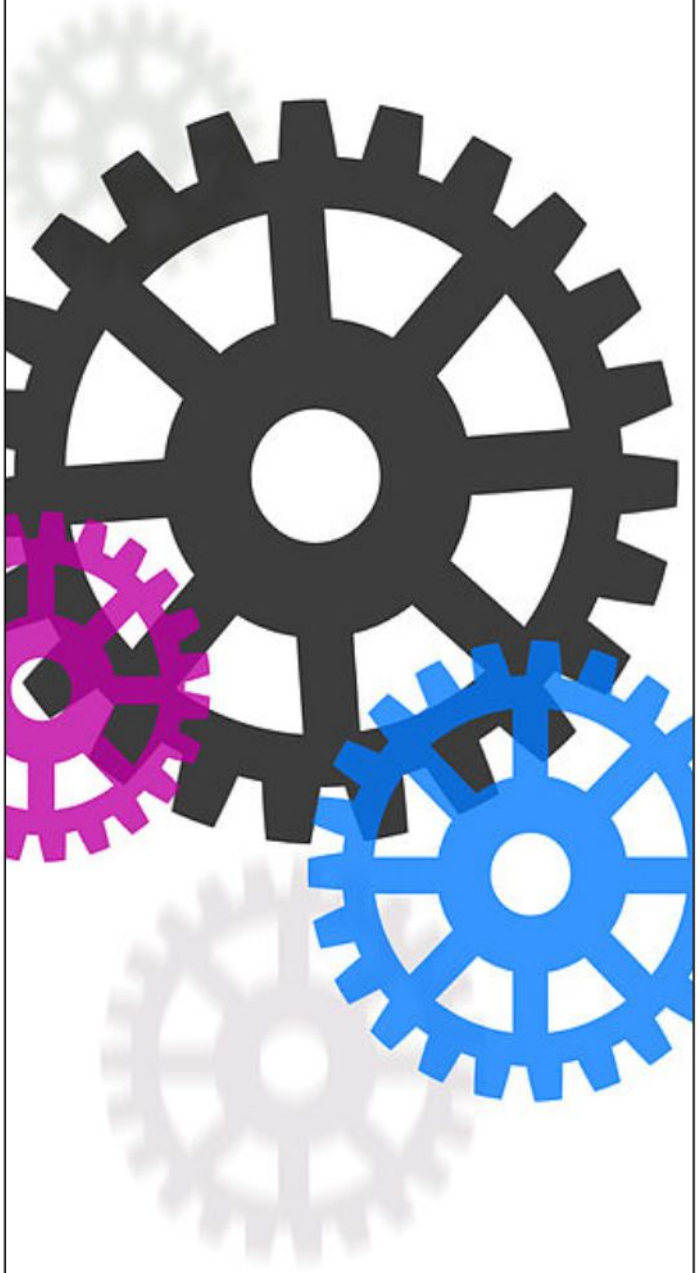
Vierte Dimension

2/2019

Das Forth–Magazin

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

In dieser Ausgabe:



Dienstleistungen und Produkte

tematik GmbH
Technische
Informatik

Feldstraße 143
D–22880 Wedel
Fon 04103 – 808989 – 0
Fax 04103 – 808989 – 9
mail@tematik.de
http://www.tematik.dewww.tematik.de

Seit 2001 entwickeln und vertreiben wir unter dem Markennamen "Servonaut" Baugruppen für den Funktionsmodellbau wie Fahrtregler, Lichtanlagen, Soundmodule und Funkmodule. Unsere Module werden vorwiegend in LKW-Modellen im Maßstab 1:14 bzw. 1:16 eingesetzt, aber auch in Baumaschinen wie Baggern, Radladern etc. Wir entwickeln mit eigenen Werkzeugen in Forth für die Freescale-Prozessoren 68HC08, S08, Coldfire sowie Atmel AVR.

Forth–Schulungen

Möchten Sie die Programmiersprache Forth erlernen oder sich in den neuen Forth–Entwicklungen weiterbilden? Haben Sie Produkte auf Basis von Forth und möchten Mitarbeiter in der Wartung und Weiterentwicklung dieser Produkte schulen?

Wir bieten Schulungen in Legacy–Forth–Systemen (FIG–Forth, Forth83), ANSI–Forth und nach den neusten Forth–200x–Standards. Unsere Trainer haben über 20 Jahre Erfahrung mit Forth–Programmierung auf Embedded–Systemen (ARM, MSP430, Atmel AVR, M68K, 6502, Z80 uvm.) und auf PC–Systemen (Linux, BSD, macOS und Windows).

Carsten Strotmann carsten@strotmann.de
https://forth-schulung.de

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

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

Cornu GmbH
Ingenieurdienstleistungen
Elektrotechnik

Weitlstraße 140
80995 München
sales@cornu.de
www.cornu.de

Unser Themenschwerpunkt ist automotive SW unter AutoSAR. In Forth bieten wir u.a. Lösungen zur Verarbeitung großer Datenmengen, Modultests und modellgetriebene SW, z.B. auf Basis eCore/EMF.

KIMA Echtzeitsysteme GmbH

Güstener Straße 72 52428 Jülich
Tel.: 02463/9967-0 Fax: 02463/9967–99
www.kimaE.de info@kimaE.de

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

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

Tannenweg 22 m D–18059 Rostock
https://www.fortech.de/

Wir entwickeln seit fast 20 Jahren kundenspezifische Software für industrielle Anwendungen. In dieser Zeit entstanden in Zusammenarbeit mit Kunden und Partnern Lösungen für verschiedenste Branchen, vor allem für die chemische Industrie, die Automobilindustrie und die Medizintechnik.

Ingenieurbüro
Klaus Kohl–Schöpe

Tel.: **(0 82 66)–36 09 862**
Prof.–Hamp–Str. 5
D–87745 Eppishausen

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

Mikrocontroller–Verleih
Forth–Gesellschaft e. V.

Wir stellen hochwertige Evaluation–Boards, auch FPGA, samt Forth–Systemen zur Verfügung: Cypress, RISC–V, TI, MicroCore, GA144, SeaForth, MiniMuck, Zilog, 68HC11, ATMEL, Motorola, Hitachi, Renesas, Lego ...
https://wiki.forth-ev.de/doku.php/mcv:mcv2

Titelbild: Zahnräder nebeneinander Autor: www.elbpresse.de 29 July 2015; Ausschnitt aus dem Werk des Autors.

Quelle: Wikimedia Commons

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

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

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe/Quartal

Einzelpreis

4,00 € + Porto u. Verpackung

Liebe Leser,

dieses Jahr ist die EuroForth bei uns in Hamburg. Sich jetzt zu registrieren, ist eine gute Idee! Auf der letzten Seite findet ihr alles, was dazu nötig ist.

Dieses Heft hat eine Menge Beiträge, die sich mit den „Innereien“ von Forth befassen. H. C. Chen hat es geschafft, Forth auf den Punkt zu bringen. Nur ein Blatt Papier braucht er auf seinen Vorträgen, um Forth so zu erklären, dass man loslegen kann!

In letzter Zeit haben wir in Deutsch und in Englisch publiziert.

Nun kommt Niederländisch dazu. Willem Ouwerkerk berichtet vom Wettbewerb, den die HCC forth gebruikesgroep bei ihrem Treffen neulich ausgerichtet hat. Eine rührige lebendige Forth-Gruppe rund um MCUs mit noForth.

Erich Wälde war unser Protokollant auf der diesjährigen Vereinsversammlung bei der Forth–Tagung. Vielen Dank für deine Mühe! Die Vorträge der Tagung hat Bernd Paysan ins Wiki des forth–ev hochgeladen — heutzutage wird gefilmt, nicht mehr gedruckt.

Bernd Paysan hat dem Gforth die „Constantenfaltung“ beigebracht. Der Begriff hat mich zunächst verwirrt: Wäsche falten, Zitronenfalter und jetzt auch noch Konstanten falten — es gibt doch Vieles, was man im Deuschen falten kann. Gemeint ist, dass man Berechnungen im Quellcode, die auf einen festen Wert hinauslaufen, auch „wegfalten“ kann und statt dessen gleich der berechnete Wert compiliert wird.

Matthias Trute nimmt den Paketierer SNAP in Dienst für eine stabile Gforth–Distribution auf verschiedenen Plattformen. Ich staune, was diese Jungs so machen.

Noch mehr staune ich ja immer, wenn es um die Forth–Hardware selbst geht. Willi Stricker beherrscht diese Kunst. Sein STRIP–Prozessor wird erweitert, bekommt mehr Speicher. Grundlegendes dazu wird erklärt.

Und dann habe ich selbst einen kleinen Beitrag geleistet über etwas Grundlegendes im Forth, das zeitlich verschobene Compilieren.

Schließlich zeigt Carsten Strotmann, wie man Forth–Anwendungen vom Betriebssystem unabhängig machen kann. In so einem „Unikernel“ bekommt die Applikation nur noch das absolut Nötige für seine Laufzeitumgebung mit auf den Weg.

Nachruf

Unser Vereinsmitglied Dr. Egmont Woitzel ist am 14.05.2019 überraschend im Alter von 58 Jahren verstorben. Die Beisetzung fand am 24.06.2019 auf dem Westfriedhof in Rostock statt. Viele von uns kannten ihn gut von den Tagungen, die er früher gern besucht hatte, oft begleitet von seiner Frau Ute. Gern erinnere ich mich an ihn als Mentor und Forth–Freund und Go–Spieler, der bis tief in die Nacht nach den Vorträgen noch mit Anderen am Brett zu finden war, selten mal besiegt. Bei den letzten Tagungen habe ich ihn vermisst, nun wurde klar warum: Er war schwer erkrankt. Der Gedankenaustausch mit ihm bei den Treffen, seine Wahrheitsliebe und Klarheit und sein Humor werden mir fehlen.

Egmont, wir vermissen Dich.

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-2019-02

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

Ulrich Hoffmann — Kontakt: Direktorium@Forth-ev.de
Bernd Paysan
Carsten Strotmann

Grüße vom neuen Drachenträger[1]

Hallo ihr tapferen Drachenboten und ehemaligen Drachenträger,

gerade war ich zu Hause angekommen und damit beschäftigt, mein Abendessen vorzubereiten, als es auf einmal an meinem Küchenfenster klopfte: Klaus und Manfred, die Drachenboten, waren zu mir gekommen, um mir zu gratulieren, mir das Amt des Drachenträgers zu verleihen und mir Swap und seine Schatzkiste zur Bewachung anzuvertrauen. In alten Sagen haben Drachen immer besondere Fähigkeiten und Talente, und ich bin gespannt, was Swap im Mecrisp-Hauptquartier bewirken wird. Eins ist schon passiert: Es ist ein Strahlen auf all den Gesichtern erschienen, denen ich von meiner neuen Aufgabe als Drachenträger berichtet habe.

Matthias Koch

zForth

zFORTH ist ein minimales, in der Programmiersprache *C* geschriebenes Forth–System. zForth ist sehr portabel, es wurde schon erfolgreich auf x86–Linux/Win32/MS-DOS (mit Turbo–C 1.0!), x86_64–Linux, macOS, PPC morphos, ARM, ARM thumb, MIPS, Atmel AVR und Intel 8051 kompiliert. Der Kernel kompiliert zwischen 4 KByte und 18 KByte groß, je nach Architektur und Größe einer *cell*.

Der zForth–Wortschatz:

```
include forth/dict.zf

words
words name next ." s" loop loop+ do j i fi
else unless if times until again begin var
allot here .. cr != not =0 >= <= > < dec
inc +! over postpone ] [ # , @ ! save
include sin quit tell . emit _postpone
compiling trace latest h & ## lits key ,,
pick sys = r> >r ( ' jmp0 jmp rot swap !!
@@ immediate pickr dup drop % / * - + ;
: <0 lit exit
```

Sehr praktisch ist die eingebaute Trace–Funktion, mit der die Arbeit des inneren Interpreters sichtbar wird. Ist der Wert der Variable `trace` größer als Null, so gibt zForth beim Aufruf eines Wortes ein Trace der Forth–Maschine aus:

```
: square ( n -- n^2 ) dup * ;

1 trace !

0186 0000 | (exit) r<<0

9 square
>>9 r>>0
[square/0457]
0457 000b | (dup) <<9 >>9 >>9
0458 0007 | (*) <<9 <<9 >>81
0459 0000 | (exit) r<<0
```

Über das Wort `save` lässt sich das Dictionary des zForth in eine Datei mit dem Namen `zforth.save` speichern und per `zforth -l zforth.save` wieder einlesen. zForth lässt sich einfach in ein C–Programm integrieren; Forth–Quellcode kann aus dem C–Programm mit einem Funktionsaufruf interpretiert werden. Carsten Strotmann (cas)

`https://github.com/zevv/zForth`

Vor dem großen Teich ...

From the other side of The Big Teich – unter diesem Titel hat HENRY VINERTS lange Jahre direkt von den Treffen der *Silicon Valley FIG* berichtet. FRED BEHRINGER fasste seine Berichte und die Inhalte des niederländischen *Feigenblattes* in seiner Rubrik *Gehaltvolles* zusammen. Beides ist Vergangenheit.

Das *Feigenblatt* wird nicht mehr verlegt. Aber der niederländische Forthverein besteht noch: Die *HCC forth gebruikesgroep* (HCC–forth–gg) trifft sich regelmäßig jeden zweiten Samstag in jedem zweiten Monat. Recht kurz vor dem jeweiligen Termin erfolgt online eine gesonderte Einladung unter dem Titel *ForthWords–Jahr–Nr*, die unter anderem einen Überblick über aktuelle Themen und geplante Aktivitäten enthält. Wie so vieles, hängt dies vom Einsatz Einzelner ab.

Holländisch ist gar nicht so schwer. Es ähnelt sehr den norddeutschen Sprachgepflogenheiten. Tipp: Manchmal hilft es, laut zu lesen! Und außerdem ist Forth sowieso international.

Neugierig? **Werden Sie Förderer der HCC–Forth–gebruikersgroep.** Dies schrieb WILLEM OUWERKERK in einer Anzeige in der *Vierten Dimension*. Und es stimmt! Ein Verein lebt unter anderem von der Anzahl seiner Mitglieder. (Ab jetzt: kurz nachdenken — und aktiv werden ...)

[1] Für gewöhnlich sucht sich der Swap seinen neuen Hüter aus der Runde der Tagungsteilnehmer und zieht mit ihm in sein Zuhause. Dieses Mal wollte er jedoch mal eben so einge hundert Kilometer weiter wegflattern ...

Was war im Februar?

Willem Ouwerkerk berichtete über Interfaces mit noForth: ein M33–Spieler, der über ein RS232–Interface angesprochen wird und eine I²C–Verbindung zu einem Lichtsensor (APDS9300). Ein Foto zeigte einen Breadbordaufbau eines anderen Projektes: Micro–nRF24L01–Platinchen, OLED–Display für das Microlaunchpad mit noForth, Stromversorgung. Er merkte an, das die Platinchen stapelbar sein werden.

Ein Wettbewerb wurde ausgeschrieben: Schreibe einen Linienfolger für den Cosey–Robot. Es wurden online zwei Wordsets dazu veröffentlicht[2]. Beim Treffen wurde der jeweilige Code eingespielt und der Cosey gestartet. Der Ausschreiber bewertete unter Anderem nach der Eleganz der Bewegungen. Die Sieger stehen jetzt fest: J. J. Hoekstra und H. Luijkx[3].

Und dann ein Tipp von Gerard van Kramer. Er empfiehlt eine Suchmaschinenanfrage: „The art of electronics“, PDF, mootoo. Das führt zu einem Wälzer, voll mit Wissen zu Bausteinen und Schaltungen, auch MCUs.

Und im April?

Ausnahmsweise, weil der Saal anderweitig belegt war, fand das Treffen am 20. April statt. Der Nachmittag war bestimmt durch die Mitgliederversammlung. Für den Vormittag rief Willem Ouwerkerk jeden, der über ein Egelboard (noForth) und/oder ein nRF24L01 verfügte auf, dieses mitzubringen, um ein Funknetzwerk damit aufzubauen. Als Vorschau wurden Codeschnipsel dazu gezeigt.

Es gab Programmieraufgaben zum Umrechnen von Fahrenheit nach Celsius und umgekehrt. Einmal als Ganzzahl und einmal mit 10tel–Ausgaben.

Martin Bitter

Philosophisch–Mystisches

Forth ist eine wundervolle Programmiersprache. Das wissen wir schon. Aber: *mystisch*?

Beim Philosophieren über Forth bin ich auf folgendes gestoßen: In vielen Forthsystemen (allen?) gibt es undefinierte Wörter. Klar — der Benutzer wird ja sein Forth erweitern. Und was nicht definiert ist, kann ja auch nicht funktionieren. Wirklich?

Ich habe ein Wort gefunden, das man nicht definieren muss, das auch im laufenden System nicht definiert ist und das trotzdem, wie vom Benutzer gewünscht, funktioniert. Da es niemals definiert wurde, verbraucht es keinen Speicher! Besonders wichtig ist das für kleine MCUs; weder RAM noch FLASH werden tangiert. Eine weitere forthtypische Freiheit: Der Name des Wortes steht gar nicht fest. Er ist in weiten Grenzen beliebig.

Wie mag dieses Wort heißen?[4] Martin Bitter

Was .S können sollte

Anton Ertl fragte kürzlich im Kreise der Gforth–Anwender nach, ob das Verhalten von `.S` geändert werden sollte[5]. Da dieses Wort ja eigentlich nur beim Debuggen benutzt wird und es auf diesen großen Systemen auch keinen Grund gibt, da allzu sparsam zu sein mit dem Speicherplatzverbrauch und zu geizig bei der Ausführungsgeschwindigkeit, kann man sich auch üppigere Infos vorstellen, als nur die Werte vom Datenstack auszugeben. Hier seine Frage:

Classic or smart .S ?

