------------------------------------------------------
8
9   800 constant xres 800 constant yres
10
11  create grafikspeicher xres yres * 3 * allot align
12  grafikspeicher xres yres * 3 * erase
13
14  : putpixel ( x y Farbe -- )
15    >r xres * + 3 * grafikspeicher +
16    r@     $FF and           over     c!
17    r@   $FF00 and  8 rshift over 1 + c!
18    r> $FF0000 and 16 rshift swap 2 + c!
19  ;
20
21  : getpixel ( x y -- Farbe )
22    xres * + 3 * grafikspeicher +
23    dup     c@           >r
24    dup 1 + c@  8 lshift >r
25        2 + c@ 16 lshift r> or r> or
26  ;
27
28  \ ------------------------------------------------------------
29  \   Zeichnen relativ zum Mittelpunkt des Bildes
30  \ ------------------------------------------------------------
31
32  : putpixel-offset ( x y Farbe -- )
33    >r swap xres 2/ +
34       swap yres 2/ + r> putpixel
35  ;
36
37  : getpixel-offset ( x y -- Farbe )
38    swap xres 2/ +
39    swap yres 2/ + getpixel
40  ;
41
42  \ ------------------------------------------------------------
43  \   Parameter vorbereiten
44  \ ------------------------------------------------------------
45
46  \         v Ausprobieren: 3, 6, 8, 12.
47  pi 2e0 f* 8e0 f/ fconstant theta
48
49  theta fcos 1e0 f- theta fsin f/ fconstant afloat
50  theta fsin                      fconstant bfloat
51
52  16 constant fractionalbits
53  1 fractionalbits 1- lshift constant rounding
54
55  1 fractionalbits lshift s>f afloat f* fround f>s constant aint
56  1 fractionalbits lshift s>f bfloat f* fround f>s constant bint
57
```

```
\ ------------------------------------------------------------
\   Mandala zeichnen
\ ------------------------------------------------------------

: arshift ( x u -- ) 0 ?do 2/ loop ;

: zykluspunkt ( x y -- x'' y' )
  swap over ( y   x   y  ) aint * rounding + fractionalbits arshift + ( y  x' )
  swap over ( x'  y   x' ) bint * rounding + fractionalbits arshift + ( x' y' )
  swap over ( y'  x'  y' ) aint * rounding + fractionalbits arshift + ( y' x'' )
  swap ( x'' y' )
;

variable farbe

: zyklusmaler ( x-start y-start -- ) \ Zeichnet einen Zyklus

  2dup 2>r \ Startpunkt zur Enderkennung auf den Returnstack

  begin
    zykluspunkt
    2dup farbe @ putpixel-offset
    2dup 2r@ d=
  until

  2drop 2rdrop
;

variable laenge

: zykluslaenge ( x-start y-start -- laenge ) \ Bestimmt die Länge eines Zyklus

  0 laenge !

  2dup 2>r \ Startpunkt zur Enderkennung auf den Returnstack

  begin
    zykluspunkt
    1 laenge +!
    2dup 2r@ d=
  until

  2drop 2rdrop
  laenge @
;

$000000 constant hintergrund \ BBGGRR

: clrscr ( -- )
  xres 0 do yres 0 do j i hintergrund putpixel loop loop
;

require random.fs
31 seed ! \ Immer den gleichen Anfangswert für die Farben

: mandala ( -- )
   clrscr
   129 -128 do
     129 -128 do
      j i getpixel-offset hintergrund =  \ Pixel noch nicht gefärbt ?
      if
        j i zykluslaenge 128 >=  \ Zyklus lang und interessant ?
        if
          rnd farbe ! \ Neue Farbe aussuchen
          j i zyklusmaler
          ." Zyklus: " j 5 .r i 5 .r ."   Länge: " laenge @ 5 .r cr
        then
      then
    loop
  loop
;

mandala

\ ------------------------------------------------------------
\   Ausgabe des gezeichneten Bildes als PNM-Grafik
```

