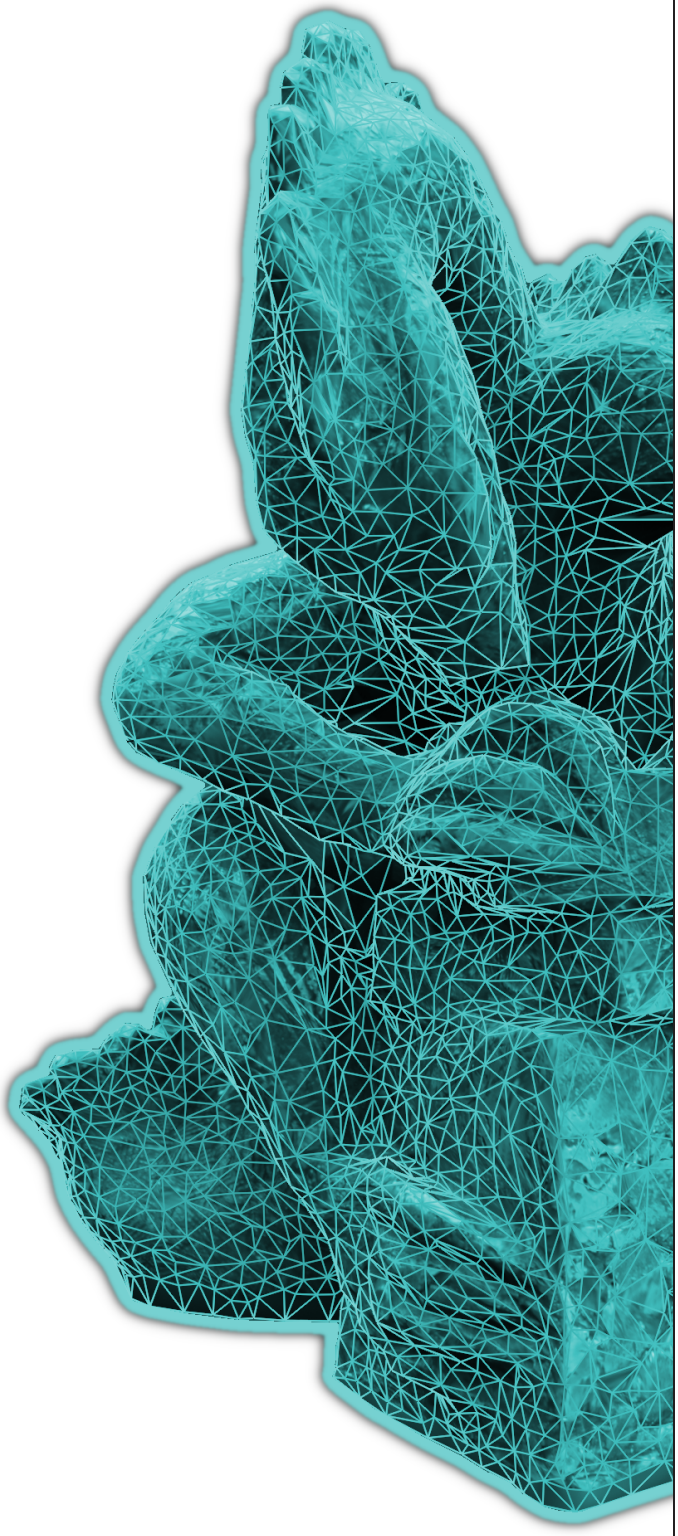
Vierte Dimension

1/2022

Das Forth–Magazin

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

In dieser Ausgabe:



Titelbild: CyberSwappy AUTOREN: Amend & Wodni
Quelle: eigenes Werk; nach den Scan–Daten vom originalen SWAP.