> In Gforth the classic .S produces just numbers in the current base, e.g.:
>
> `<7>`[6] `140403976810720 0 140403977391576 140403977391576 0 11192792 140403976810872`
>
> Gforth also supports a smart `.S` that uses heuristics to determine what kind of data is on the stack and displays it appropriately; the same stack data displayed with the smart `.S` shows:
>
> `<7> ‘noop 0 ‘fuenf ‘fuenf 0 $81F9D8 ‘execute`
>
> where `‘<word>` means the execution token of <word>. Addresses are shown in hex with a "$" prefix. The smart `.S` currently also shows strings:
>
> `s" abcdefghijk" 3 /string .s`
>
> outputs:
>
> `<2> "defghijk" ok 2`

[2] Im Heft ist der Workshop–Aufruf veröffentlicht.

[3] Vorn, auf dem Boden sitzend; von links nach rechts.

[4] Auflösung irgendwo im Heft.

[5] Siehe auch: .S und ok — Eine Revision; Heft 4d2013-02

[6] Der Prompt von Gforth gibt an, wie viele Items noch auf dem Datenstack sind.

> Up to a few days ago you got the classic .S if you typed ".S". To get the smart .S, you had to type "..." .
>
> With the current git head, you get the smart .S when you type ".S". To get the classic .S, you have to say
>
> ```
> ' . is .s.
> ```
>
> After doing that, the last stack shown above is shown as
>
> ```
> <2> 21765267 8
> ```
>
> What do you think should be the default for .S and ~~: the classic or the smart .S ?
>
> - anton

Ihr seht, im jüngsten Release des Gforth gibt es diese üppigere Debugfunktion bereits. Sie wurde dort ... genannt. Drei Pünktchen, weil man das schnell eintippen kann. Denn das wird ja meist interaktiv gebraucht beim Debuggen. Und man ahnt auch schon, wie die smarte Version von .S gemacht worden ist. Das geschieht in der Zahlenausgabe, dem .S., das als deferred–word angelegt ist. So kann man das Verhalten von .S leicht umschalten. Die Phrase

```
' . is .s.
```
[7]

holt die klassische Ausgabe zurück. Und man selbst kann sich da nun auch nach Herzenslust austoben und weitere Infos einbauen, wenn man möchte.

Und in der FIG–Taiwan?

In dem Zusammenhang ist es ja vielleicht auch interessant, wie H. C. Chen das .S in seinem *peforth* macht. peforth ist eine Anwendung von Forth, mit der man alle möglichen *Python–Programme* debuggen kann. Ihm dient sein peforth und darin das .S als *AI-Debugger*[8]. Es wird von Breakpoints getriggert[9]. Das .S sieht dabei z. B. so aus, dass diese Eingabezeile:

```
OK 123 45.6 s" abc" 0xabcd 0b0011 true false none [] "" {}
OK .s
```

folgende Ausgabe ergibt:

```
0: 123  7Bh (<class 'int'>)
1: 45.6     (<class 'float'>)
2: abc      (<class 'str'>)
3: 43,981 ABCDh (<class 'int'>)
4: 3    3h  (<class 'int'>)
5: True     (<class 'bool'>)
6: False    (<class 'bool'>)
7: None     (<class 'NoneType'>)
8: []       (<class 'list'>)
9:          (<class 'str'>)
10: {}      (<class 'dict'>)
OK
```

Spannende Sachen passieren da drüben. mk

`https://github.com/hcchengithub/peforth/wiki/Know-the-FORTH-programming-language-in-one-page`

MicroCore alias uCore

Aus der Korrespondenz mit Klaus Schleisiek (Wendland) war zu erfahren:

> „Ahoi, klar gibt es den *uCore*, sogar besser denn je, wenn auch nicht wirklich im Internet. Das liegt vor allem daran, weil ich uCore in den letzten Jahren für meine Arbeit — FPGAs mit Inhalt füllen — genutzt und dabei ständig weiterentwickelt habe. Und damit war ich voll ausgelastet. Nachdem ich nach dem letzten Projekt auch noch die 2–Phasigkeit des uCore–Instruktionszyklus losgeworden bin, habe ich etwas, was ich für verbreitungswürdig halte. Ich werde ein *Buch* darüber schreiben, weil inzwischen reichlich erklärungsbedürftige neue Technik zusammengekommen ist.
>
> Auf der nächsten *euro4th* werde ich darüber berichten, wie man trotz einphasigem Zyklus das interne Block–RAM auslesen kann und dabei auch noch das Instruktionsregister los wird.“

Spannende Sachen auch hier! Da freue ich mich schon auf Hamburg. mk

PS:
Die Anfänge sind übrigens noch zu finden im alten SVN–Repos der Forth–Gesellschaft:

```
repos - Revision 2757: /microcore/trunk/Microcore
..
Library/
floating_point/
hardware/
lcc_stack/
uCore/
uCore2/
volks4th/
Powered by Subversion version 1.6.17 (r1128011).
```

Quelle: `https://mx.forth-ev.de/repos/microcore/trunk/Microcore/`

[7] Sprich: „Tick dot is dot.s.dot“

[8] AI — artificial intelligence; hierzulande KI — künstliche Intelligenz — genannt.

[9] Siehe auch seinen Beitrag dazu in diesem Heft.

Creole–Forth for Excel

Neulich beim Forthtreffen im Unperfekt–Haus in Essen meinte Andreas scherzhaft: „Für jede noch so komplizierte Formel gibt es mindestens zehn Menschen, die diese Formel in *Excel* formuliert haben. Ich bin überzeugt, dass es auch möglich ist, ein Forth in Excel zu programmieren.“ Und: Voila! `https://github.com/tiluser/Creole-Forth-For-Excel`

Ein Auszug aus dem README:

> Like other Forths, Creole has a colon compiler. Below are two example programs:

```
: SQR DUP Nx ;
: TESTBU BEGIN 1 N+ DUP 10 GT UNTIL ;
```

> If you paste these programs into the Input Area and hit submit, you will see their entry into the dictionary. To execute, put the following in the input area. ...

Interessant! Aber: Wer probiert es mal aus?

Martin Bitter

Retro Computing

Für die vielen Freunde alter Maschinen.[10]

Wer macht mit beim Pettil–Forth für 6502–Rechner?

Charlie Hitselberger, der Autor von Pettil–Forth, einem Forth–83 für den Commodore–PET, sucht nach Mitstreitern für sein Projekt. Gesucht werden Programmierer mit Interesse an Forth und 6502–Maschinensprache. Pettil soll auf andere 6502–Systeme portiert werden, z. B. Apple II oder C64. Der Quellcode zu Pettil ist auf Github zu finden; der Autor kann unter der Email–Adresse <pettilmobile@gmail.com> erreicht werden.

`https://github.com/chitselb/pettil`

Third Forth

Schon ein wenig älter (von 1998) ist *Third* von Ben Hoyt, ein Forth für 16–Bit–x86–PCs unter DOS (oder DOS–Emulator wie DOSBOX). 16–Bit–Forth–Systeme für DOS gibt es viele, doch im Gegensatz zu Third sind die alten Systeme der 1980er Jahre nicht ANS–Forth kompatibel. Third ist modular, das Kernsystem implementiert den CORE–Wortschatz von ANS–Forth, die *BIG–Variante* fügt `CORE DOUBLE EXCEPTION FACILITY FILE TOOLS SEARCH` und `STRING` hinzu. Ein neuer Third–Kern wird mit einem Forth–Meta–Kompiler gebaut. Third kommt mit ausreichender Dokumentation und ist freie Software unter der BSD–Lizenz. cas

Third Forth — `https://github.com/benhoyt/third`

DOSBox — `http://www.dosbox.com`

Den Gameboy mit Forth (wieder-)entdecken

Auf dem „Free and OpenSource Developers Meeting“ (FOSDEM) im Februar 2019 in Brüssel haben zwei Programmierer vom Projekt *GBForth* berichtet. Beruflich mit der Programmierung von JavaScript beschäftigt, sind Tijn Kersjes und David Vázquez Púa (siehe auch den Artikel zu *eulex* in VD 2019-01) auf eine Reise aufgebrochen, um ein Gerät ihrer Jugend, den Gameboy, zu erforschen. Dabei haben die Entwickler ein Forth erschaffen, um das System interaktiv zu erforschen, das ROM des Gameboy zu dokumentieren und ein kleines eigenes Gameboy–Programm, basierend auf Bernd Paysans *Sokoban* für gForth, zu erstellen. cas

„GBForth: Using Forth to understand the Game Boy“ — `https://fosdem.org/2019/schedule/event/retro_gbforth/`

GBForth Projekt — `https://gbforth.org`

Fortsetzung auf Seite 19

[10] Die Abbildung zeigt 3,5“–Disketten.

Know the FORTH programming language in one page

H. C. Chen

You guys know how to debug already. We all do. But when it comes to Machine Learning and Tensorflow or the likes, things are getting annoying if we were still using traditional debuggers. A programmable debugger was on my mind and probably in yours too. One breakpoint to investigate about everything! At this point, you can then test whatever you want, supported by all the power of forth. So I wrote peforth, which now serves as a programmable python debugger. And to picture it to programmers unfamiliar with forth, here is a short introduction. Visit peforth on github to learn more about this python debugging tool (see link below).

Fundamentals

For a *programmable python debugger*, FORTH syntax is what I choose among shells and programming languages I know, put aside to define a my–way for the purpose.

FORTH is the easiest programming language.

Everything is a *command* in FORTH.

A number, say `123`, is a command that dictates FORTH to push `123` onto the FORTH data stack. FORTH commands implicitly work on the data stack all the time.

Whether the number is an integer or a float, single or double precision, or even a complex is up to the host environment; for peforth that's python.

`+` is a command too that dictates FORTH to pop two operands out of the data stack and `+` them and push the result back to the data stack. As a convention, most FORTH commands consume their operands, like the `+` command mentioned here. Keep this in mind.

`+` is a "word" in FORTH. Anything in FORTH that is not a number is a "word".

The command line where you type in commands is `TIB`, terminal input buffer.

Some words get their operands from `TIB` instead of from the data stack. For example `." hello world!"` prints hello world! where the word `."` is of that kind that gets its operand from the `TIB`.

`TIB` can be given with an entire text file like the peforth source code `peforth.f` that builds up the peforth system top on its kernel `projectk.py` that provides only two beginning words `code` and `end-code`.

FORTH has two working states, *interpret state* and *compiling state*. I believe, most of our debugging usages will be only in the interpret state. So you probably never need to know about the compiling state. But it's easy! When we define a colon word like

```
: hi ." Hello World!" cr ;
```

between `:` and `;` is in the compiling state.

The FORTH programming language is the result of a set of words that work together. In contrast to other languages, `while`, `repeat`, `if`, `then` and `else` are not reserved key words. They are as normal as other words. Variable, constant, for–loop, object oriented, or whatever future programming language concepts, all of them are not from FORTH but from the specific FORTH system author who defined those words that bring out such meanings.

I repeat, we have covered the most important points about FORTH *already* except words, words, and words. You can list all peforth words by the `words` command and view their help message and comments by `help <word name>`, or just `help` to list them all, and see their details through `see <word name>`.

We're going to do exercises with words that are supposed to be used in your debuggings in the next section.

peforth one–liners[1]

Not only a number pushes itself onto the data stack but also strings, booleans, etc. Try copy–paste this line to peforth:

```
OK 123 45.6 s" abc " 0xabcd 0b0011 true false
none [] "" {}
```

That `OK` is the terminal prompt. The input follows: `123` is an integer number, `45.6` a float, `abc` a string, `0xabcd` a hexadecimal number, `true` and `false` are booleans, `none` is pythons 'nothing', `[]` is an array or a list, `""` is the null–string, and `{}` a set.

`.s` command dumps the data stack without consuming anything.

```
OK .s
0: 123 7Bh (<class 'int'>)
1: 45.6 (<class 'float'>)
2: abc (<class 'str'>)
3: 43,981 ABCDh (<class 'int'>)
4: 3 3h (<class 'int'>)
5: True (<class 'bool'>)
6: False (<class 'bool'>)
```

[1] The appearance was adapted for the print version. The online–original is even more impressive. See his github page.

```
7: None (<class 'NoneType'>)

8: [] (<class 'list'>)

9: (<class 'str'>)

10: {} (<class 'dict'>)

OK

OK dropall .s <----drop all cells from the
data stack

empty <------------so it's now empty
```

More complicated things are represented by in–line python directly:

```
OK dropall py> 67+89j py> {'aa':11,'bb':22}
py> [11,22,33] py> {44,55,66} .s
```

The items put on the stack are: complex number, dictionary, list (array), set, finaly printed by `.s` command:

```
0: (67+89j) (<class 'complex'>) --

1: {'aa': 11, 'bb': 22} (<class 'dict'>)

2: [11, 22, 33] (<class 'list'>)

3: {66, 44, 55} (<class 'set'>)
```

Some peforth words are actually the same thing represented by in–line python:

```
OK {} py> {} = . cr

True
```

The '=' command pops two operands, compares them and pushes back a boolean.

```
OK [] py> [] = . cr

True

OK "" py> "" = . cr

True

OK
```

For in–line python, try copy–paste these lines together:

```
cls dropall py: print('hello')
py:~ print('hello world!')
.s py> 12+34j .s
dropall py>~ 12 + 34j
.s
```

The executed results of the above lines are:

`cls` Clears the screen.

`py:` Enters in–line python, no return value, no spaces allowed in the python statement; `hello` is printed imediatly.

`py:~` In–line phthon, no return value. The rest of the line are all python. Spaces are therefore allowed; `hello world` is printed imediatly.

The next two examples for `py>` and `py>~` both result in the same python item on the stack:

```
0: (12+34j) (<class 'complex'>)
```

Now use `pop()` and `tos()` in in–line python:

```
OK dropall

py> 12+34j py> tos().real py> pop(1).imag

.s

0: 12.0 (<class 'float'>)

1: 34.0 (<class 'float'>)

OK
```

To understand the above example, let's do it again step by step commented with stack diagram:

```
                "stack diagrams" are like these
                they are actually comments
                ------------------------------
dropall
py> 12+34j      ( 12+34j )
py> tos().real  ( 12+34j 12.0 )
py> pop(1)      ( 12.0 12+34j )
:> imag         ( 12.0 34.0 )
.s
  0:  12.0 (<class 'float'>)
  1:  34.0 (<class 'float'>)
OK
```

The *Fantastic Four* members from the FORTH tradition: `dup`, `swap`, `over`, `drop` and friends `pick`, `roll`, `nip` and `rot` are not used as often as traditional FORTH because in–line python is convenient enough. Copy–paste the below block to see their help messages at once:

```
cls
help dup
help swap
help over
help drop
help pick
help roll
help nip
help rot
help -rot
```

Results :

```
OK help dup

( a -- a a ) Duplicate TOS.

OK help swap

( a b -- b a ) stack operation

OK help over

( a b -- a b a ) Stack operation.

OK help drop

( x -- ) Remove TOS.

OK help pick

( nj ... n1 n0 j -- nj ... n1 n0 nj ) Get a
copy

Use py> tos(n) is better I think

OK help roll

( ... n3 n2 n1 n0 3 -- ... n2 n1 n0 n3 ) Make
a rolling
```

```
see rot -rot roll pick

OK help nip

( a b -- b )

OK help rot

( w1 w2 w3 -- w2 w3 w1 )

see rot -rot roll pick

OK help -rot

( w1 w2 w3 -- w3 w1 w2 )

see rot -rot roll pick

OK
```

Object, dictionary, function, and list names mostly appear with `.` , `(...)` , or `[...]` , for example:

```
py> print . cr \ "print" is a function

py: print('hello') \ it mostly followed with a (...)

py> print :: ('hello') \ same thing in peforth syntax
```

so `::` is the peforth connector in between the name and the followings. Again, to involve spaces we use `::~` :

```
OK py> print ::~ ( " hello world ! ! " )

hello world ! ! OK
```

Therefore `:>` and `:>~` are easy now, they leave a return value at the `TOS`, top of the stack, after the execution:

```
OK py> {'aa':11,'bb':22}
\ now tos() is {'aa':11,'bb':22}

OK :> ['aa'] . cr
\ equivalent to: py> pop()['aa']

11

OK
```

If you find any questions, please open *issues* to the peforth GitHub. If you want to read a book, although I wish that the aboves are enough for all python debuggings, I recommend OLPC's Forth Lessons or many other resources recommended there. If you find words mentioned in it are not existing in peforth then you can define them by yourself. No kidding, refer to peforth.f to see how to build an entire FORTH system from only two beginning words code and end–code, indeed FORTH is the easiest programming language ever.

May the FORTH be with you!

Links

H. C. Cheng, FIG Taiwan:
`https://github.com/hcchengithub/peforth/wiki/Know-the-FORTH-programming-language-in-one-page`

One Laptop per Child Forth Lessons
`http://wiki.laptop.org/go/Forth_Lessons`

Abbildung 1: Taipei, the capital city and financial centre of Taiwan — Skyline, Blick vom Elefantenberg auf die Stadt. [Bildquelle: Heeheemalu; CC BY-SA 4.0; Taipei Skyline (Wikipedia)]

Workshop: Cosey als lijn volger

Willem Ouwerkerk

Im Februar diesen Jahres veranstaltete die HCC forth gebruikesgroep bei ihrem Treffen ein Cosey-Wettrennen. Cosey? Das ist der jüngste kleine Robot der Gruppe. Ihr seht ihn im Bild. Der Aufruf zum Treffen ist hier mal wiedergegeben, um zu zeigen, was die Gruppe dort so macht. Niederländisch lesen ist so schwer nicht, ihr werdet es schon schaffen! Das Listing zeigt den Code des Gewinners J. J. Hoekstra.

De sensoren

Dit zijn de sensoren die we gaan gebruiken, de middelste sensor is voor simpel lijn volgen niet nodig.

Abbildung 1: Cosey bottom view

De opgave

Cosey staat op een witte ondergrond met daarop een zwarte lijn van 25 mm breed. Het gaat er om op een originele manier een rondje over dat spoor te rijden.

Je moet bijsturen als een sensor de zwarte lijn detecteert, doe je dat voldoende dan blijft Cosey de lijn volgen.

Let op! het gaat om originaliteit in software en/of manier van voortbewegen. En de winnaar krijgt een prijs!

Iedereen krijgt maximaal 30 minuten maar mag voorafgaande aan de opdracht Cosey iets laten doen, om extra informatie te verkrijgen. Eventueel met een zelfgeschreven, test die wel vooraf ingediend moet worden.