```
\ ------------------------------------------------------------

s" Mandala.pnm" w/o create-file throw value fd-grafik

create \f 13 c, align

s" P6"            fd-grafik write-file throw
\f 1              fd-grafik write-file throw
xres s>d <# #S #> fd-grafik write-file throw
s"  "             fd-grafik write-file throw
yres s>d <# #S #> fd-grafik write-file throw
\f 1              fd-grafik write-file throw
s" 255"           fd-grafik write-file throw
\f 1              fd-grafik write-file throw

grafikspeicher xres yres * 3 * fd-grafik write-file throw

fd-grafik close-file throw
bye
```

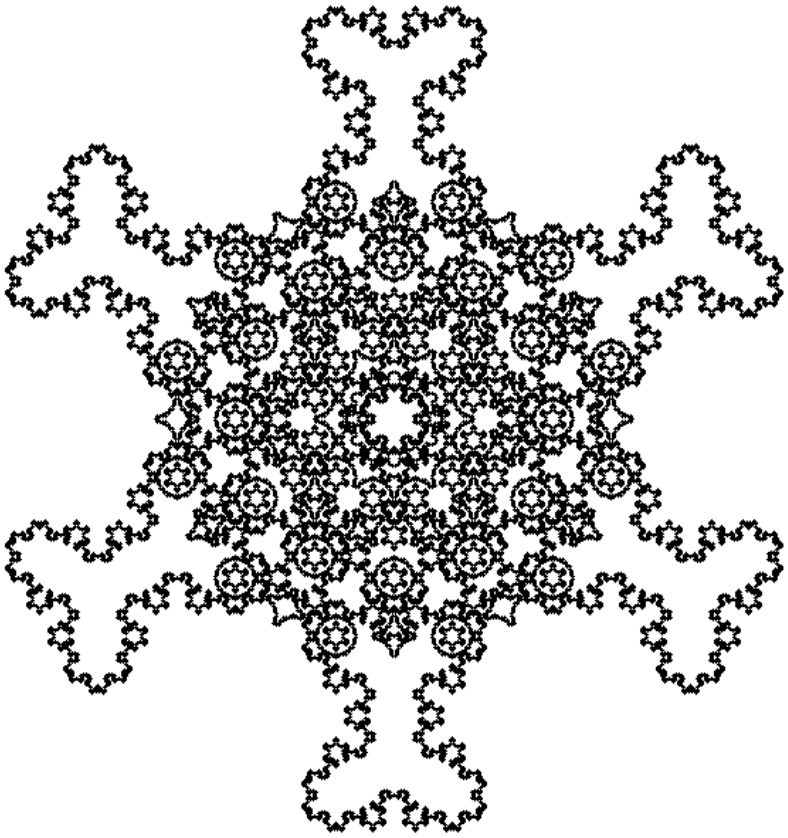


Abbildung 3: Mandala mit einem Drehwinkel von 1/3 des Vollkreises. Die nach außen abstehenden Strukturen bestehen aus zwei Zyklen, von denen jeder Zyklus jeweils drei zeichnet. Genauso ist es mit den 12 „Wagenrädern“ im Inneren, von denen jeweils drei zum gleichen Zyklus gehören und die so von vier separaten Zyklen gezeichnet werden. Alle Mandalas setzen sich übrigens noch viel weiter fort — wer neugierig ist, möge einfach größere Bilder mit Zyklen zeichnen, die weiter vom Koordinatenursprung entfernt liegen.

Forth–Gesellschaft e.V. — Ordentliche Mitgliederversammlung 14.11.2021

Wolfgang Strauß

Versammlungsleiter: Anton Ertl

Protokollant: Wolfgang Strauß

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

Sitzungsdatum: Sonntag, 14.11.2021

Sitzungsbeginn: 9:15 Uhr

Sitzungsende: 12:05 Uhr

Sitzungsort: pandemiebedingt online per Videokonferenz

Begrüßung der anwesenden Mitglieder

Ulrich Hoffmann 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 92 Mitglieder, von denen 13 anwesend sind. Damit ist die Versammlung beschlussfähig (mehr als 10 Prozent der Mitglieder sind anwesend).

Ergänzungen zur Tagesordnung

Carsten Strotmann spricht das Thema „Besetzung des Forth–Büros“ an. Ewald Rieger möchte das Amt abgeben und so wird ein neuer Kandidat gesucht. Carsten Strotmann stellt sich zur Verfügung, wenn jemand seine Stelle im Direktorium übernimmt. Für die Wahrnehmung beider Ämter fehlt ihm die Zeit.

Ulrich Hoffmann dankt Carsten Strotmann für seinen Vorschlag und Ewald Rieger für seine jahrzehntelange, vorbildliche Arbeit in der Verwaltung.

Alternativ stellt Carsten Strotmann die Aufteilung der Aufgaben der Verwaltung zur Diskussion, um Interessenten, denen das komplette Amt zu umfangreich ist, die Kandidatur zu erleichtern. Als Aufgaben werden die Kassenführung, die Mitgliederführung sowie der Versand der Vereinszeitschrift genannt.

Ewald Rieger meldet sich zu Wort. Er ist gegen eine Aufteilung, da Schwierigkeiten zu erwarten sind.

Ewald Rieger erläutert im Einzelnen die Aufgaben:

- Versand der Vereinszeitschrift VD
- Mitglieder pflegen
- Steueranmeldung
- einmal pro Monat Bankkonto prüfen
- Lastschriften
- Säumige Zahler