Het woord `ADJUST` laat Cosey om zijn as draaien en de sensoren afregelen. `S>` gebruikt de data. Zo lees je ze uit:

- Links met `4 S>`
- Links midden `5 S>`
- Rechts midden `6 S>`
- Rechts met `7 S>`

Voorb:

```
4 S> BLACK < \ True als links op de lijn zit
```

Nadat een `READ` is gedaan staat de sensor data klaar en ze zijn uit te lezen met `S>` De gelezen waarde loopt ruwweg van decimaal 0 tot 1300 of meer. Waar alles beneden 500 zwart genoemd wordt en alles boven 750 wit. De motorsnelheid heeft een bereik van 0 tot 780 !!

Twee woorden sets

De basis set

`ADJUST`	(1 to 20 --) Rotate while storing minimum sensor values; give rotation speed from 1 to 20 decimal, does only one rotation and then stops.
`READ`	(--) Read & store four of the floor sensors and adjust sensors using the previous scanned values.
`.READ`	Show sensor readings from left to right.
`S>`	(+n -- x) Read value +n from data array it's value ranges roughly from 0 to 2600.
`BLACK`	Value containing the black line treshold; any value lower then this must be black.
`WHITE`	Value containing the white line treshold; any value higher then this must be white.
`STOP`	(--) Stop Cosey.
`FORW`	(--) Cosey drives forward.
`LEFT`	(--) Turn Cosey to left.
`RIGHT`	(--) Turn Cosey to right.
`>SPEED`	(spd --) Set motor speed (0 to 780).
`COSEY-ON`	(spd --) Activate Cosey hardware & set default speed.

Voor specialisten nog extra

`>B&W`	(b w --) Set new values for black & white
`B&W`	(--) Default black & white values
`>LEFT`	(spd --) Set motor speed for left motor
`>RIGHT`	(spd --) Set motor speed for right motor
`LEFTROT`	(--) Rotate cosey to left
`RIGHTROT`	(--) Rotate Cosey to right
`BACKW`	(--) Drive Cosey backward

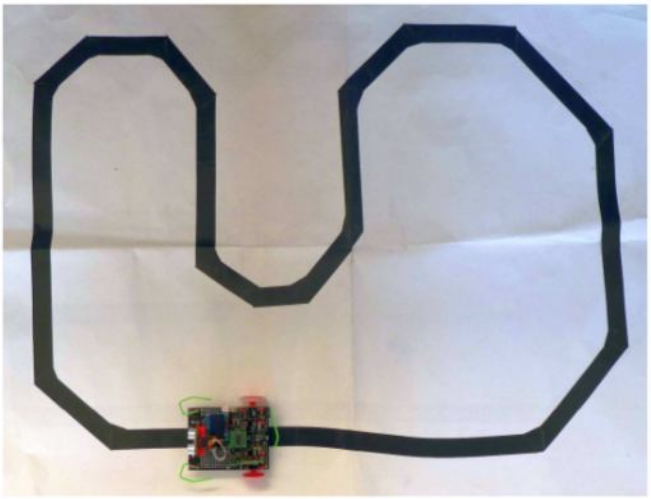


Abbildung 2: Cosey op het speelveld

Initialisatie

De initialisatie kan zo:

```
100 cosey-on 5 adjust ( Zet snelheid en ijk de sensors )
```

Run ...

Wat moet je doen om een lijn te volgen:

- Lees de sensoren
- Controleer of je op de lijn bent
- Corrigeer de rijrichting als dat niet zo is

Link

https://forth.hcc.nl/w/uploads/Agenda/ForthWords_2019_02.pdf

Listing

```
1
2   \ J.J. Hoekstra - 2019
3   ( slightly adapted )
4   ( based on experiences on 09 February 2019 )
5
6   decimal ( <- NoForth quirk )
7
8   : INIT 100 cosey-on 5 adjust ;
9
10  ( randomisation based on Marsaglia '03 )
11  ( all calculations 16bit - cyclus: 2^32-1 )
12  ( rando function is largely untested )
13
14  value sd0 23 to sd0 ( <- NoForth quirk )
15  value sd1
16
17  : DOSEEDS sd0 sd1 to sd0 ;
18
19  : RNDM doseeds
20
21      dup 7 lshift xor
22      dup 10 rshift xor
23      dup 9 lshift xor
24
25      dup to sd1
26      8 rshift ;
27
28  : >SP 20 * >speed ( forw ) ;
29
30  ( 6 strategies )
31  : RD  12 >sp forw ;
32  : LF   7 >sp left ;
33  : RT   7 >sp right ;
34  : SLF  4 >sp left left ;
35  : SRT  4 >sp right right ;
36  : RN   3 >sp rndm 128 < if left left
37                          else right right
38                          then ;
39
40  ( 16 decision-points )
41  create dtbl
42              ' rd    ,
43              ' slf   ,
44              ' lf    ,
45              ' slf   ,
46              ' rt    ,
47              ' rn    ,
48              ' rn    ,
49              ' slf   ,
50              ' srt   ,
51              ' rn    ,
52              ' rn    ,
53              ' slf   ,
54              ' srt   ,
55              ' srt   ,
56              ' slf   ,
57              ' rn    ,
58
59  : ESC? key? if key 27 = else false then ;
60
61  : BLACK? s> black < ;
62  : SENS> read 0 4 7 do 2* i black? - -1 +loop ;
63  : >STRAT 2* dtbl + @ execute 10 ( 0 ) ms ;
64
65  : START  init  begin sens> >strat esc? until  stop ;
66
67  init start
68
69  shield JEROEN\  freeze
```

Das mystische Wort im Forth

Es ist:

```
CS ( -- )   Clear Stack
```

Beim Aufruf von `CS` gibt es eine Fehlermeldung und der Stack ist leer — wie gewünscht. Man kann natürlich auch `cgh` oder `dfb` oder irgendein unbekanntes Wort schreiben. Caveat: In einigen Systemen gibt es `clearstack` tatsächlich. Bah! Das verbraucht Platz! Martin Bitter

Forth–Gesellschaft e.V.
Ordentliche Mitgliederversammlung 14.04.2019

Erich Wälde

Versammlunsleiter Martin Bitter

Protokollant Erich Wälde

Teilnehmer
Direktorium: Carsten Strotmann, Ulrich Hoffmann, Bernd Paysan
insgesamt 18 stimmberechtigte Mitglieder (Anzahl der ausgegebenen Stimmkarten)

Sitzungsdatum 14.04.2019

Sitzungsbeginn 09:15 Uhr

Sitzungsende 11:50 Uhr

Sitzungsort
Hotel Weingut Sandwiese
Fahrweg 19
67550 Worms–Herrnsheim

Begrüßung

Carsten Strotmann begrüßt im Namen des Direktoriums die anwesenden Mitglieder.

Wahl des Schriftführers

Als Protokollant wird Erich Wälde gewählt.

Wahl des Versammlungsleiters

Zum Versammlungsleiter wird Martin Bitter gewählt. Der Versammlungsleiter stellt fest, dass die Versammlung fristgerecht einberufen wurde. Es wurden 18 von 94 Stimmkarten ausgegeben. Damit sind mehr als 10 % der Mitglieder anwesend und die Versammlung ist beschlussfähig.

Ergänzungen zur Tagesordnung

Anträge zur Ergänzung der Tagesordnung liegen keine vor. Die Tagesordnung wird einstimmig angenommen.

Bericht des Direktoriums

a. Bericht der Verwaltung (Ewald Rieger)

Ewald Rieger verliest nunmehr im 12. Jahr den Bericht der Verwaltung.

Mitgliederentwicklung Im Jahr 2018 gab es zwei Neuzugänge und 5 Austritte (davon einer durch Ableben des Mitglieds). Der Verein hatte zum Jahresende 2018 94 Mitglieder. In 2019 sind bis dato zwei weitere Zugänge zu verzeichnen. Die Mitgliederzahlen weisen weiterhin nach unten (28 Mitglieder weniger seit 2008).

Finanzen Ewald Rieger erläutert im Detail die Einnahmen und Ausgaben, getrennt nach Verein und Zweckbetrieb (Vierte Dimension). Es ergibt sich ein Vermögen des Vereins von 7.252,13 € zum Jahresende.

Im ideellen Bereich ergeben sich Einnahmen von 2.188,00 € sowie Ausgaben von 2.045,54 €. Dabei schlägt vor allem der Betrieb von immer noch zwei Servern für die Internetpräsenz mit ca. 460 € zu Buche.

Im Zweckbetrieb sind Einnahmen von 2.037,06 € und Ausgaben von 2.649,46 € zu verzeichnen. Die Ausgaben sind mit der Abrechnung der Vierten Dimension 2017-04 im Jahr 2018 zu erklären.

Das vorhandene Vermögen sollte beim Finanzamt nicht zu Beanstandungen oder konkreten Plänen zur Ausgabe im Sinne des Vereinszwecks führen.

b. Bericht des Kassenprüfers (Thomas Prinz)

Die Prüfung der Kasse fand am Freitag, den 12.04.2019 durch Thomas Prinz statt. Er berichtet, dass alle Buchungen belegt und nachvollziehbar sind. Die Buchhaltung ist vorbildlich. Der Kassenprüfer empfiehlt die Entlastung der Verwaltung ohne Einschränkung.

Die Versammlung entlastet die Verwaltung einstimmig.

c. Rund um das Forth–Magazin (Ulrich Hoffmann)

Die *Vierte Dimension* erscheint weiterhin; sie ist die letzte gedruckte Veröffentlichung in Sachen Forth auf dem Planeten. Dank geht ausdrücklich an Michael Kalus, der sich mit hohem Einsatz um die Beschaffung von Inhalten und die Herstellung des Hefts kümmert. Das Heft 2019-01 wurde auf der Tagung ausgeteilt.

Das pdf–Archiv der Vierten Dimension wird ordentlich frequentiert und ist auch über Google zu finden. Carsten Strotmann erwähnt, dass der sanft angemahnte Redaktionsschluss auch ein guter Antreiber ist, eigene Artikel fertig zu stellen. Die Redaktion ist erreichbar unter `vd@forth-ev.de`.

Nachdem Fred Behringer bedauerlicherweise nicht mehr unter uns weilt, hat Wolfgang Strauß sich bereit erklärt, die Korrekturen im Heft zu übernehmen und damit bereits angefangen. Vielen Dank!

Die Webseite von `forth.org` erfährt nur noch wenige Updates. Dafür gibt es auf `github.com` ein `forthhub`. Auch die Webseite vom Forth Standard Comitee ist eine gute Quelle.

Es wird wie jedes Jahr ausdrücklich gewünscht, dass die Vorträge zur Tagung in Textform für die Vierte Dimension eingereicht werden.

An dieser Stelle wurde die Sitzung unterbrochen. Zum einen wurden Ewald und Andrea Rieger für ihren jahrelangen Einsatz für den Verein mit einem kleinen Geschenk geehrt. Wir danken!

Und zum anderen wurde der Drachenpreis vergeben — zum ersten Mal an eine Person, die nicht auf der Tagung anwesend war. Der Bote Klaus Zobawa wurde beauftragt, den Drachen und seine Schatzkiste auf dem Heimweg an den Preisträger Matthias Koch zu überbringen.

Matthias Koch hat in einer erstaunlich kurzen Zeit eine lange Liste von ARM-Cortex Prozessoren mit Mecrisp Forth ausgestattet und damit durchaus auch neues Interesse an Forth erzeugt. Mecrisp selbst ist ein native Forth, *welches über einen hohen Grad an Optimierungen verfügt. Der Drachenrat befand ihn würdig, den Preis zu erhalten.*

d. Internet–Präsenz (Ulrich Hoffmann)

Nach 15 Jahren auf dem alten Server mit der längst nicht mehr gepflegten Software `geeklog`, mit Spam und anderen Unsicherheiten und mit einer unguten Vermengung von Inhalt und Präsentation ist diese Ära am Vorabend zu Ende gegangen.

Das von Gerald Wodni gepflegte `kern.js` wurde auf dem neuen Server endgültig aktiviert, die Inhalte (seit 2013!) migriert und die Webseite hat seither ein neues Aussehen.

Email und die Fotoalben sind noch umzuziehen. Bis das Protokoll im Heft erscheint, sollte das alles Geschichte sein! Bitte Lob, Tadel, Ungereimtheiten, kaputte Links etc. an Gerald melden.

e. Außendarstellung und Projekte (Bernd Paysan)

Auf dem Vintage Computing Festival Europe waren wir 2018 nicht vertreten.

Die Maker Faire Hannover fand gleichzeitig mit der Euroforth in Edinburgh statt. Dennoch waren auf der Maker Faire vier Personen, um Forth an Interessierte zu vermitteln, unterstützt mit dem Modell–Truck von Servonaut.

Ebenso waren wir auf dem 35C3 in Leipzig vertreten, wo man neben „Chaos West“ aus dem Ruhrgebiet einen guten Stand mit hoher Frequentierung hatte. Dieses Mal konnte direkt ein neues Mitglied gewonnen werden.

Für dieses Jahr ist die Teilnahme an der Maker Faire Ostwestfalen/Lippe geplant. Es hat sich in der Vergangenheit sehr deutlich gezeigt, dass ordentliche Hingucker zwingend notwendig sind, um Publikum überhaupt bis zum Stand zu lotsen. Das Algenaquarium von Mattias Koch ist dafür phänomenal, allerdings schwierig zu transportieren. Dennoch sind wir nicht schlecht ausgestattet:

Die Bitkanonen sind inzwischen in einem Versand–tauglichen Zustand angekommen und können für solche Veranstaltungen bereitgestellt werden.

Der Triceps benötigt neue Servos (die alten sind wieder abgeschabt), diesmal teure mit magnetischen Winkelsensoren. Neue Teile für den Triceps sind per 3D–Druck zu erstellen, da es unseren bevorzugten Teilelieferanten nicht mehr gibt (Friedel Amend hat seine Werkstatt verkauft). Außerdem ist eine neue Ansteuerung mit einem Lattice IOSys geplant, da das vorhandene Board nur noch schlecht erhältlich ist (nicht mehr zur Nachahmung geeignet).

Martin Bitter hat den Stirling–Motor und seine Vermessung (Druck, Temperatur) inzwischen auch transporttauglich gemacht.

Das LKW–Modell von Servonaut ist ebenfalls einsetzbar.

Entlastung des Direktoriums

Thomas Prinz stellt den Antrag das Direktorium zu entlasten.

Die Versammlung erteilt mit 16 Ja–, keinen Nein–Stimmen und 2 Enthaltungen die Entlastung. Das Direktorium ist damit entlastet.

Wahl des Direktoriums

Die bislang amtierenden Mitglieder des Direktoriums sind bereit, für eine weitere Amtszeit zu kandidieren.

Das Direktorium (Carsten Strotmann, Ulrich Hoffmann, Bernd Paysan) wird einstimmig für eine weitere Amtszeit gewählt. Alle Gewählten nehmen die Wahl an.

Projekte

Drei der neuen **Bitkanonen v2** sind Eigentum des Vereins und zum Vorzeigen bei Veranstaltungen gedacht. Diverse Mitglieder haben ihre eigenen Exemplare erstanden. Auf der Software–Seite sind schon eine ganze Reihe interessanter Programme entstanden, wie z. B. `rain`, welches an Regentropfen erinnernde Spuren auf die LED–Anzeige malt, deren Farbe sich mit der Neigung der Platine ändert.

Es wird darüber diskutiert, in welcher Form diese Platinen als Bausatz oder bestückt bei Veranstaltungen verkauft werden können (sofern vom Veranstalter nicht ausdrücklich untersagt). Der Preis sollte bei vielleicht 42 € liegen (derzeit 56 €), was sportlich, aber noch machbar ist. Ein Teil des Erlöses soll als Spende an den Verein gehen. Die bevorzugte Variante sei es, dass Gerald (eigenes Gewerbe) die Platinen verkauft. Denn eine Verkaufstätigkeit des Vereins ist vereins– und steuerrechtlich sehr heikel.

Carsten Strotmann betreut weiterhin den **Mikrocontrollerverleih**.

Martin Bitter regt die Entwicklung eines sehr günstigen **Forth–Handhelds** an (vielleicht < 10 €), welcher in geringen Mengen auch zum Verschenken auf Veranstaltungen geeignet wäre. Vielleicht auf Basis der *Blue Pill*–Platine.

Wolfgang Strauß erwähnt den NumWorks [1], einen Taschenrechner mit offener Software/Hardware, auf dem auch Mecrisp–Stellaris läuft. Der hat allerdings mit 79 € auch seinen Preis.

Ulrich Hoffmann weist auf den AIR602 [2, 3, 4] hin, der einen WLAN–Anschluss zur Verfügung stellt bei einem Stückpreis von ca. 2 €.

Verschiedenes

Ewald Rieger regt an, ein **Web–Formular** zur Anmeldung im Verein bereitzustellen. Derzeit muss man ein Formular finden, ausdrucken, ausfüllen und per Post an die Verwaltung schicken. Zeitgemäß ist das nicht mehr. Das Webformular sollte gleich eine Mitgliedernummer auswürfeln und die notwendigen Zutaten zur Lastschrift und DSGVO enthalten.

Ewald und Andrea Rieger sind bereit, die **Verwaltung** für ein weiteres Jahr zu übernehmen. Für die Zukunft gibt es bereits einen Plan B. Die Verwaltung muss nur die buchhalterisch relevanten Unterlagen für ebenso buchhalterisch relevante Zeiten aufbewahren.

Die **Tagung 2020** soll in Berlin stattfinden. Carsten Strotmann sucht noch Mithelfer.

Michael Kalus berichtet, dass **Win32 Forth** vom Microsoft Virenscanner zu Unrecht als Schädling eingestuft wird — bei selbstmodifizierendem Code vielleicht nicht ganz von der Hand zu weisen. Ulrich Hoffmann berichtet, dass eine ältere Version (aus Braunschweig?) unproblematisch sei. Darüber hinaus kann ein von Microsoft beglaubigtes Zertifikat helfen (ca. 30 €/Jahr).

Wolfgang Strauß macht sich Gedanken, wie man denn dem **Mitgliederschwund** begegnen könnte. Es ensteht eine lebhafte Diskussion, nicht zum ersten Mal zu diesem Thema.