Ewald Rieger erklärt, dass bei der Änderung der Adresse der Verwaltung auch das Finanzamt wechseln kann, dieses aber nicht zwangsläufig der Fall sein muss.

Gerald Wodni bewirbt sich für einen Posten im Direktorium. Er gibt zu bedenken, dass er österreichischer Staatsbürger ist und fragt, ob dies einen Hinderungsgrund darstellt. Bernd Paysan erklärt, dass eine europäische Staatsbürgerschaft ausreicht. Der Sitz des Vereins (Verwaltung) sollte aber in Deutschland sein.

Die anwesenden Mitglieder wählen Carsten Strotmann einstimmig mit einer Enthaltung in die Verwaltung des Vereins. Carsten Strotmann nimmt die Wahl an und übernimmt zum 1.1.2022 das Amt.

Bericht des Direktoriums

Bericht der Verwaltung (Ewald Rieger)

Pandemiebedingt fand 2020 keine ordentliche Mitgliederversammlung statt. Deshalb im Folgenden die Daten der Wirtschaftsjahre 2019 und 2020.

Mitgliederentwicklung

Im Jahr 2019 gab es zwei Neuzugänge und vier Austritte (zwei davon durch Ableben des Mitglieds). 2020 ist ein Mitglied dem Verein beigetreten und zwei sind ausgetreten. In 2021 konnten bis dato drei Mitglieder gewonnen werden und zwei Mitglieder haben den Verein verlassen.

Momentan hat der Verein 92 Mitglieder.

Finanzen

Ewald Rieger erläutert im Detail die Einnahmen und Ausgaben, getrennt nach Verein und Zweckbetrieb (Vereinszeitschrift Vierte Dimension). Zum Ende des Wirtschaftsjahres 2019 ergibt sich ein Vermögen von

7.458,34 EUR und zum Ende des Wirtschaftsjahres 2020 ein Vermögen von 8.065,49 EUR.

Das Vereinsvermögen ist in den Jahren 2019 und 2020 um insgesamt 853,36 EUR gewachsen.

Bericht des Kassenprüfers

Thomas Prinz hat die Kasse des Jahres 2019 am 5.10.2021 und die des Jahres 2020 am 2.11.2021 geprüft. Friedel Amend war als Zeuge anwesend. Thomas Prinz 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 hält die Papierversion trotz des Kostenfaktors für Druck und Versand für unverzichtbar. Er beschreibt das Gefühl, eine physikalische Version der Zeitschrift im Briefkasten zu finden und sich auf das gemütliche Lesen zu freuen. Ulrich Hoffmann dankt Michael Kalus für seine unermüdliche Arbeit an dem Magazin. Seine Artikelsuche und Animation potentieller Autoren zum Verfassen von Artikeln sorgen dafür, dass das Heft immer mit interessantem Inhalt gefüllt ist.

Die Vierte Dimension enthält regelmäßig Artikel in englischer Sprache. Damit ist sie auch für ein internationales Publikum interessant.

Für den Fortbestand der Zeitschrift werden neue Artikel benötigt. Michael Kalus appelliert an die Vortragenden der Tagung, ihre Präsentation als Artikel aufzubereiten und für das Heft zur Verfügung zu stellen.

Ulrich Hoffmann spricht die mangelnde Druckqualität der letzten Ausgaben an. Der Text ist zu hell und damit schwer lesbar. Ein Gespräch mit dem Druckhaus soll den Mangel für die Zukunft beseitigen.

Internet–Präsenz (Ulrich Hoffmann)