- Wir tun einiges, so auch diese Tagung, die immerhin 14 Vorträge sowie andere Aktivitäten vorweisen kann. Und viel mehr Leute, die etwas tun wollen und können, haben wir nicht.
- Die VD ist angesichts der Download–Zahlen ein wichtiger Teil.
- Vorschlag: *Forth in 4 Minuten*–Video für die Generation youtube. (OBS screencast?)
- Einleitung/Anfänger–Material? Gibt es, ist aber verstreut oder nicht als solches zu erkennen. Soll man ein Sonderheft für den Einstieg machen? Oder eine ordentlich gepflegte Wiki–Seite?
- Vorschlag youtube: Warum ich Forth hasse — und es dennoch tue
- Vorschlag youtube: die 30 schlimmsten Forth–Fehler — letztlich ist es der reißerische Titel, der vielleicht Zuschauer zieht.

Carsten Strotmann macht den Vorschlag, ob man denn vielleicht auch eine **Profi–Messe** (z. B. die embedded world in Nürnberg) mit einem gesponsorten Stand verzieren könnte? Gespickt mit einem kleinen Programmierwettbewerb vielleicht.

Schluss

Die Jahresversammlung endet um 11:50 Uhr.

Hechingen, den 03.06.2019

gez. Erich Wälde

Referenzen

1. `https://www.numworks.com`
2. `https://www.shotech.de/de/air602-wifi-module.html`
3. `https://www.shotech.de/de/air602-wifi-development-board.html`
4. `https://blog.hackster.io/the-air602-wifi-module-a-new-esp8266-competitor-2d00ad5fdd0c`

Abbildung 1: Eingang von Worms, Nibelungenbrücke über den Rhein. (Wikimedia)

Constant Folding für Gforth

Bernd Paysan

Die neuen Header von Gforth enthalten ein „intelligentes“ `compile,`*, mit dem man Optimierungen durchführen kann. Allerdings fehlt noch etwas, nämlich eine Infrastruktur für Konstantenfaltung. Dabei habe ich mich an Mecrisp angelehnt, denn* Matthias Koch *hat da eine sehr einfache Methode gefunden, wie man Konstanten faltet.*

Einleitung

Eigentlich fing alles damit an, dass Anton Ertl sich die neuen Header anguckte und entdeckte, dass es dort für TO und DEFER@ Methoden gibt, die sowohl mit `execute` als auch mit `compile,` behandelt werden; ersteres für Interpretation, letzteres beim Compilieren. Allerdings war das nicht ganz richtig, also nicht so, wie man es von einem „normalen“ Wort erwartete. Diese `compile,`–Methoden erwarteten auf dem Stack zusätzlich das xt des Wortes, auf das `to` oder `defer@` angewandt wurde. Beim Interpretieren ist das gut, beim Compilieren passt das nicht, denn das ist eigentlich ein Laufzeit–Effekt. `compile,` hat im Standard ja den Stackeffect (xt —) und nicht (xt1 xt2 —). Und ganz generell muss das Äquivalent gelten, dass `compile,` das gleiche ausführt wie `]] literal execute [[`, nur halt optimiert.

Aber natürlich will man so ein `set-to` so optimieren, dass das eh schon zur Compilation bekannte xt wegoptimiert und durch die zugehörige Adresse des Values ersetzt wird. Da wir in Gforth keine Infrastruktur zum Constant–Folding hatten, wurde jenes xt eben auf dem Stack übergeben. Um den „komischen“ Stack–Effekt zu erklären, hatten wir schon ein `to-opt:` für die Definition dieses Teils. Aber letztlich blieb die Situation unbefriedigend. Da musste also eine bessere Lösung her.

Konstanten im Compiler

Die grundlegende Compiler–Technik, die hier fehlt, nennt man *Constant Folding*. Im speziellen Fall ist das einfach eine Berechnung im Code, die zu einer Konstante evaluiert werden kann. Es gibt natürlich auch Fälle, bei denen nur ein Teil des Ausdrucks aus Berechnungen mit Konstanten besteht, die wegoptimiert werden können. *Mecrisp* kann das schon länger und macht das elegant, aber etwas abseits des Standards[1]: Alle Konstanten werden nicht gleich compiliert, sondern erst mal auf den Stack geworfen und ein Zähler wird um eins hochgezählt.

Trifft nun der Compiler auf ein Wort wie +, dann guckt er nach, ob das für Constant Folding geeignet markiert ist. Wenn ja, wird geprüft, ob die notwendige Anzahl Konstanten auf dem Stack liegt und das Wort einfach ausgeführt. Der Zähler wird entsprechend angepasst.

Matthias macht das mit vielen Flags und Code, der sich speziell die Flags ansieht. Das ist natürlich nichts für Gforth, wofür haben wir denn diese generische `compile,`–Methode? Eigentlich sollte das System von solchen Optimierungen gar nichts wissen. Aber gar nichts geht auch nicht — irgendwo müssen die Literals ja zwischengelagert werden. Ein Stack ist eine gute Idee, doch der normale Datenstack kommt leider nicht so recht in Frage, denn der ist für `literal` tabu. Das hat ja einen wohldefinierten Stackeffekt zur Compile–Zeit.

Zum Glück gibt es ja noch andere Stacks in Forth, wie Recognizer oder Search–Order, und Gforth hat für die schon im Kernel Worte wie `>stack` und `stack>` und ein bisschen mehr. So einen Stack können wir für die Literals auch machen, der ist nicht im Weg.

Alles, was kein Literal ist, muss vor dem Compilieren jetzt nur noch den Literal–Stack wegräumen, also ins Dictionary schreiben, dann ist die Kernel–Änderung auch schon erledigt. Das waren nur ein paar Zeilen.

```
Variable litstack
: >lits ( x -- ) litstack >stack ;
: lits> ( -- x ) litstack stack> ;
: lits# ( -- u ) litstack stack# ;
: lits, ( -- )
    litstack $@ bounds
    litstack @ >r  litstack off
    ?DO  postpone lit  I @ ,  cell +LOOP
    r> free throw ;
```

und natürlich der Einsatz in `basic-block-end`

```
:noname ( -- )
    lits, 0 compile-prim1 ; is basic-block-end
```

und `peephole-compile,` dem Compiler für echte Primitives:

```
: peephole-compile, ( xt -- )
    lits, here swap , xt-location
    compile-prim1 ;
```

Das war es dann auch im Wesentlichen schon, nur noch in `begin-like` und `then-like` musste ebenfalls auf jeden Fall ein `lits,` — denn auch an der Stelle gibt es Handlungsbedarf, weil da ja auch kein Code compiliert wird.

Konstanten falten

Um dann die Konstanten richtig zu falten, braucht man nur etwas generischen Code für verschiedene Fälle; ich habe die hier erst mal alle auscodiert. Haben wir zwei

[1] . . . was bei Mecrisp öfter vorkommt. :)

Inputs und ein Wort, das daraus einen Output zurückgibt, dann können wir das wie folgt machen:

```
: fold2 ( xt -- )
    \G check if can fold two literals;
    \G if so, does so, otherwise
    \G compiles the primitive
    lits# 2 u>= IF
        >r lits> lits> swap r> execute >lits
    ELSE  peephole-compile,  THEN ;
```

Sind genügend Inputs als Literals auf dem Stack, dann kann man die einfach herunternehmen, das Wort ausführen, und das Ergebnis wieder auf den Literal–Stack werfen. Dieses `fold2` muss man nur noch in die entsprechende Methoden–Tabelle des neuen Headers eintragen. Das sieht dann so aus:

```
: fold2: ( "name" -- )
    vt, ' dup (make-latest)
    ['] fold2 set-optimizer ;
```

Hier verwandeln wir das folgende Wort in ein aktives Wort um, als wäre es gerade compiliert worden, weil wir dann die `compile,`–Methode darin ändern können.

Das ist hier nur ein ausformuliertes Beispiel, wie man das machen kann. Tatsächlich ist die Implementierung noch ausgefeilter.

Feld–Zugriffe optimieren

Ein recht gebräuchlicher Sonderfall der Konstantenfaltung ist der Zugriff auf Felder. Dabei bringt das Feld–Index–Wort die Konstante, die aufaddiert werden muss, gleich mit. Liegt schon eine Konstante auf dem Stack, kann man die gleich zur Compilezeit addieren.

```
: field+, >body @
    lits# 0> IF  lits> + lit,
    ELSE  ['] lit+ peephole-compile, ,  THEN ;
```

TO–Methoden optimieren

Doch das ursprüngliche Problem harrt noch der Lösung: Denn bei einem `to` haben wir es ja nicht mit lauter Konstanten zu tun, sondern mit einem Effekt auf eine Speicherzelle. Für Values, die `+to` und `addr` implementieren, hängt das dann auch noch von einem Zustand ab, nämlich davon welcher Operand davor stand. Das muss also wirklich zur Compile–Zeit optimiert werden.

```
: value-to ( n value-xt -- )
    >body !-table to-!exec ;
opt: ( self-xt -- ) \ run-time: ( n -- )
    ?fold-to >body postpone Literal
    !-table to-!, ;
```

Im Grunde ist das da unten hinter dem `opt:` nur das gleiche wie oben, aber halt compiliert. Fast so, wie es vorher war, als Anton sich beschwert hat, dass das kein „normales“ Wort ist. Nur: Jetzt steht da noch `?fold-to`. Was macht das? Na, es guckt, ob da tatsächlich ein Literal auf dem Stack liegt, und ersetzt das nutzlose self–xt durch dieses Literal, also das value–xt, das dem `value-to` übergeben wird. Wenn da kein Literal liegt, dann compiliert es das Wort direkt und tut sonst weiter nichts. Solchen Direkt–Compilaten fehlt dann zwar die Unterscheidung von `to`, `+to` und `addr`, aber das ist vertretbar. Wer die auch haben will, muss halt optimieren.

Das `?fold-to` selbst ist also nur ein Dreizeiler:

```
: ?fold-to ( set-to-xt -- to-xt )
    lits# 0= IF  :, rdrop  EXIT  THEN
    drop lits> ;
```

Das war es. Gforths Image wird mit dem angehängten Konstantenfalter um knapp 3 % kleiner; die meisten Gelegenheiten zum Constant–Folding dürften mit `[ ... ]L`–Konstruktionen ohnehin schon abgedeckt sein.

Ausblick

Langfristig haben wir geplant, die in C codierten Primitives in Gforth durch einen eigenen Forth–Native–Code–Compiler zu ersetzen. Das intelligente `compile,` ist da nur ein Teil davon, ein Schritt in die richtige Richtung. Auch der Literal–Stack ist nur ein inkrementeller Schritt hin zu einem „richtigen“ Compiler. Der Literal–Stack wird in so einem System durch einen virtuellen Register–Stack ersetzt, und da kann man auch erkennen, ob ein Wert ein Literal ist. D. h. die Benutzung so wie hier gezeigt wird weiter möglich sein, auch wenn die Abfragen dann mehr kosten.

Referenzen

[1] Matthias Koch, *Flags, Konstantenfaltung und Optimierungen,* VD 2015–arm, Seite 16ff.

Listing

```
\ Constant folding for some primitives

\ Copyright (C) 2019 Free Software Foundation, Inc.

\ This file is part of Gforth.

\ Gforth is free software; you can redistribute it
\ and/or modify it under the terms of the GNU General
\ Public License as published by the Free Software
\ Foundation, either version 3 of the License, or (at
\ your option) any later version.

\ This program is distributed in the hope that it will
\ be useful, but WITHOUT ANY WARRANTY; without even the
\ implied warranty of MERCHANTABILITY or FITNESS FOR A
\ PARTICULAR PURPOSE.  See the GNU General Public
\ License for more details.

\ You should have received a copy of the GNU General
\ Public License along with this program. If not, see
\ http://www.gnu.org/licenses/.

: 2lits> ( -- d )  lits> lits> swap ;
: >2lits ( d -- )  swap >lits >lits ;
: 3lits> ( -- t )  2lits> lits> -rot ;
: >3lits ( -- t )  rot >lits >2lits ;
: 4lits> ( -- q )  2lits> 2lits> 2swap ;
```

```
28  : >4lits ( q -- )  2swap >2lits >2lits ;
29
30  : folder [{: n xt: pop xt: push :}d
31          lits# n u>= IF
32              >r pop r> execute push
33          ELSE  peephole-compile,  THEN ;] ( xt ) ;
34  : folds {: folder-xt -- :}
35      BEGIN  >in @ >r parse-name r> >in !
36          nip  WHILE
37              vt, ' dup (make-latest)
38              folder-xt set-optimizer
39      REPEAT ;
40
41  1 ' lits>  ' >lits folder
42  dup folds invert abs negate >pow2
43  dup folds 1+ 1- 2* 2/ cells cell/
44  dup folds floats sfloats dfloats
45  dup folds float/ sfloat/ dfloat/
46  dup folds c>s w>s l>s w>< l>< x><
47  dup folds wcwidth
48      folds 0> 0= 0<
49  1 ' lits>  ' >2lits folder
50      folds dup
51  2 ' 2lits> ' >lits folder
52  dup folds + - * / mod u/ umod and or xor
53  dup folds min max umin umax
54  dup folds drop nip
55  dup folds rshift lshift arshift rol ror
56  dup folds = > >= < <= u> u>= u< u<=
57      folds d0> d0< d0=
58  2 ' 2lits> ' >2lits folder
59      folds m* um* /mod swap d2*
60  2 ' 2lits> ' >3lits folder
61      folds over tuck
62  3 ' 3lits> ' >lits folder
63      folds */
64  3 ' 3lits> ' >2lits folder
65      folds um/mod fm/mod sm/rem */mod du/mod
66  3 ' 3lits> ' >3lits folder
67      folds rot -rot
68  4 ' 4lits> ' >lits folder
69      folds d= d> d>= d< d<= du> du>= du< du<=
70  4 ' 4lits> ' >2lits folder
71  dup folds d+ d-
72      folds 2drop 2nip
73  4 ' 4lits> ' >4lits folder
74      folds 2swap
```

(Fortsetzung der Meldungen)

Forth–Tagung 2019 im Hotel Sandwiese bei Worms

Die Forth–Tagung 2019 fand vom 12.04. bis zum 14.04.2019 im *Hotel und Weingut Sandwiese* bei Worms statt. Ihr findet die komplette Dokumentation zur Tagung in unserem Wiki, Bernd Paysan war so freundlich, diese dort inzwischen einzustellen. Die Liste zeigt, was ihr dort finden werdet.

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

Freitag, 12.04.2019

09:15 Gerald Wodni — Bitkanone v2. (video)

10:00 Manfred Mahlow — VOC statt VOCABULARY & VOCS, ITEMS und STICKY Words. (video, slides-1, slides-2, examples)

12:00 Anton Ertl — Der neue Gforth–Header. (video, slides)

15:00 Bernd Paysan — CloudCalypse, was nun? Daten aus Google+/Facebook&Co. nach net2o importieren. (video)

16:00 Martin Bitter — Die Forthkiste. (video)

16:30 Ulrich Hoffmann — Seedforth. (video, slides)

17:30 Wolfgang Strauß — Jupiter ACE. (video)

Samstag, 13.04.2019

14:15 Jörg Völker — Forth auf dem Steckbrett. (video)

15:15 Carsten Strotmann — Eulex Forth, ein modernes Bare–Metal–Forth. (video)

15:45 Erich Wälde — Platinenkrams. (video)

16:30 Carsten Strotmann — Unikernel und Forth: Zurück in die Zukunft. (video)

16:45 Alexander Korn — Forth–Prozessor im Messgerät. (auf Wunsch des Sprechers kein Video online)

17:20 Anton Ertl — Forth–Quellcode im Flash. (video)

21:15 Erich Wälde — Einführung in KiCad. (video)

Sonntag, 14.4.2019

10:20 Drachenpreisverleihung. (video)

mk

Abbildung 1: Weinhotel Sandwiese

Gforth als Snap

Matthias Trute

Gforth ist ein bemerkenswertes Tool. Vor allem, wenn man es nicht selbst nutzt, sondern als Ausgangspunkt nimmt, ganz was anderes zu machen. Der Anregung von Carsten Strotmann folgend, Gforth auch abseits der eingetretenen Pfade zu nutzen, gibt es wieder etwas Neues zum Ausprobieren.

Snap?

Was ist ein Snap? *Snappy* hat *Canonical*, die Firma hinter *Ubuntu*, erfunden, um Softwarepakete verteilen zu können. Anfangs war das wirklich nicht viel mehr und alle Welt hat mit den Schultern gezuckt: „one more package format“.

Mittlerweile ist das aber etwas gewachsen und net2o–Chatter Erich Wälde hat neulich die Frage geäußert, „Hast Du schon mal mit *snap*, dem neumodischen Paket–Dings, rumgekaspert?“. Die Diskussion, die sich daraufhin entspann, war kurz und lässt sich so zusammenfassen:

```
$ snap search gforth
Keine passenden Snaps für "gforth"
$
```

Selber Machen

Wenn es etwas nicht gibt, kann man entweder weiter darauf verzichten, oder es selbst machen. Ob es deswegen toll wird, sei dahingestellt. Snap ist zunächst ein Paketformat, das ein Programm und seine Abhängigkeiten enthält. Wenn man ein Snap startet, startet das angegebene Programm und es gibt eine Rechteverwaltung, sofern das der Snapersteller aktiviert hat.

Startpunkt ist eine Textdatei, die alles steuert. Die heißt `snapcraft.yaml` und das Tool zum Bauen heißt `snapcraft`. Das gibt es als, richtig, `snap` zum selber Installieren.

```
$ snap install snapcraft --classic
```

Die Datei `snapcraft.yaml` ist nicht schwer zu lesen:

```
name: gforth-mtrute
base: core18
version: '0.7.9-20190501'
summary: gforth snap
description: |
  This is gforth snap
grade: devel
confinement: strict
license: GPL-3.0+
apps:
  gforth:
    command: bin/gforth
    environment:
      GFORTHPATH: /snap/gforth-mtrute/current/
          lib/gforth/0.7.9_20190501
    plugs:
      - home
      - network
parts:
  gforth:
    source: https://www.complang.tuwien.ac.at/
        forth/gforth/Snapshots/.....
        /gforth-0.7.9_20190501.tar.xz
    plugin: autotools
   build-packages:
     - libtool-bin
     - libltdl-dev
stage-packages:
     - libltdl7
```

Ein paar allgemeine Angaben, dann eine Applikation und schließlich Komponenten. Die Komponenten werden aus den Quellen gebaut und die Applikation erhält einige Rechte, *plugs* genannt, wenn sie läuft.