Die Technik der neuen Website stammt von Gerald Wodni und funktioniert gut. Gerald erklärt, er möchte das Einstellen von Meldungen einfacher machen, um Mitgliedern die Mitarbeit zu erleichtern. Die Vereins–Website muss mit mehr Leben gefüllt werden, um auf sich aufmerksam zu machen. Michael Kalus wird als fleißiger Poster genannt.

Gerald Wodni erwähnt den YouTube–Kanal des Vereins. Mitglieder sollen auch dort Beiträge einstellen.

Ulrich Hoffmann fragt, ob man den Server mit den alten Seiten dann abschalten kann. Bernd Paysan merkt an, dass vor dem Abschalten noch die E–Mails umgezogen werden müssen.

Außendarstellung und Projekte (Bernd Paysan)

Bernd Paysan berichtet vom Besuch des 36C3 Ende 2019 und seinem Vortrag „CloudCalypse 2: Social network with net2o“. 2020 fand der Kongress online als rC3 statt. Die Veranstaltung litt unter Problemen mit der Videokonferenzsoftware Jitsi.

Carsten Strotmann spricht über seinen Besuch auf dem Vintage Computer Festival Berlin. Er hat dort Rechner ausgestellt und Collapse OS gezeigt, ein auf Forth basierendes Betriebssystem. Ein Artikel für die VD folgt.

Gerald Wodni erwähnt, dass die Benutzung der Streamingplattform Twitch mit dem eigenen Namen Pluspunkte vom Algorithmus gibt. Unsere Tagung hat dort 54 Zuschauer, er ermuntert zu Mitmachen. Das erhöht die Sichtbarkeit im Internet. Das Gleiche gilt für YouTube. Gerald Wodni empfiehlt für die Kommunikation untereinander den Dienst Mattermost. Es wird überlegt, auf welchem Server in Zukunft Videokonferenzen gehostet werden können. Anton Ertl sagt, BBB (BigBlueButton) der TU Wien kann benutzt werden. Er möchte einen neuen Server aufsetzen.

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.

Teilnahmeinformationen sollen auf der Website `forth-ev.de` angezeigt werden.

Weitere Projekte des Vereins werden genannt: Bitkanone 2 (Gerald Wodni), Feuerstein (Wolfgang Strauß), Gforth 1.0 (Anton Ertl, Bernd Paysan, Gerald Wodni), AmForth (Erich Wälde), Mecrisp (Matthias Koch).

Entlastung des Direktoriums

Anton Ertl stellt den Antrag, das Direktorium zu entlasten. Der Antrag wird mit 12 Ja–Stimmen, 0 Nein–Stimmen und einer Enthaltung angenommen.

Das Direktorium ist damit entlastet.

Wahl des Direktoriums

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

In einer geheimen Wahl über die Internetplattform `strawpoll.de` werden alle Kandidaten einstimmig gewählt.

Alle Gewählten nehmen die Wahl an.

Projekte

Ein neues Forth–Buch

Carsten Strotmann möchte ein modernes Forth–Buch ins Leben rufen. Es soll unter einer freien Lizenz erscheinen und als Download oder auch gedruckt erhältlich sein. Eine englische und eine deutsche Version sind angedacht. Als Forth–System soll Gforth 1.0 verwendet werden. Auch Video–Tutorials zur Begleitung der Themen im Buch sind geplant. Carsten Strotmann wird einen Artikel über das Projekt für die Vierte Dimension schreiben. In einem noch zu findenden Termin sollen Einzelheiten besprochen werden.