Der Befehl `snapcraft` baut das Snap, das Ergebnis liegt dann im lokalen Verzeichnis als eine squashfs–Datei, wobei, wie auch bei der Erzeugung der Docker–Container, eine beeindruckende Menge an Meldungen vorbeiscrollt, die man nicht allzu ernst nehmen sollte.

```
gforth-snap$ snapcraft
Using 'snapcraft.yaml': Project assets
    will be ..
Launching a VM.
Launched: snapcraft-gforth-mtrute
.....
============== INSTALL SUCCEEDED ==============
Bash users: type 'hash -r' to empty the cache
Staging gforth
Priming gforth
Snapping 'gforth-mtrute' \
Snapped gforth-mtrute_0.7.9-20190501_amd64.snap
$
```

Damit entsteht ein knapp 4 MByte großes File.

Snapstore

Heutzutage hat alles einen Store; Snaps sind da keine Ausnahme. Wie auch die Docker–Hub, kann der *Snapstore* die Pakete selbst aus den Quellen bauen, sofern sie bei github liegen. Soweit, so bekannt. Der Snapstore schlägt auch eine Namenskonvention vor, wie man die Snaps benennen sollte. Man kann entweder den Programmnamen verwenden, sofern man der primäre Autor ist, oder schon im Snapnamen verdeutlichen, dass man eine abgeleitete Fassung hat und setzt seinen Nutzernamen dahinter. Nun wollte ich den Gforth–Autoren nichts verbauen, also heißt das `gforth` im Snapstore `gforth-mtrute`.

```
$ snap search gforth
Name Version Herausgeber Hinweise Zusammenfassung
gforth-mtrute 0.7.9-20190501 mtrute - gforth snap
$
```

Wenn man das Bauen der Snaps dem Snapstore überlässt, wartet der mit einer doch netten Überraschung auf: Er baut nicht nur für das normale 64–Bit–Linux, sondern gleich für ein halbes Dutzend Plattformen, darunter auch eher Exoten wie *S390* — hat jemand so etwas daheim?

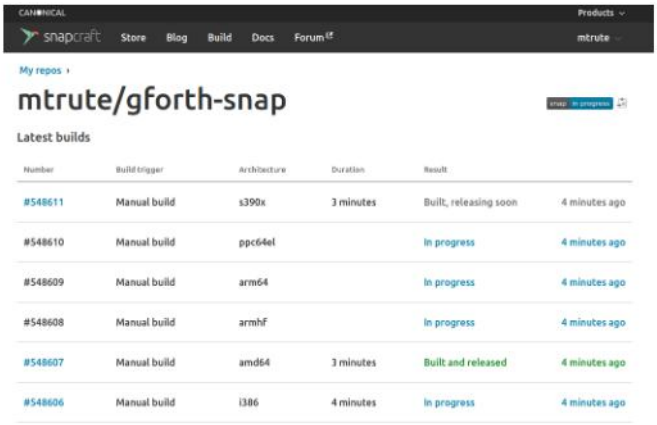


Abbildung 1: Snap–Report

Soviel sei verraten, es werden nicht alle *grün*. Aus nicht ganz nachvollziehbaren Gründen scheitern auch mal Plattformen, die schon funktioniert haben. Die wichtigen tun jedoch.

Einsatz

Snap ist als Paketierer gestartet, der erste Schritt ist also installieren. Die offensichtliche Variante ist das selbst erzeugte Snap–File.

```
$ snap install gforth-mtrute_0.7.9-20190501
_amd64.snap \ --devmode
```

Dafür muss man nicht mal lokale Adminrechte nutzen. Den `devmode` braucht man, um alle Sicherheitsprüfungen zu umgehen. Sehr praktisch, wenn man experimentiert.

Snaps haben vier verschiedene Qualitätslevel. Damit kann der Nutzer steuern, welche Versionen er nutzen möchte, sofern der Anbieter diese bereitstellt: *Edge* ist dasjenige Level, bei dem man mit allem rechnen muss. Etwas besser im Leumund ist *beta*. Dann kommen *candidate* und zu guter Letzt *stable*. Gforth ist derzeit nur als edge vorhanden, die anderen Level müssen noch reifen; das hat aber nichts mit der Qualität von Gforth zu tun.

```
$ snap info gforth-mtrute
name:      gforth-mtrute
summary:   gforth snap
publisher: Matthias Trute (mtrute)
license:   GPL-3.0+
description: |
  This is gforth snap
snap-id: cXf2H05AAoRzw6TOwmRucmuqHqkPJ0YQ
channels:
  stable:    -
  candidate: -
  beta:      -
  edge:      0.7.9-20190501 2019-05-04 (9) 3MB -
$ snap install --edge gforth-mtrute
gforth-mtrute (edge) 0.7.9-20190501
    from mt (mtrute) installed

$ snap list
Name          Version         Rev   Tracking Publisher      Notes
core          16-2.38.1       6818  stable   canonical*     core
core18        20190409        941   stable   canonical*     base
gforth-mtrute 0.7.9-20190501  3     edge     mtrute         -
```

Mit dem so installierten Gforth kann man dann loslegen:

```
$ /snap/bin/gforth-mtrute.gforth
Gforth 0.7.9_20190501, Copyright (C) 1995-
Gforth comes with ABSOLUTELY NO WARRANTY;
Type ‘help’ for basic help
version-string   ok 2
type 0.7.9_20190501 ok
bye
$
```

Dies und Das

Das Snap hat einige Einschränkungen. Die sind rein willkürlich und dem experimentellen Charakter zuzuschreiben. So kann es zum Beispiel nur auf Dateien aus dem eigenen `HOME`–Directory zugreifen.

```
$ cat huhu
.( huhu home )
$ cat /tmp/huhu
.( huhu tmp)
$ gforth-mtrute.gforth
s" huhu" included huhu home ok
s" /tmp/huhu" included
*the terminal*:6: error: No such file or directory
s" /tmp/huhu" >>>included<<<
Backtrace:
           0 $7F4C7053F3F8 throw
bye
```

Jeder Snapbuild zählt eine Versionsnummer hoch. Das ist soweit auch ganz praktisch. Man sollte nur aufpassen, wenn man Snaps mit identischem Namen aus unterschiedlichen Quellen installiert. Dann überschreibt schon mal rev.1 die rev.19 und beim Löschen sind dann beide weg.

Der Plan für die Zukunft ist, das gforth–Snap analog zum Docker–gforth aktuell zu halten und den Snapshots der Autoren zu folgen. Feedback und Vorschläge sind natürlich willkommen.

Verweise

`https://snapcraft.io/gforth-mtrute` Gforth als Snap

`https://github.com/mtrute` Quellen für die Container und Snaps

`https://wiki.ubuntuusers.de/snap/` Snaps kurz erläutert

`https://hub.docker.com/r/mtrute/gforth-container/` Gforth als Docker–Container

`https://savannah.gnu.org/git/?group=gforth` GIT Repository von Gforth

`https://www.complang.tuwien.ac.at/forth/gforth/Snapshots/` Gforth Snapshot

`https://en.wikipedia.org/wiki/Snappy_(package_manager)`

Erweiterung des Adressbereiches im Forth-System

Willi Stricker

Zunächst ein kurzer Rückblick in die Mikroprozessor– und Forth–Historie, um zu zeigen, warum mich das Thema beschäftigt: Die ersten Mikroprozessoren besaßen einen 8–Bit–Datenbus und einen 16–Bit–Adressbus, dazu 8–Bit–Datenregister und 16–Bit–Adressregister und dementsprechend einen Adressraum von $2^{16} = 64$ KByte. Für kleinere Programme war das ausreichend. Dann folgten 16–Bit–Prozessoren, also solche mit 16–Bit–Datenregistern, mit zunächst ebenfalls 16–Bit–Adressregistern. Erst später folgten 32–Bit–Prozessoren (32–Bit–Daten- und Adressregister) mit dem entsprechend größeren Adressraum von $2^{32} = 4$ GByte. Speziell bei den 16–Bit–Prozessoren zeigte sich, dass wegen deren 16–Bit–Adressregistern der Adressraum von 64 KByte in der Praxis oft zu klein war. Es wurden dann zusätzliche Maßnahmen erforderlich, um den Adressraum zu erweitern, ohne noch breitere Adressregister machen zu müssen. Dabei wurden zwei unterschiedliche Verfahren benutzt: Entweder wurden einige „erweiterte Register" eingeführt (z. B. Motorola 68000, „Long–Adressierung") oder es gab neue Segmente dazu, die den gleichen Adressraum hatten, aber über Segmentregister erreicht werden konnten (z. B. Intel 8086, Segmentierung des Adressraumes, vier Segmentregister).

Das Forth–System, und damit auch ein Forth–Prozessor, kennt im Gegensatz zu anderen Programmiersprachen und Prozessoren prinzipiell nur eine einzige Datengröße (Bitanzahl) für alle Operanden wie Zahlen, Flags oder Adressen. Das gilt auch für die internen Adressregister, die „Pointer", wie den Stackpointer und den Instructionpointer. Das bedeutet für das Forth–System und für einen Forth–Prozessor ebenso: Eine Erweiterung durch „Long–Befehle" oder „Long–Register" ist nicht möglich. Bleibt nur die Segmentierung und damit die Einführung von Segmentregistern.

Segmentierung für ein Forth–System bzw. einen Forth–Prozessor

Im Prinzip wäre nur ein einziges Segmentregister nötig, mit dem der gesamte Programm– und Adressbereich erweitert wird. In der Praxis sind jedoch Programme und Daten im Adressraum nahezu immer hardwareseitig und damit räumlich getrennt. Etwa in RAM und ROM (Flash–ROM). Deswegen werden zwei Segmentregister vorgesehen, das *Code–Segmentregister* `CS` und das *Daten–Segmentregister* `DS`. Ein Stack–Segment ist nicht nötig, da die Stacks im Forth–Prozessor hardwareseitig getrennt werden und deren Adressraum ausreichend sein sollte.

Adressbildung mit Segmentregistern

Die Adresse einer Stelle bei segmentiertem Speicher besteht immer aus zwei Komponenten, dem Segmentregister und dem Offsetregister. Bei Forth ist das Offsetregister immer der Instructionpointer und falls vorhanden, auch ein internes Daten–Adressregister. Bei einer System-Bitanzahl (sb) und einer gewünschten Adresserweiterung, den Extension–Bits (ext), gilt:

adr = ext + sb

mit $0 < \text{ext} \leq \text{sb}$ und der Segment–Bitanzahl zwischen 1 und sb.

Dabei gibt es verschiedene Möglichkeiten der Gestaltung.

Volle aneinander grenzende Segmente

Das Segmentregister erhält die Bitanzahl **ext**, was der Anzahl an Segmenten entspricht.

Die ersten Hardware–Forth–Prozessoren (NOVIX, RTX) waren 16–Bit–Prozessoren, hatten die vorgenannten Adressprobleme und beide benutzten die Segmentierung mit vollen nicht überlappenden Segmenten.

Überlappende Segmente

Hierbei besitzen das Segmentregister und das Offsetregister beide die gleiche Bitanzahl **sb**. Die Gesamtsumme ergibt sich aus der Summe von Offsetregister und einem um **ue** Bits (Überlappung) nach rechts verschobenem Segmentregister. Das führt zu überlappenden Segmenten, wie z. B. bei der Intel 8086–Familie. Mit **ue = sb - ext** beträgt die Gesamt–Adress–Bitanzahl dann **adr = ext + sb** oder **adr = 2*sb − ue**. Die Überlappungs–Bitanzahl liegt zwischen 1 und sb.

Für das Forth–System ist die Überlappung günstiger, da bei ihr beide Register die gleiche Systembreite **sb** besitzen; außerdem ist das System flexibler. Es soll deswegen benutzt werden.

Software

Alle Befehle, die den Programm–Ablauf steuern, benutzen das Code–Segment:

```
CALL (RS: -- adr)
RETURN (RS: adr -- )
BRANCH ( adr -- )
?BRANCH ( adr f -- )
```

Nur die Daten–Zugriffsbefehle benutzen das Daten–Segment:

```
@ ( adr -- x ) „Fetch"
! ( x adr -- ) „Store"
```

Alle anderen Befehle kommunizieren über Daten auf dem Stack. Sie sind daher unabhängig vom aktuellen Speicher–Segment.

Zusätzliche Befehle

Grundsätzlich müssen die beiden Segmentregister frei zugänglich und manipulierbar sein.

Für das *Daten–Segment* werden dazu zwei neue Befehle benötigt:

```
DS@ ( -- DS ) Fetch Data-Segment
DS! ( DS -- ) Store Data-Segment
```

Ein *Code–Segment–Fetch* und *–Store* ist nicht möglich, aber auch nicht sinnvoll! Deswegen wird ein anderer Weg gewählt: Das Code–Segment wird mit dem Inter–Segment–Call[1] gesetzt und durch den Inter–Segment–Return zurückgesetzt. Ich nenne das „call extended“ und „return extended“. Diese erfordern die Übergabe von Segment und Offset.

```
CALLX ( cs1 os1 -- R: cs0 os0)
```

Der call extended legt das aktuelle Segment– und Offset–Wertepaar `cs0 os0` auf dem Returnstack ab und ruft dann die neue Adresse auf, bestehend aus Segment `cs1` und Offset `os1` auf dem Datenstack.

Ein return extended würde dann das Wertepaar vom Returnstack holen und die zugehörigen Register damit laden.

```
;X (  R: cs0 os0 -- )
```

Dass es auch ohne solch ein explizites Extended–Return geht, indem man ein Segment–Flag benutzt, ist weiter unten beschrieben.

Programm–Ablauf

Das Programm startet immer im Code–Segment, es wird beim Booten initialisiert. Ein Wechsel des Code–Segmentes ist nur durch einen `CALLX`–Befehl möglich, ein Rücksprung nur durch einen `;X`–Befehl. Datenaufrufe werden vom Daten–Segment bewirkt. Es muss initialisiert werden, kann aber jederzeit während des Programmablaufes mit Hilfe von `DS!` verändert werden.

Besonderheit beim STRIP Forth–Prozessor: Das Segment–Flag

Die im FPGA integrierten RAMs stellen meist ein zusätzliches Bit–(n+1) in den Speicherplätzen bereit, z. B. Bit–16 und ein 17. Bit bei 16–Bit–Speichern. Dieses Bit wird als „Segment–Flag“ oder „Segment–Bit“ `SB` benutzt. Der Befehl `CALLX` setzt dieses Bit in der Rücksprungadresse.

Und jedes Return fragt immer auch das Segment–Bit ab. Für `SB = 0` wird ein „normaler“ Return–Befehl ausgeführt. Dann wird nur der *top of return stack* in den Instructionpointer geladen. Für `SB = 1` wird auch das Segmentregister eingelesen. Auf diese Weise entfällt der explizite Return–Extended–Befehl sogar! Man sieht, es werden nur 3 zusätzliche Befehle für das Arbeiten mit Segmenten gebraucht:

```
DS@,
DS!
CALLX
```

Damit können Unterprogramme, die Forth–Words, sowohl intra- als auch intersegmental aufgerufen werden. Der Return erfolgt automatisch gemäß dem Segment-Flag ins gleiche oder ein anderes Segment.

Ein praktisches Beispiel für überlappende Segmente

Für ein Forth–System mit der Operand–Größe sb = 16 Bit gilt:

Segmentregister = Offsetregister = 16 Bit

Erweiterung ext = 10 Bit, Überlappung: 16-10 = 6 Bit (=> 64 Bytes)

Das ergibt einen 26–Bit–Adressraum, entsprechend 64 Megabytes, mit 16-Bit breiten Segmenten, also 64–KByte große Segmente. Die Überlappung bewirkt, dass die obersten 64 Byte jedes Segmentes gleich den untersten 64 Byte des nächst höheren Segmentes sind.

Exkurs

Ich meine, es macht Sinn, hier einmal ein paar Begriffe zu klären. Und dann in einem Minibeispiel der Sache weiter auf den Grund zu gehen, um den STRIP–Forth–Prozessor besser zu verstehen.

Einerseits sind da Begriffe, die sich auf die Technik beziehen, andererseits gibt es welche für die logischen Operationen. Man muss aufpassen, in welchem begrifflichen Bereich man sich bewegt. Im Jargon wird da oft nicht sauber getrennt, und man muss aus dem Kontext erschließen, ob „Speicher“ nun dessen technische Realisierung oder seine logische Bedeutung meint.

Technische Begriffe

Speichermodul Hardware mit den Speicherzellen. Meist byteweise organisiert. Das ganze System kann technisch ganz verschieden ausgelegt sein: RAM, ROM, FRAM, serielle Typen. „Hauptspeicher“.

[1] Intel Prozessoren kennen Inter–Segment–Calls als CALLF und RETF — „call far“ und „return far“. Dabei werden code segment CS und instruction pointer IP auf dem Returnstack hinterlegt.

Speicherinterface Die Schaltung, mit der eine Zelle im Speichermodul ausgewählt und dann geschrieben oder gelesen werden kann.

Adressraum Anzahl der Speicherzellen insgesamt in einem Speichermodul.

Effektivadresse: Adresse, mit der letztlich auf den Speicher zugegriffen wird. Also das, was das Speicherinterface letztendlich benutzt.

Adressbusbreite Anzahl der Leitungsbahnen im Adressbus. Repräsentiert durch die Anzahl der Bits im Bustreiber — kurz „Bus“ genannt.

Register Vom Hauptspeicher getrennte kleinere Speichereinheiten, die mit der ALU verbunden sind. Die ALU liest daraus und gibt ihre Ergebnisse dorthin zurück.

Stack In einer nativen Stackmaschine (Forth) ist die ALU direkt mit dem Stack verbunden statt mit Registern. Der Stack ist technisch gesehen auch eine kleinere, vom Hauptspeicher getrennte Speichereinheit.

Registerbreite Anzahl der Bits in den verarbeitenden Registern einer CPU. Meist wird nur „Register“ gesagt und vorausgesetzt, dass man dessen Breite kennt: 8–Bit–, 16–Bit–, 32–Bit– oder 64–Bit–Maschine.

Stackbreite Identisch mit der Registerbreite. Auch beim „Stack“ wird meist vorausgesetzt, dass man schon weiß, welche Bitbreite die Maschine hat.

Die Stackbreite kann abweichen von der Busbreite. So sind kleinere MCUs heute meist 16–Bit–Maschinen, können aber einen breiteren Adressbus haben. Dann muss ein größerer Adressraum bedient werden und man steht vor dem Problem, dass der Stack oder die Register so eine große Adresse gar nicht fassen können.

Logische Begriffe

Speicher Für die Adressberechnungen verwendete logische Abstraktion vom Speichermodul.

Speicheradressierung Logische Operationen, um eine Speicherstelle auszuwählen. Die technische Umsetzung macht das Speicherinterface.

Speicheradresse Die logische Position einer Speicherstelle; kurz „Adresse“.

Speichersegment Eine feste Anzahl von aufeinander folgenden Adressen im Speicher. Kurz „Segment“.

Segmentgröße Anzahl der Adressen (Bytes), die innerhalb eines Segments liegen.

Segmentadresse Index, der das Segment bestimmt.

Segmentanfang Adresse, an der ein Speichersegment im Speicher beginnt.

Adressoffset Anzahl an Adressen vom Segmentanfang an gezählt; der Index in das Speichersegment hinein; kurz „Adresse“ oder „Offset“.

Verknüpfung von Technik und Logik, Speicher und CPU

Um den Hauptspeicher an die CPU anzubinden, braucht es ein *Speicherinterface*. Die CPU schreibt die Adresse und die Zugriffsoperation, lesen oder schreiben, dort hinein und das Interface erledigt das Gewünschte im Speicher. Es nimmt den zu speichernden Wert aus einem Übergaberegister und legt den zu lesenden Wert dorthin zurück. Diese Werte sind oft ganze Bytes, also 8–Bit breite Werte. Die können auf den Stack einer 16–Bit–Maschine gegeben werden, das passt. Aber wenn es eine 26–Bit–Adresse wird, wie im STRIP–Forth-Prozessor, passt es nicht mehr, da muss man tricksen. Intel hatte das gleiche Problem mit der Speicheradressierung in seinen früheren Maschinen.

Der Trick liegt darin, den Hauptspeicher in logische Stücke einzuteilen: die Segmente. Die Segmentgröße entspricht dann praktischerweise der Anzahl an Adressen, die mit einem Stackwert noch dargestellt werden kann. Bei einer Stackbreite von 16 Bit sind das dann $FFFF Adressen, also 65535 Byte oder 64 KByte je Segment.

Der Trick der überlappenden logischen Segmente

Um die effektive Adresse für das Speicherinterface zu erzeugen, braucht man folglich zwei Register. Eines für die Segmentadresse und eins für die Adresse darin, den Offset. Wie man diese mit dem Speicherinterface verbindet, ist dann die nächste Designentscheidung. Gern wird dann „überlappt“.

Dazu wird das Segmentregister SR ebenso wie das Adressregister AR in der CPU–eigenen Breite, also z. B. 16 Bit, gemacht. Der Adressbus im Speicherinterface kann aber z. B. 26 Bit breit gemacht werden, um mehr Speicher anzusprechen. Und ein Addierwerk im Speicherinterface macht daraus dann die effektive Addresse `ea` auf dem Bus, die x-Bits. (Abb.1)

25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
s	s	s	s	s	s	s	s	s	s	s	s	s	s	s	s											SR
										a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	AR
+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	add
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	ea

Abbildung 1: Überlappung der Register

Wie man sieht, kommt das einem logischen

```
SR 10 LSHIFT AR |+bus|
```

gleich. Wobei das `|+bus|` andeuten soll, dass die Addition nicht auf dem Stack erfolgt, sondern im Speicherinterface, und das Ergebnis an den Bus ausgegeben wird.

Eine solche Shift–Operation von 16–Bit kann man sich auf dem Stack in Forth ansehen.

```
hex .s <1> FFFF ok
decimal 10 lshift
2 base ! .s <1> 11111111111111110000000000 ok
hex .s <1> 3FFFC00 ok
```

Das lässt sich in *Gforth* natürlich nur so schön simulieren, weil es auf einer 64–Bit–Maschine ist.

```
00  00  00  00  SR     01  01  01  01  SR
 00  01  10  11 AR      00  01  10  11 AR
+++ +++ +++ +++ add    +++ +++ +++ +++ add
000 001 010 011 bus    010 011 100 101 bus
  0   1   2   3 ea       2   3   4   5 ea

10  10  10  10  SR     11  11  11  11  SR
 00  01  10  11 AR      00  01  10  11 AR
+++ +++ +++ +++ add    +++ +++ +++ +++ add*
100 101 110 111 bus    110 111 000 001 bus
  4   5   6   7 ea       6   7   0   1 ea

    (* Überlauf fällt weg, roll over)
```

Tabelle 1: Kombinationen

Wie man in Abb. 1 auch sehen kann, gibt es bei dieser Realisation der Adressierung mittels überlappender Register auch Kombinationen mit gleichwertigen Ergebnissen. Denn man bekommt dieselbe effektive Adresse auf den Bus, egal ob man im überlappenden Bereich die s–Bits oder die a–Bits dafür nimmt.

Ich habe das mal für eine 2–Bit–Maschine durchexerziert, die einen 3–Bit–Bus haben soll. Tabelle 1 zeigt die möglichen Kombinationen für die effektive Adresse `ea`.

Da ist leicht zu erkennen, dass mit dem 3–Bit–Bus zwei Segmente adressiert werden können, Segment–0 mit den effektiven Adressen 0…3 und Segment–1 mit den effektiven Adressen 4…7. Damit können also effektive Adressen von 0…7 eingestellt werden. Und der Wert im Adressregister wirkt immer nur als Offset zum Segmentanfang. Die kontinuierliche Adressierung `0 1 2 3 4 5 6 7` gelingt dann, wenn nur das oberste Bit des `SR` benutzt wird. Angenommen, ein Segmentzähler läge auf den Stack. Dann wäre der Segmentanfang dieser Mini–Maschine immer bei:

```
TOS 2 * SR !
```

Ausgerüstet mit dieser Einsicht ist es vermutlich besser möglich zu verstehen, was bei der Speicheradressierung passiert. Mir ist es jedenfalls so ergangen.

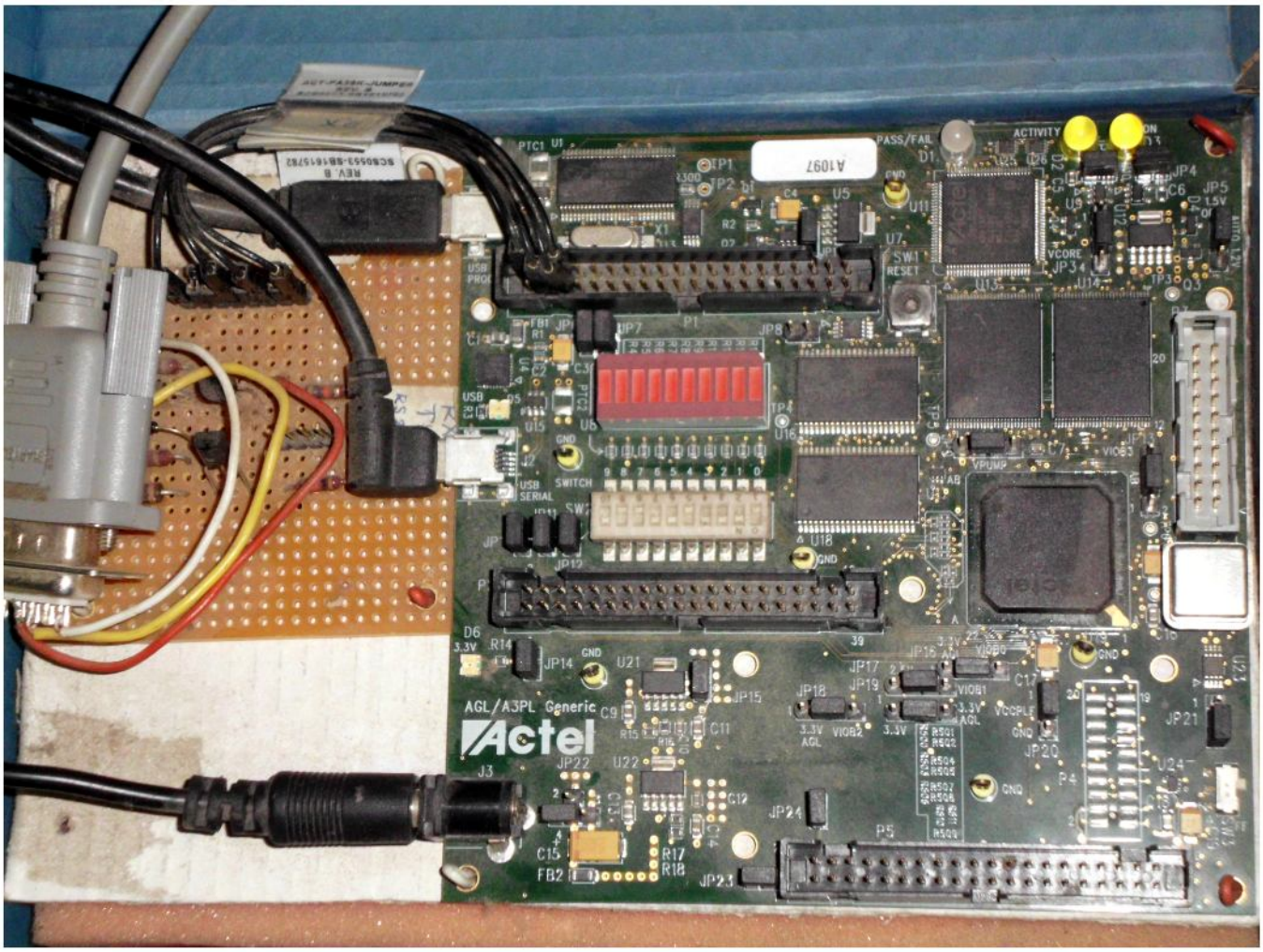


Abbildung 2: Der STRIP–Prozessor im Evaluation–Board; links oben die RS232–Schnittstelle.

postpone — entmystifiziert anhand von Beispielen

Michael Kalus

Ein Forth–Wort, das postpone verwendet, funktioniert wie ein Makro. Führt man es aus, compiliert es das, was in der zukünftigen Definition sein soll.

> "Append the compilation semantics of name to the current definition. Thats it."

sagte Alex neulich auf `comp.lang.forth`. Stimmt — eigentlich. Doch ist das gar nicht so einfach zu verstehen, wenn man gerade erst dabei ist, Forth kennen zu lernen. Er fuhr fort:

> "You use the same language for defining and processing macros, not a separate preprocessing language and processor. Consequently, the full power of Forth is available in macro definitions. E. g., you can perform arbitrarily complex computations, or generate different code conditionally or in a loop..."
>
> (Quelle: `https://www.complang.tuwien.ac.at/forth/gforth/Docs-html/Macros.html#Macros`)

So ist es. Doch schauen wir uns mal näher an, wie das gemeint ist.

Im Gforth–Tutorial wird erklärt, wie man `POSTPONE` verwenden kann, allerdings in sehr abstrahierter Form. Daher hier ein praktischeres Beispiel. Hier ahme ich mal das Forthwort `."` nach. Und danach folgt noch ein Beispiel aus einer echten Anwendung mit solch einer schon ziemlich "arbitrarily complex computation". Also etwas, dass weg vom Forth–Kern–Selbermachen führt.

Das erste Beispiel: `_."`

```
: _." ( ccc" -- )
[char] " parse
postpone sliteral
postpone type
; immediate
```

Hm, was passiert da?

Schritt für Schritt

Das `."` macht bekanntlich sowas:

```
: HALLO ." world" ; <ret>
ok
HALLO <ret>
world ok
```

Da wird ein neues Wort `HALLO` compiliert. Dabei wird der String[1] `world` ins neue Wort compiliert. Und zur Laufzeit gibt `TYPE` den String `world` im Terminal aus. Aber dazu muss `."` schon Vorwissen haben! Es muss wissen, was später mit dem String geschehen soll. Das Wort `."` muss also nicht nur den nachfolgenden String einlesen und compilieren, sondern darüber hinaus auch noch all das hinzu compilieren, was dereinst mal mit dem String gemacht werden soll. Also auch die Terminal–Ausgabe. Was ja durch den Punkt vor den Anführungszeichen symbolisiert werden soll.

Nun machen wir das nach.

Wie bringt man Forth dazu, den String `world` aus dem `TIB`[2]zu holen und in eine neue Definition zu compilieren? Das geht so:

```
                .--- Kommentar ----.
                |                  |
                |                  |
[CHAR] " PARSE ( ccc<"> -- adr1 n )
```

Diese Sequenz legt den String `ccc` ins RAM ab und hinterlässt dessen RAM–Adresse adr1 und seine Länge n auf dem Datenstack.

```
SLITERAL ( adr1 n1 -- ) ( RT: -- adr n )
```

compiliert diesen String dann ins Wörterbuch und gibt Ort und Länge davon später, zur Laufzeit RT[3], auf dem Datenstack zurück. Sodass man mit

```
TYPE ( adr n -- )
```

den String dort holen und drucken kann. Soweit so gut, das ist der Ablauf.

Doch zu dem Zeitpunkt, an dem `_."` definiert wird, ist noch gar kein solcher String da! Der kommt ja erst, wenn `_."` seinerseits in einer Definition benutzt wird. Erst dann kann der String gespeichert werden. Und erst dann kann angegeben werden, was damit später geschehen soll. Der Compilevorgang muss also verschoben werden, bis dass es soweit ist — `postpone`.

Also schreibt man:

```
[char] " parse
postpone SLITERAL
postpone TYPE
```

[1] Zeichenkette

[2] Terminal Input Buffer

[3] run time

Warum man [char] nehmen muss und nicht das char selbst, das ist wieder eine andere Geschichte.

Natürlich kann das nicht einfach so im Raum stehen, es soll ja eine Definition werden. Also muss noch der Compiler drum herum gesetzt werden.

```
.-- starte den Compiler
| .-- gebe der Definition ihren Namen
| |
v v
: _." ( ccc<"> -- )
[char] " parse
postpone literal
postpone type
;          <-- beende den Compiler
immediate <-- und noch ein Trick hintendran!
```

Die Klammer (ccc<"> --) ist wieder nur ein Kommentar. Der zeigt, dass zur Laufzeit ein String ccc erwartet wird, der bis zum Delimiter–Zeichen " geht. Der Trick am Ende, das immediate, bewirkt, dass die soeben erzeugte Definition selbst dereinst unmittelbar ausgeführt werden wird. Sie wird dann innerhalb der zukünftigen Definition ablaufen, statt eincompiliert zu werden.

Nun erweitern wir das Beispiel noch um die Zwischenfunktion upper[4], mit der man einen String in Großbuchstaben wandelt.

```
: _." ( ccc<"> -- )
[char] " parse
postpone literal
postpone upper
postpone type ;
immediate
```

Damit würde unser _." den String später in Großbuchstaben drucken. Doch zur besseren Faktorisierung würde man zunächst das Wort

```
: utype ( c-addr u -- ) upper type ;
```

definieren und es dann im _." verwenden.

```
: _." ( ccc<"> -- )
[char] " parse
postpone literal
postpone utype ;
immediate
```

Denn auch utype selbst ließe sich noch gut wiederverwenden:

```
: _.(    ( ccc<)> -- )
  [char] ) parse utype ; immediate
```

Was ganz nützlich ist, um Kommentare bereits während einer laufenden Interpretation von Quellcode so zwischendurch im Terminal auszugeben.

Tatsächlich existiert im Gforth das .(bereits, was genau das macht.[5]

Das zweite Beispiel: keymap

Dieses Beispiel, das freundlicherweise Matthias Koch beigesteuert hat, stammt aus einem Projekt, bei dem die komfortable Belegung der Tasten des Numworks–Taschenrechners gescannt wird. Der komplette Code ist im Anhang gelistet. Dort findet ihr eine hübsche Anwendung von postpone. Um die verschiedenen Kombinationen von Tastendrücken lesen und an ein Terminal ausgeben zu können, sind eine Menge von Vergleichsoperationen zu machen. Je nachdem, ob die Modus–Tasten *SHIFT* oder *ALPHA* gedrückt sind oder nicht, sind unterschiedliche Aktionen erforderlich. Diese lassen sich zu Gruppen ordnen. Das ergibt schließlich die keymap. Darin werden key-press–Serien und die modulierten Tastenbelegungen erkannt und ausgeführt. postpone dient dort dazu, die komplizierte Compilation dieser keymap übersichtlich zu halten. Dazu wird eine spezielle Datenstruktur keym generiert, die dann den Entscheidungsbaum compiliert — so etwas wie eine Tabelle, in der man nachschaut, was gerade an Tasten gedrückt ist, um dann der Vorschrift entsprechend zu handeln. Was in diesem Fall bedeutet, entweder einzelne Zeichen oder kurze Zeichenketten zu senden, je nach Tastendruck. Das Beispiel ist in Mecrisp verfasst. Der Tastatur–Scanner arbeitet im Multitasker und schickt die auslesende MCU schlafen, wenn nichts zu tun ist.