In dem Zusammenhang fragt Wolfgang Strauß nach dem Status des Projektes einer Aktualisierung des Buches „Thinking Forth“. Bernd Paysan erklärt, dass an dem Projekt momentan nicht gearbeitet wird.

Anton Ertl erwähnt sein Projekt eines Buches mit dem Titel „Rethinking Forth“.

Forth–Flyer (à la Mecrisp–Flyer)

Gerald Wodni möchte einen aktuellen Forth–Flyer zum Verteilen auf Messen gestalten. Als Basis soll der vorhandenen Mecrisp–Flyer dienen.

BitKanone 2

Gerald Wodni spricht sein Projekt „BitKanone 2“ an. Es fehlt seit zwei Jahren eine Dokumentation. Mitstreiter werden gesucht.

Projekt Feuerstein

Wolfgang Strauß spricht über das Projekt Feuerstein. Es soll den Einstieg in die Programmiersprache Forth auf RISC–V–Mikrocontrollern erleichtern. Vorgestellt auf der Online–Tagung 2020, sind inzwischen eine Fülle von Ideen besprochen worden, erste Quelltexte entstanden und Entwicklungsumgebungen getestet und auch entwickelt worden. Aus der Fülle an Daten wird in den nächsten Wochen ein Konzept für die weitere Entwicklung des Projekts ausgearbeitet.

Gforth 1.0

Für das Open–Source–Forth Gforth steht der Sprung auf Version 1.0 an. Es ist bis dahin noch einiges zu tun. Die Textdatei „ToDo“ in der Gforth–Distribution führt die Aufgaben auf.

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 Wolfgang Strauss.

Martin Bitter hielt die Laudatio.

Geplant ist die Übergabe des Drachens an den Drachenhüter nebst Gruppenfoto im Unperfekthaus in Essen im Rahmen eines Treffens der lokalen Forthgruppe Ruhrgebiet.

Verschiedenes

Carsten Strotmann macht sich Gedanken, wie man neue Mitglieder gewinnen kann. Er schlägt vor, auch international aktiv zu werden. Der Verein ist aktiv im Internet u. a. auf den Plattformen YouTube, Twitch, GitHub und natürlich auf der eigenen Website mit Wiki–Bereich.

Anton Ertl fragt sich, wie Forth–Material langfristig archiviert werden kann. Die Website `taygeta.com` hat ein großes Angebot. Auch auf den complang–Seiten der TU Wien befindet sich eine umfangreiche Sammlung an Informationen über Forth.

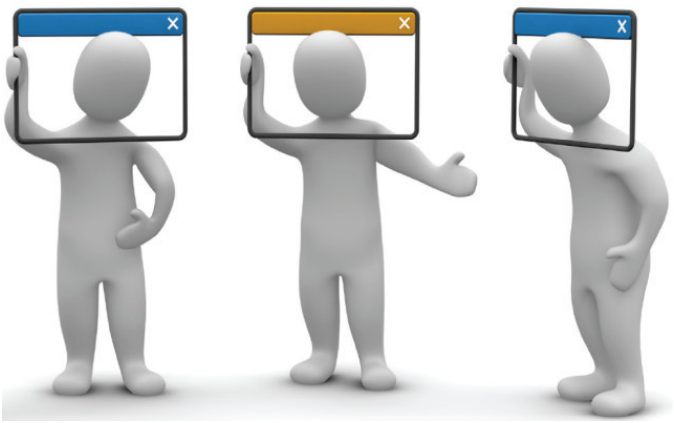
Es wird überlegt, wann die nächste Forth–Tagung stattfinden soll. Gerald Wodni schlägt eine Tagung per Videokonferenz im Frühjahr vor. Eine Präsenzveranstaltung im Sommer soll je nach Infektionsgeschehen stattfinden.

Schluss

Die Jahresversammlung endet um 12:05 Uhr.

Wesel, den 19.12.2021

gez. Wolfgang Strauß



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 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://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**$_p$

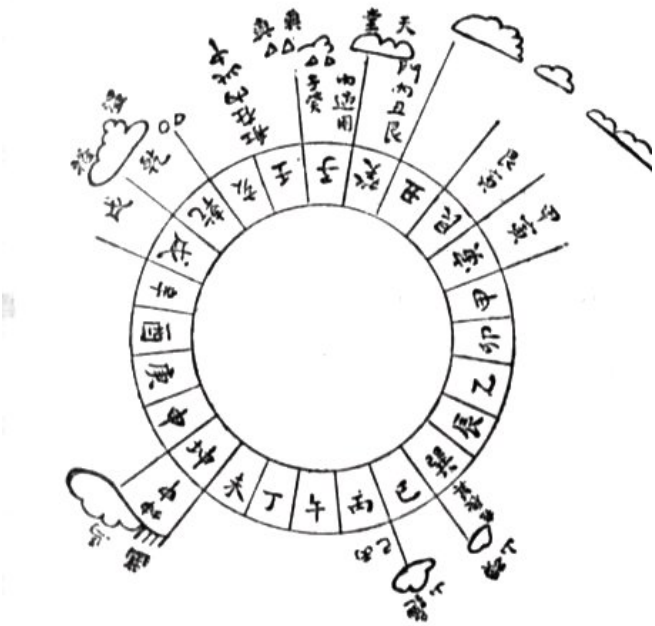
Termine

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 lernen
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`