Falls ihr euch über die Notation der Aktion–Strings wundert, sowas wie sin"asin"g"G" hinter dem keym, das sind "–separierte Strings. Der Parser von Mecrisp holt das ganze „Wort“ sin"asin"g"G" in den TIB. Von dort liest keym viermal nacheinander die Substrings ein, immer bis zum nächsten "–Delimiter. Ein cooler Trick. Wenn man weiß, wie Mecrisp oder sein jeweiliges Forth-System innerlich funktioniert, kommt man auf sowas. :)

An der beispielhaften Zeile :

```
.--- Sollwert           .---Action-String
|                       |
|                       |
v                       v
$00000000.00040000 keym e^x"["a"A"
```

soll nun die Arbeitsweise von postpone im keym erklärt werden. Ihr seht, da kann nun einfach der Scan–Sollwert auf den Datenstack gelegt werden, den keym zum Vergleich heranziehen soll. Und danach wird der Action–String geholt. keym kümmert sich dann selbst darum, all das, was damit nun geschehen soll, an Ort und Stelle hinzuzukompilieren. Da wird dann später geprüft werden, ob ein gültiger Tastendruck da war, und wenn ja, welcher. Um dann den passenden inject-string zu „normal“, „shift“, „alpha“ oder „shiftalpha“ zu machen. Die einzelnen Action-Teile dafür hat sich keym aus dem Action-String geholt und passend zurechtgeschnitten.

Ihr müsst zugeben, dass das sehr unübersichtlich und fehleranfällig wäre, müsste man diese verzweigte Datenstruktur im Quellcode manuell anlegen. Da ist es schon

[4] Sofern upper in deinem Forth–System schon definiert ist.

[5] Siehe auch Heft 4d2002-02, Berichte generieren mit Hilfe von Gforth — ein Kinderspiel

besser, sich das Defining-Word dafür zu überlegen und anzulegen. Der `inject-string` arbeitet übrigens ganz ähnlich wie das _." aus unserem ersten Beispiel. Es holt sich seinen String und gibt den aus — in diesem Fall an einen Ringbuffer, über den die MCU mit dem Terminal kommuniziert.

Historisches

Das Wort `POSTPONE` wurde mit dem Forth–Standard von 1994, dem *ANS–94*[6], eingeführt.

Damals wurde sowas als typische Nutzung von `POSTPONE` angesehen:

```
: ENDIF POSTPONE THEN ; IMMEDIATE
: X ... IF ... ENDIF ... ;
```

`POSTPONE` ersetzt die meisten Funktionen von `COMPILE` und `[COMPILE]`. `COMPILE` und `[COMPILE]` wurden für den gleichen Zweck verwendet: Verschieben des Kompilierungsverhaltens des nächsten Wortes im `TIB`[7] in die Zukunft. `COMPILE` war da, um auf non–immediate Worte angewendet zu werden, während `[COMPILE]` auf die Immediate–Worte angewendet werden muss. Dies belastete den Programmierer damit wissen zu müssen, welche Worte in einem Forth–System immediate sind und welche nicht. Infolgedessen musste das im Forth–Standard spezifiziert sein. Das schränkte aber Implementierer unnötigerweise ein. Denn man kann bei der Implementation eines Forth–Systems auf unterschiedlichen Wegen zum gleichen Ergebnis kommen.

Ein zweites Problem bei `COMPILE` war, dass einige Programmierer eine besondere Art der Implementierung des `COMPILE` erwarteten und ausnutzten, nämlich:

```
: COMPILE R> DUP @ , CELL+ >R ;
```

Diese Implementierung funktionierte aber nicht auf allen Forth–Systemen. In einem Nativ–Code–Forth mit Inline–Code–Expansion und Peephole–Optimierung variiert die Größe des produzierten Objektcodes. Es ist schwierig, diese Informationen an ein „dummes“ `COMPILE` zu übermitteln. Ein „smartes“ `COMPILE` hätte dieses Problem nicht, aber dies war in früheren Normen noch verboten. Aus diesen Gründen wurde `COMPILE` nicht mehr in den Standard aufgenommen und `[COMPILE]` wurde zugunsten von `POSTPONE` verschoben.

Zum Beispiel:

```
COMPILE 2*
```

Das scheint erstmal richtig zu sein. Nur eben da nicht, wo `2*` selbst als ein Compiling–Wort definiert ist, wie etwa in *cmForth*, wo es Inline–Instruktionen generiert oder die aktuelle Instruktion modifziert, so dass der Effekt von `2*` entsteht. Daher besser:

```
POSTPONE 2*
```

denn das hat dieses Problem nicht. `POSTPONE` findet selbst raus, was zu tun ist, es ist smarter[8].

So, und nun: Fröhliches fortheln allerseits!

Dank

Mein Dank geht besonders an Matthias Koch, der das Source–Code–Beispiel beigesteuert hat, und an Ulrich Hoffmann für seine kritischen Anmerkungen und Hinweise beim Verfassen dieses Beitrages. Danke auch an all die engagierten Forther von `comp.lang.forth`, die meine Fragen beantwortet haben.

Links

`https://www.complang.tuwien.ac.at/forth/gforth/Docs-html/POSTPONE-Tutorial.html#POSTPONE-Tutorial` `http://forth.org/svfig/Win32Forth/DPANS94.txt`

Listing

```
\ -------------------------------------------------------
\  RGB LED
\ -------------------------------------------------------

PC7 constant led-red
PB1 constant led-green
PB0 constant led-blue

: init-led ( -- )
  led-red ioc! led-green ioc! led-blue ioc!
  OMODE-PP dup led-red   io-mode!
           dup led-green io-mode!
               led-blue  io-mode!
;

\ -------------------------------------------------------
\  Keyboard scan
\ -------------------------------------------------------

\  The keyboard is a matrix that is laid out as follow:
\
\       |  PC0 |  PC1 | PC2  | PC3  | PC4  | PC5  |
\  -----+------+------+------+------+------+------+
\   PE0 | K_A1 | K_A2 | K_A3 | K_A4 | K_A5 | K_A6 |
\  -----+------+------+------+------+------+------+
\   PE1 | K_B1 | K_B2 |   8  |   9  |  10  |  11  |
\  -----+------+------+------+------+------+------+
\   PE2 | K_C1 | K_C2 | K_C3 | K_C4 | K_C5 | K_C6 |
\  -----+------+------+------+------+------+------+
\   PE3 | K_D1 | K_D2 | K_D3 | K_D4 | K_D5 | K_D6 |
\  -----+------+------+------+------+------+------+
\   PE4 | K_E1 | K_E2 | K_E3 | K_E4 | K_E5 | K_E6 |
\  -----+------+------+------+------+------+------+
\   PE5 | K_F1 | K_F2 | K_F3 | K_F4 | K_F5 |  35  |
\  -----+------+------+------+------+------+------+
\   PE6 | K_G1 | K_G2 | K_G3 | K_G4 | K_G5 |  41  |
\  -----+------+------+------+------+------+------+
\   PE7 | K_H1 | K_H2 | K_H3 | K_H4 | K_H5 |  47  |
\  -----+------+------+------+------+------+------+
```

[6] American National Standard for Information Systems, Programming Languages, Forth. Die gedruckte offizielle Fassung war das ANSI X3.215-1994.

[7] or any Parse Area

[8] Das Thema wurde damals diskutiert in: Hayes, J.R., "Postpone", Proceedings of the 1989 Rochester Forth Conference.

```
\   PE8 | K_I1 | K_I2 | K_I3 | K_I4 | K_I5 |  53  |
\  -----+------+------+------+------+------+------|
\
\   We decide to drive the rows (PE0-8)
\   and read the columns (PC0-5).
\
\   To avoid short-circuits, the pins E0-E8 will not
\   be standard outputs but only open-drain.
\   Open drain means the pin is either driven low
\   or left floating.
\   When a user presses multiple keys, a connection
\   between two rows can happen.
\   If we don't use open drain outputs,
\   this situation could trigger a short circuit
\   between an output driving high
\   and another driving low.
\
\   If the outputs are open-drain, this means
\   that the input must be pulled up.
\   So if the input reads "1", this means the key is
\   in fact *not* pressed,
\   and if it reads "0" it means that there's a short
\   to an open-drain output.
\   Which means the corresponding key is pressed.

: init-keyboard ( -- )

   %000000000 16 lshift %111111111 or
    PE0 io-base GPIO.BSRR +  !
    \ All floating

   OMODE-OD
   \ Set PE0 to PE8 as open drain outputs
    PE0 %111111111 io-modes!
   \ Set PC0 to PC5 as inputs with pullups
    IMODE-HIGH PC0    %111111 io-modes!

\  OMODE-OD
\  dup PE0 io-mode!
\  dup PE1 io-mode!
\  dup PE2 io-mode!
\  dup PE3 io-mode!
\  dup PE4 io-mode!
\  dup PE5 io-mode!
\  dup PE6 io-mode!
\  dup PE7 io-mode!
\      PE8 io-mode!

\  IMODE-HIGH
\  dup PC0 io-mode!
\  dup PC1 io-mode!
\  dup PC2 io-mode!
\  dup PC3 io-mode!
\  dup PC4 io-mode!
\      PC5 io-mode!
;

: scan-keyboard-row ( ud -- ud* )
  6 2lshift  swap  10 us
  PC0 io-base GPIO.IDR + @ not
  %111111 and  or  swap
;

: scan-keyboard ( -- ud )

  0.

   %100000000 16 lshift %011111111 or
    PE0 io-base GPIO.BSRR +  !  scan-keyboard-row
   %010000000 16 lshift %101111111 or
    PE0 io-base GPIO.BSRR +  !  scan-keyboard-row
   %001000000 16 lshift %110111111 or
    PE0 io-base GPIO.BSRR +  !  scan-keyboard-row
   %000100000 16 lshift %111011111 or
    PE0 io-base GPIO.BSRR +  !  scan-keyboard-row
   %000010000 16 lshift %111101111 or
    PE0 io-base GPIO.BSRR +  !  scan-keyboard-row
   %000001000 16 lshift %111110111 or
    PE0 io-base GPIO.BSRR +  !  scan-keyboard-row
   %000000100 16 lshift %111111011 or
    PE0 io-base GPIO.BSRR +  !  scan-keyboard-row
   %000000010 16 lshift %111111101 or
    PE0 io-base GPIO.BSRR +  !  scan-keyboard-row
   %000000001 16 lshift %111111110 or
    PE0 io-base GPIO.BSRR +  !  scan-keyboard-row
   %000000000 16 lshift %111111111 or
    PE0 io-base GPIO.BSRR +  !
    \ All floating to reduce current consumption
;

: scan-keyboard-debounce ( -- ud )
  0. 16 0 do scan-keyboard 2or loop
;

\ -------------------------------------------------------
\  Ring buffers by Jean-Claude Wippler
\ -------------------------------------------------------

\ Ring buffers, for serial ports, etc
\ - size must be 4..256 and power of 2
\ TODO setup is a bit messy right now,
\ should put buffer: word inside init

\ Each ring needs 4 extra bytes for internal
\ housekeeping:
\   addr+0 = ring mask, i.e. N-1
\   addr+1 = put index: 0..255
\            (needs to be masked before use)
\   addr+2 = get index: 0..255
\            (needs to be masked before use)
\   addr+3 = spare
\   addr+4..addr+4+N-1 = actual ring buffer, N bytes
\ Example:
\   16 4 + buffer: buf  buf 16 init-ring

: init-ring ( addr size -- )
  \ initialise a ring buffer
  1- swap !
  \ assumes little-endian so mask ends up in ring+0
;

: c++@ ( addr -- b addr+1 )
  \ fetch and autoinc byte ptr
  dup c@ swap 1+ ;

: ring-step ( ring 1/2 -- addr )
  \ common code for saving and fetching
  over +    ( ring ring-g/p )
  dup c@ >r ( ring ring-g/p R: g/p )
  dup c@ 1+ swap c!  \ increment byte under ptr
  dup c@ r> and swap 4 + + ;

: ring# ( ring -- u )
  \ return current number of bytes in the ring buffer
  \ TODO could be turned into a single @ word access
  \ and made interrupt-safe
  c++@ c++@ c++@ drop - and ;
: ring? ( ring -- f )
  \ true if the ring can accept more data
  dup ring# swap c@ < ;
: >ring ( b ring -- )
  \ save byte to end of ring buffer
  1 ring-step c! ;
: ring> ( ring -- b )
  \ fetch byte from start of ring buffer
  2 ring-step c@ ;

\ -------------------------------------------------------
\  Terminal injection
\ -------------------------------------------------------

\ Character buffer for "key"
```

```
128 4 + buffer: character-ring

: inject-character ( -- )
  character-ring ring?
  if
    character-ring >ring
  else
    drop \ Forget characters which would overflow
         \ the ring buffer.
  then
;

: inject-string ( addr len -- )
  0 ?do
    dup c@ inject-character
    1+
  loop
  drop
;

: ring-key? ( -- ? )
  serial-key? if serial-key inject-character then
  character-ring ring# 0<>
;

: ring-key ( -- c )
  begin ring-key? until
  character-ring ring>
;

: enable-ring ( -- )
  character-ring 128 init-ring
  ['] ring-key  hook-key  !
  ['] ring-key? hook-key? !
;

: disable-ring ( -- )
  ['] serial-key  hook-key  !
  ['] serial-key? hook-key? !
;

\ -------------------------------------------------------
\  Sleep mode
\ -------------------------------------------------------

false variable just-woke-up

: sleep-a-while ( -- )

  \ cr ." Entering sleep mode..." cr

  singletask
  led-green ioc!

  \ Insert a power down sequence here,
  \ perhaps cutting clock to peripherals
  \ Simplest possibility:
  \ Divide clock down to a very low frequency.

  \ Prepare the row which carries the power button
  %000000010 16 lshift %111111101 or
  PE0 io-base GPIO.BSRR +  !
  10 us

  \ Simulate behaviour by busy waiting for PC1
  \ going low, begin PC1 io@ not until

  \ Insert a power up sequence here,
  \ reenabling and initialising peripherals

  true just-woke-up !
  led-green ios!
  multitask

  \ cr ." Welcome back !" cr cr
;

\ -------------------------------------------------------
\  Key mapping
\ -------------------------------------------------------

0. 2variable    last-keys
0. 2variable current-keys

: key-press ( xd -- ? )
  2dup current-keys 2@ 2and d0<>
  -rot    last-keys 2@ 2and d0=
  and
;

: key-release ( xd -- ? )
  2dup current-keys 2@ 2and d0=
  -rot    last-keys 2@ 2and d0<>
  and
;

false variable shift-lock
false variable alpha-lock

: normal?     ( -- ? )
  shift-lock @ not alpha-lock @ not and ;
: shift?      ( -- ? )
  shift-lock @     alpha-lock @ not and ;
: alpha?      ( -- ? )
  shift-lock @ not alpha-lock @     and ;
: shiftalpha? ( -- ? )
  shift-lock @     alpha-lock @     and ;

: keym
  postpone key-press
  postpone if

    postpone normal?
    postpone if
    postpone s"
    postpone inject-string
    postpone then

    postpone shift?
    postpone if
    postpone s"
    postpone inject-string
    postpone then

    postpone alpha?
    postpone if
    postpone s"
    postpone inject-string
    postpone then

    postpone shiftalpha?
    postpone if
    postpone s"
    postpone inject-string
    postpone then

  postpone then
  immediate
;

: keymap ( -- )

  \ ------------------------------------------
  $00000000.00000001 key-press
                     if 27 inject-character
                        91 inject-character
                        68 inject-character then
                     \ Cursor left
  $00000000.00000002 key-press
                     if 27 inject-character
                        91 inject-character
                        65 inject-character then
```

```
                    \ Cursor up
   $00000000.00000004 key-press
                    if 27 inject-character
                       91 inject-character
                       66 inject-character then
                    \ Cursor down
   $00000000.00000008 key-press
                    if 27 inject-character
                       91 inject-character
                       67 inject-character then
                    \ Cursor right
   \ ----------------------------------------------
   $00000000.00000010 key-press
                    if 27 inject-character
                       91 inject-character
                       72 inject-character then
                    \ OK --> Pos1
   $00000000.00000020 key-press
                    if 27 inject-character
                       91 inject-character
                       70 inject-character then
                    \ Return key --> End
   $00000000.00000040 key-press
                    if 4 @ execute then
                  \ Home --> Restart Mecrisp-Stellaris
   $00000000.00000080 key-release
                    if just-woke-up @
                      if false just-woke-up !
                      else sleep-a-while
                      then
                    then
                    \ Power button --> Sleep mode
   \ ----------------------------------------------
   $00000000.00001000 key-press
                    if -1 shift-lock xor! then
   $00000000.00002000 key-press
                    if -1 alpha-lock xor! then
   $00000000.00004000 keym x,n,t"cut":":"
   $00000000.00008000 keym var"copy";";"
   $00000000.00010000 keym Nut&Bolt"paste"'"'"
   $00000000.00020000 key-press
                    if 8 inject-character then
                    \ Backspace
   \ ----------------------------------------------
   $00000000.00040000 keym e^x"["a"A"
   $00000000.00080000 keym ln"]"b"B"
   $00000000.00100000 keym log"{"c"C"
   $00000000.00200000 keym i"}"d"D"
   $00000000.00400000 keym ,"_"e"E"
   $00000000.00800000 keym x^y"sto>"f"F"
   \ ----------------------------------------------
   $00000000.01000000 keym sin"asin"g"G"
   $00000000.02000000 keym cos"acos"h"H"
   $00000000.04000000 keym tan"atan"i"I"
   $00000000.08000000 keym Pi"="j"J"
   $00000000.10000000 keym sqrt"<"k"K"
   $00000000.20000000 keym ^2">"l"L"
   \ ----------------------------------------------
   $00000000.40000000 keym 7"7"m"M"
   $00000000.80000000 keym 8"8"n"N"
   $00000001.00000000 keym 9"9"o"O"
   $00000002.00000000 keym ("("p"P"
   $00000004.00000000 keym )")"q"Q"
   \ ----------------------------------------------
   $00000010.00000000 keym 4"4"r"R"
   $00000020.00000000 keym 5"5"s"S"
   $00000040.00000000 keym 6"6"t"T"
   $00000080.00000000 keym *"*"u"U"
   $00000100.00000000 keym /"/"v"V"
   \ ----------------------------------------------
   $00000400.00000000 keym 1"1"w"W"
   $00000800.00000000 keym 2"2"x"X"
   $00001000.00000000 keym 3"3"y"Y"
   $00002000.00000000 keym +"+"z"Z"
   $00004000.00000000 keym -"-" " "
   \ ----------------------------------------------
   $00010000.00000000 keym 0"0"?"?"
   $00020000.00000000 keym ."."!"!"
   $00040000.00000000 keym x10^x"x10^x"x10^x"x10^x"
   $00080000.00000000 key-press
                    if 32 inject-character then
                    \ Space \ keym Ans"Ans"Ans"Ans"
   $00100000.00000000 key-press
                    if 10 inject-character then
                    \ LF    \ keym EXE"EXE"EXE"EXE"
   \ ----------------------------------------------
;

\ ------------------------------------------------------
\ ------------------------------------------------------

: keyboard-handler ( -- )
  scan-keyboard-debounce current-keys 2!

  current-keys 2@ last-keys 2@ 2xor
  d0<>   \ Any changes in key state ?
  if
    \ ." Key change: " current-keys 2@ hex. hex. cr
    keymap \ Insert a lot of handling here !
  then

  current-keys 2@ last-keys 2!
;

\ ------------------------------------------------------
\  Task for handling the keyboard in background
\ ------------------------------------------------------

task: keyboard-task

: keyboard& ( -- )
  keyboard-task activate

    begin
      keyboard-handler
      pause \ Pause is called in the delay
            \ used in scan-keyboard,
            \ therefore strictly speaking
            \ no pause is necessary here.
    again
;

: keyboard-into-terminal ( -- )

  init-led
  init-delay
  init-keyboard
  enable-ring

  led-green ios!

  multitask
  keyboard&
;
```

Unikernel und Forth

Carsten Strotmann

Die „Cloud“, also die vielen Server–Systeme der heute benutzten Internet–Dienste, sind erst durch Virtualisierung möglich geworden. Anwendungen und Betriebssysteme werden nicht mehr direkt auf der physischen Hardware installiert, sondern in virtuellen Maschinen basierend auf Virtualisierungs–Lösungen wie QEmu, VMWare, Virtualbox oder Hyper–V. Durch Virtualisierung können die Rechner–Ressourcen besser ausgenutzt werden, Anwendungen werden dynamisch zwischen physischen Rechner–Systemen verschoben, wenn die Auslastung es notwendig macht. Auf den meisten virtuellen Servern laufen die gleichen Betriebssysteme (Linux, Windows oder ein BSD–Unix), und jeder Server verbraucht (neben seiner Anwendung) für diese Betriebssysteme mehrere Megabyte Hauptspeicher und Gigabyte Plattenspeicher. Bei vielen tausend Servern ein großer Verbrauch durch redundanten Programm–Code.