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.

Vom Drachenhüten und anderen Schwierigkeiten

Wolfgang Strauß

Tja, jetzt hat es mich auch erwischt.[1] *Swap, der Hausdrache der Forth–Gesellschaft, ist bei mir eingezogen und mischt schon kräftig mit in Sachen Forth.*

Ein langer Weg

Normalerweise schließt sich Swap dem neuen Drachenhüter direkt auf der Forth–Tagung an. In Pandemiezeiten ist aber nichts normal. Unsere Tagung fand per Videokonferenz statt, also nix mit Übergabe. Der geplanten Zusammenführung bei einem Treffen der lokalen Forth–Gruppe Ruhrgebiet machte die Pandemie dann später ebenfalls einen Strich durch die Rechnung. Also blieb Swap nichts anderes übrig, als per Kurier in einem Postpaket durch die Lande an den Niederrhein zu reisen.

Das Malheur mit der Schatzkiste

Swap reist niemals allein. Er hat immer eine Kiste mit all seinen gesammelten Forth–Kostbarkeiten dabei. Jeder Drachenhüter fügt einen weiteren Schatz zu dieser Aussteuer hinzu.

Was sich nun genau in dem dunkelen Paket zugetragen hat, in welchem Swap zu mir kam, wird wohl für immer ein Geheimnis bleiben. Ich meine jedoch ein schlechtes Gewissen bei Swap bemerkt zu haben, als er mir erzählte, es hätte ihm am Fuß gejuckt und als er sich kratzen wollte, hätte er einen trockenen Knacks gehört, sich aber nichts dabei gedacht.

Abbildung 1: Na, die Schatzkiste hat es wohl hinter sich. Das kommt dabei heraus, wenn Drachen sich langweilen.

Jedenfalls hat mir Swap gleich seine Ideen für eine neue Schatzkiste mitgeteilt. Sie soll u. a. stabil sein (ach nee!), mehrere Ebenen zur Aufbewahrung und ein Geheimfach haben. Wir zwei haben jetzt einen Deal: Ich baue ihm eine neue Truhe und er versorgt mich mit tollen Forth–Ideen.

Drachenhüten

Nun habe ich also einen Mitbewohner, der das kollektive Forth–Wissen aller bisherigen Drachenhüter in sich trägt. Ich habe vor, das Jahr mit Swap gut zu nutzen und meinen Horizont mit seiner Hilfe zu erweitern.

Abbildung 2: Manchmal flattert Swap auf meine Schulter und schaut sich an, was ich da so mache. Hier wundert er sich, wieso ich in einem Buch über Forth–Entwurfsstrategien lese, wo das Thema doch trivial ist.

Mein spezielles Interesse gilt Forth auf Mikrocontrollern, auch gerne RISC–V. Hier ist es mir ein Anliegen, Neugierigen den Einstieg in die Materie mit guter Dokumentation und passenden Werkzeugen zu erleichtern. Das ist eine Menge Arbeit, aber auch jede Menge Spaß. Und dümmer wird man auch nicht davon.

Unter dem Namen „Projekt Feuerstein“ arbeitet momentan ein kleines Team an der Umsetzung des Themas. Mehr dazu im nächsten Jahr.

So, ich muss jetzt Schluss machen — Swap verlangt nach Erdbeeren. Wenn ich nur wüsste, wer ihm diesen Floh ins Ohr gesetzt hat …

[1] Es „erwischt“ jedes Jahr jemanden, den der Drachenrat auswählt, weil er sich in besonderer Weise um die Programmiersprache Forth verdient gemacht hat.