Unikernel

Eine Lösung des Problems sind Unikernel. Bei einem Unikernel wird die Anwendung direkt mit einem Betriebssystem–Kern in einer Einheit verbunden. Wikipedia[1] definiert den Begriff „Unikernel“ als

> A unikernel is a specialised, single address space machine image constructed by using library operating systems. A developer selects, from a modular stack, the minimal set of libraries which correspond to the OS constructs required for their application to run. These libraries are then compiled with the application and configuration code to build sealed, fixed-purpose images (unikernels) which run directly on a hypervisor or hardware without an intervening OS such as Linux or Windows.

Die Anwendung wird in einem Unikernel–System zu einem Teil des Betriebssystems. Unikernel–Systeme sind im Vergleich zu traditionellen Betriebssystemen sehr schlank (einige wenige Megabyte), können aber trotzdem direkt in einem Hardware–Virtualisierer oder sogar auf echter Hardware gestartet werden. Unikernel Systeme sind heute meist für die 64–Bit Intel/AMD–CPU–Architekturen verfügbar.

Vorteile von Unikernel-Systemen

Die Vorteile eines Unikernel–Systems gegenüber klassischen Betriebssystemen sind:

- Schneller Programmstart — es muss kein volles Betriebssystem mit vielen Hintergrund–Diensten gestartet werden, sondern nur die Unikernel–Anwendung. Eine Unikernel–Anwendung startet auf moderner Server–Hardware innerhalb weniger Sekunden
- Kein Kontext–Switch zwischen Anwendung und Betriebssystem. Die Anwendung läuft im gleichen Sicherheitskontext wie die Kernel–Funktionen. Ein Wechsel des Sicherheitskontextes ist bei Betriebssystemen eine relativ aufwändige und langsame Operation. Während bei klassischen Betriebssystemen für Hardware–Zugriffe die Anwendung aus Ring–3 auf Ring–0 (Betriebssystem–Kernel) wechseln muss, läuft bei einer Unikernel Anwendung alles in Ring–0 ab.
- Eine Unikernel–Anwendung ist schlanker als ein volles Betriebssystem; es laufen nur Programmteile, die für die Anwendung benötigt werden. Daher ist die Angriffsfläche der Anwendung kleiner als bei traditionellen Systemen; auch müssen seltener Sicherheitslücken gepatched werden. Die regelmäßigen Betriebssystem–Updates sind nicht notwendig, da es kein klassisches Betriebssystem gibt.
- Die Anwendung hat vollen Zugriff auf die Hardware. Es sind daher, je nach Unikernel, auch harte Echtzeit–Anwendungen realisierbar.

Unikernel–Anwendungen ähneln von den Vorteilen her den *Bare–Metal*–Forth–Systemen wie *Color–Forth*, *FIG–Forth* oder dem vor ein paar Ausgaben vorgestellten *Eulex–Forth*.

Es ist daher interessant, Forth–Anwendungen als Unikernel zu betreiben. Die Forth–Anwendung kann auf einem traditionellen Betriebssystem entwickelt und getestet werden, um dann später als Unikernel benutzt zu werden. Dabei kann die Forth–Anwendung weiterhin auf die Dienste des Betriebssystems zurückgreifen, z. B. das TCP/IP–Netzwerk oder das Dateisystem.

Unikernel–Anwendungen erstellen

Um eine Unikernel–Anwendung erstellen zu können, wird erst das Unikernel–Entwicklungssystem auf einem traditionellen Betriebssystem wie Windows, Linux oder BSD installiert.

Die Anwendung, also das Forth–System selbst in diesem Falle, wird danach entweder mit einem speziell angepassten Compiler[1] zum Unikernel übersetzt. Oder es kann ein schon als Binärprogramm vorliegendes Forth–System zu einem Unikernel verlinkt werden. Bei diesem Schritt wählt der Entwickler die Module des Unikernels aus, welche mit in die Anwendung eingebunden werden sollen, z. B. Dateisystem–, Netzwerk– und Audio–Treiber.

Das Ergebnis ist dann ein Unikernel–Betriebssystem–Kern, welcher die Anwendung enthält. Diese Kernel–Datei kann dann in einer Virtualisierungs–Lösung (Hypervisor) gestartet und getestet werden.

[1] C oer C++ Compiler

Beispiele von Unikernel–Systemen

Es gibt viele verschiedene Unikernel–Systeme. Der verlinkte Wikipedia–Artikel enthält weiterführende Informationen. Drei Systeme werden nachfolgend vorgestellt.

rumprun

rumprun ist ein Unikernel basierend auf *NetBSD*. NetBSD ist eines der klassischen Unix–Betriebssysteme aus der BSD–Familie[3]. NetBSD war schon immer ein sehr modular aufgebautes Unix, welches auf viele verschiedene Hardware–Plattformen portiert wurde. rumprun bietet der Anwendung eine Standard POSIX–API (Unix Programmierschnittstelle); textbasierte Anwendungen für Linux/Unix sollten daher ohne größere Änderungen direkt im rumprun–Unikernel–System lauffähig sein.

Bei rumprun werden die Anwendungen mittels eines speziellen rumprun–Cross–Compilers in den Unikernel übersetzt.

Offiziell gibt es keinen Support für Tastatur–IO[2] in rumprun. Ein mit rumprun mitgeliefertes Beispiel, das Spiel *nethack*[4], implementiert jedoch Tastatur–IO, sodass auch ein interaktives Forth mit rumprun möglich sein sollte. rumprun kommt mit vielen Beispielen und einem Video–Tutorial, welches die Erstellung einer Unikernel–Anwendung erklärt[5].

IncludeOS

Einen anderen Ansatz, um zu einer Unikernel–Anwendung zu kommen, verfolgt IncludeOS[6]. IncludeOS, selbst in C++ geschrieben, verwandelt eine in C oder C++ geschriebene Anwendung in eine Unikernel–Anwendung. Dazu wird in den Quellcode die Unikernel–Funktionalität per `include` importiert:

```
/ Minimal, bootable IncludeOS application
#include <os>

int main() {
printf("IncludeOS successfully booted.")
}
```

Neben C und C++ sind für die Zukunft auch noch andere Sprachen geplant. Schon heute lassen sich Programmiersprachen mit IncludeOS verwenden, welche sich per *Transpiler* in die Sprache C umwandeln lassen. Bei Forth ist das z. B. mit 4th[7] möglich. Andere Beispiele sind *Nim*, *Pascal*, *Modula2* und *Oberon*.

IncludeOS beinhaltet Treiber für Tastatur und VGA–Grafik; damit lassen sich interaktive Forth–Systeme als Unikernel übersetzen.

ops und nanos

Wiederum einen anderen Ansatz bietet *ops* mit dem *nanos*–Unikernel[8]. Bei ops wird eine im Binärformat (ELF) vorliegene Linux–Anwendung mit Teilen des Linux–Kernels zu einem Unikernel verlinkt. So können Anwendungen in beliebigen Sprachen, also auch Skript–Sprachen wie *PHP*, *Python* oder *Ruby*, in Unikernel eingebaut werden.

ops kommt mit einem eigenen Paketsystem, in dem Basisanwendungen für Unikernel vorbereitet sind. In diesem Paketsystem ist GNU/Forth (Gforth) schon enthalten. Der interessierte Forth–Entwickler kann mit diesem System sofort loslegen[9]:

```
ops load gforth_0.7.3 -a hello.fth
Extracting /home/cas/.ops/packages/
    gforth_0.7.3.tar.gz to \
/home/cas/.ops/.staging/gforth_0.7.3
[gforth hello.fth]
booting /home/cas/.ops/images/gforth.img ...
assigned: 10.0.2.15
Hello World!
exit status 1
```

Warum Forth als Unikernel-Anwendung?

Der Unikernel–Ansatz ermöglicht es dem Forth–Entwickler, auf modernen Systemen hardwarenah zu entwickeln. Die Zielsysteme für Unikernel sind in der Regel Virtualisierungs–Hypervisoren, jedoch lassen sich die Unikernel auch auf echter Hardware ausführen.

Der Entwickler hat mehr Kontrolle über die Ausführung der Anwendung (Realtime–Eigenschaften), die Anwendung startet schnell und verbraucht wenig Speicher verglichen mit einem vollen modernen Betriebssystem. Trotzdem ist die Anwendung portabel, kann mit den normalen Entwicklungswerkzeugen unter einem traditionellen Betriebssystem entwickelt und getestet werden und muss nicht auf Betriebssystem–Dienste wie Netzwerk verzichten.

Unikernel sind sicher keine Universallösung für alle Anwendungsfälle, aber ein interessantes neues Werkzeug für die Forth–Handwerker.

Referenzen

[1] `https://en.wikipedia.org/wiki/Unikernel`

[2] `https://github.com/rumpkernel/rumprun` und `http://rumpkernel.org/`

[3] `https://en.wikipedia.org/wiki/Berkeley_Software_Distribution`

[4] `https://github.com/anttikantee/rumprun-nethack`

[5] `https://github.com/rumpkernel/wiki/wiki/Tutorial:-Rumprun-unikernel-video-series`

[6] `https://www.includeos.org/`

[7] `https://thebeez.home.xs4all.nl/4tH/`

[8] `https://github.com/nanovms/ops`

[9] `https://nanovms.com/dev/tutorials/running-forth-unikernels`

[2] Unikernel–Awendungen sind für Dienste in Rechenzentren ausgelegt, dort arbeitet niemand an der Tastatur–Konsole der Rechner.

35 Jahre Forth–Gesellschaft

Karsten Roederer

Jetzt ist es 35 Jahre her seit der Gründung der Forth–Gesellschaft. Andrea und Ewald Rieger waren so freundlich und richteten die Tagung im wunderschönen Worms am Rhein aus. Sie entdeckten dabei so ganz nebenbei den „kleinen Bruder“ des Swap–Drachens, den Drop–Drachen. Für uns zweidutzend Leute hatten die beiden ein sehr angenehmes und entspanntes Programm zusammengestellt. Wobei der Wettergott, als wir draußen in Worms auf den Spuren Luthers und seiner Verfolger unterwegs waren, es ein bisschen auf die Teilnehmer abgesehen hatte. Ich hoffe, es hat sich keiner erkältet.

Andrea kümmerte sich rührend um die Forth–Passivisten, während die fleißigen Forth–Gesellschafter tagten. Das Kernprogramm reichte von anspruchsvoll bis leicht verständlich und das Aha–Erlebnis stellt sich bei mir manchmal erst nach der Durchsicht der von Bernd gemachten Video–Aufnahmen und Präsentationen ein.[1]

Gut besucht war auch die Platinen–Layout–Programm–Präsentation von Erich Wälde am Abend, er gab eine Einführung in KiCAD. (Als frisch gebackener Funkamateur (DD5DW) hatte ich den Erich (DL7TUX) letztens auf einem Foto von FUNKEN–LERNEN entdeckt und konnte feststellen, dass auch Ulli (DL5LU) die A–Zulassung besitzt. Und das sind wohl nicht die einzigen unter uns, oder? Gut zu wissen, wie bei einem eventuellen Ausfall des Internets die Kommunikation aufrechterhalten werden kann.[2] Bei FUNKEN–LERNEN hat Erich neulich einen bleibenden Eindruck hinterlassen. Er hat es versucht und ich nun auch, doch haben wir es (noch?) nicht geschafft, „den Klaus“ als Forth–Mitglied zu gewinnen.

An einem Abend gab es Flammkuchen in der Bier- und Weinschänke. Die großen Portionen hat wohl keiner geschafft. Das uns zugewiesene Gewölbe hatte etwas von „unter den Talaren, der . . .“, also ganz lutherischer Zeitgeist. Zu guter Letzt rückte Andrea den Swap–Drachen schweren Herzens wieder heraus, um ihn an Matthias Koch, der allerdings nicht anwesend war, weiterzugeben. Klaus Zobawa nahm ihn stellvertretend entgegen und erklärte sich bereit, ihn heile nach Hannover zu bringen.

Alles in allem eine angenehme Tagung, die mir sehr gefallen hat. Und das Haus Sandwiese war klasse ausgewählt: feine Zimmer, bester Service, gute Kost, prima Tagungsräme mit allem, was man brauchte, WLAN inklusive.

Danke, Andrea und Ewald.

Abbildung 1: Tagungsteilnehmer in Worms vor dem Hotel Sandwiese am „gelben Drop“

[1] Findet ihr im Wiki der FG.

[2] Als DARC–Net, sozusagen. (der Sätza :-)

Forth–Gruppen regional

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

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

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

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

Gruppengründungen, Kontakte

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

μP–Controller Verleih

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

Spezielle Fachgebiete

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

Termine

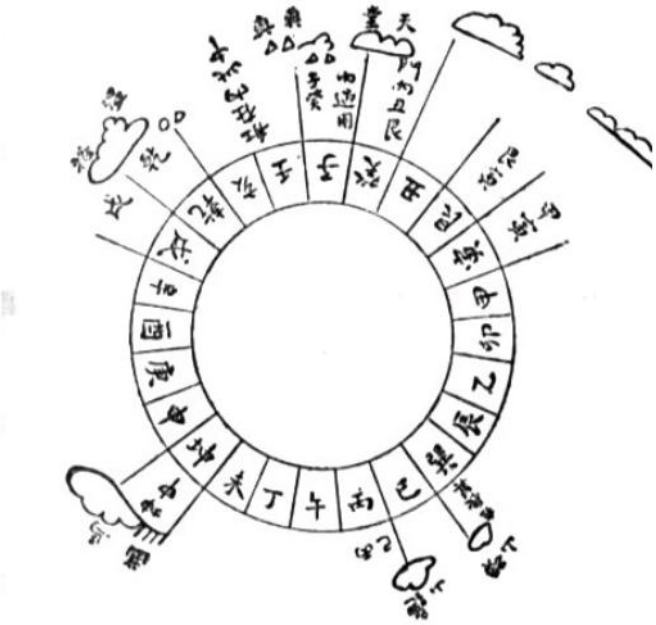
Donnerstags ab 20:00 Uhr
Forth–Chat net2o forth@bernd mit dem Key keysearch kQusJ, voller Key:
kQusJzA;7*?t=uy@X}1GWr!+0qqp_Cn176t4(dQ*

17.–18.08.2019
Maker Faire Hannover
http://www.makerfairehannover.com

21.–25.08.2019
Chaos Communikation Camp
https://events.ccc.de

13.–15.09.2019
EuroForth
http://www.euroforth.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.

Einladung zur
EuroForth 2019 vom 13. bis 15. September 2019
in Hamburg

The 35th EuroForth conference takes place in Hamburg/DE.

`http://euro.theforth.net/`

The conference will be preceeded by the Forth standards meeting which starts on September, 11th.

Both, meeting and conference will be hosted in the H4 Hotel Hamburg–Bergedorf.

The exact address for your navigation system is:
Holzhude 2, 21029 Hamburg, DE.

Programme

Forth standard meeting

Wednesday, 11th September 13:00 –
Friday, 13th September 12:00

EuroForth conference

Friday, 13th September 13:00 –
Sunday, 15th September 14:00

Saturday: Hamburg excursion

Besides having wonderful discussions with other splendid Forthers, we will roam Hamburg intensively saturday afternoon and have our conference dinner at saturday evening. See the image to the right to get some impressions.

Bring your partners!

As usual this years conference will also feature a separate track for a Forther's partner, including but not limited to: Enjoying a great view over Hamburg from the Elbphilharmonie Plaza, visiting Hamburg city hall, shopping at the famous Jungfernstieg and Mönkebergstraße.

Images:

1. St. Michaelis Church
   anonymous / GNU FDL + CC BY–SA 3.0 unported
2. Landungsbrücken
   User Batintherain on en.wikipedia / public domain DE
3. Hamburg Elbphilharmonie
   Specialpaul / CC BY–SA 4.0 Int
4. Speicherstadt
   Dietmar Rabich / Wikimedia Commons / "Hamburg, Speicherstadt, Wasserschloss – 2016 – 2971" / CC BY–SA 4.0