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 1030
48481 Neuenkirchen
Tel: +49(0)6239 9201-85, Fax: -86
E–Mail: Secretary@forth-ev.de
Direktorium@forth-ev.de
Bankverbindung: Postbank Hamburg
BLZ 200 100 20
Kto 563 211 208
IBAN: DE60 2001 0020 0563 2112 08
BIC: PBNKDEFF

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00 € + Porto u. Verpackung

Liebe Leser,

der *CyberSwappy* vom Titelblatt ist eine Gemeinschaftsarbeit. Friedel Amend hat neulich den Swap von allen Seiten fotografiert. Sein Sohn Bernd hat daraus ein *3D–Modell* erstellt, musste allerdings einiges manuell nachbearbeiten, da aus den Fotos heraus nicht alle Flächen perfekt erkannt worden sind. Gerald Wodni hat dann diese Scandaten in seinen Slicer geladen und das *Mesh–Bild* erstellt. Und damit Swappy etwas „cyberiger“ daherkommt, wurde das Originalmesh noch drastisch reduziert, die jungen Leute nennen so etwas „Low Poly“. Die türkise Farbe bekam er im GIMP. So ziert CyberSwappy nun die Anmeldeseite für die *Forth–Videokonferenzen*.

Und weil der vernetzte SWAP als Symbol auch gut zum Thema *Wireless Mesh Network* passt, das Willem Ouwerkerk und Henny Luijkx beigesteuert haben, lugt er da nun um die Ecke, um den Freunden von Forth beim Werkeln zuzusehen.

Heftig weitergewerkelt hat inzwischen auch Jörg Völker am *FlexiFloat*, seiner radikalen Abkehr vom reinen Integer der Forth–Sprache. Wozu ich die Daumen drücke, das würde mir so manches erleichtern.

Doch auch die ganz kleinen Forthsysteme für die ganz kleinen MCUs gibt es weiterhin. Dr. Chen–Hanson Tings *eForth* ist so eins, und für diverse MCUs zu haben. Mein Favorit fürs Basteln ist immer noch das *TI Launchpad* mit der *Forth Dream Machine*, einem MSP mit FRAM. Manfred Mahlow bastelt damit nicht, sondern macht ernsthafte Sachen. Seine Vorarbeit erlaubte einen weiteren Port des eForth auf den kleineren FRAM–Typ von TI.

Nochmal zu den *Forthtreffen* in Videokonferenzen: Weil man dafür ja nicht reisen muss, könnten die eigentlich auch öfters im Jahr stattfinden. Wolfgang Strauss macht das ja schon einige Zeit jeden Montag für eine kleine eingeschworene Schar Forthler. Dort kam das Thema *Stringstack* auf, weshalb an dessen Funktion hier erinnert wird.

So Stacks sind leicht zu machen, im Prinzip. Dass das Drumherum dann doch tückisch sein kann, zeigt Bernd Paysan im Abschnitt *synonym vs. alias* kurz auf.

Und nun angenehme Lektüre.

Auf ein freudiges Wiedersehen, Montags, im Mai und im Sommer.

Bis dahin, euer Michael

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
http://fossil.forth-ev.de/vd-2022-01

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

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

Schwarze Schrift

Ein Anruf hat genügt und die Schrift war wieder schwarz.[1] Das Heft ist auch in dem gewohnten Zeitrahmen bei uns angekommen. Wenn ich die Qualität gleich beim ersten grauen Druck beanstandet hätte, wäre auch nur ein Heft mit grauer Schrift entstanden. Vor rund 10 Jahren gab es schon mal ein Heft mit gleichem Fehler. Übrigens finde ich den Kommentar in der VD über die Druckqualität nicht in Ordnung — wir drucken nun schon 13 Jahre im REPROZENTRUM MARQUARDT in Darmstadt.[2] Abgesehen von den 2 Pannen waren sonst die Hefte immer gut. Mich stört, wenn wir in aller Öffentlichkeit behaupten, dass die Firma Toner auf Kosten der Qualität spart, um sich an seinen Kunden zu bereichern (steht nicht so drin — kann man aber so reininterpretieren). Das hat der Betrieb garantiert nicht nötig und wäre auch ein schlechtes Geschäftmodell. Ewald Rieger

Abbildung 1: Dort in Darmstadt werden unsere Hefte gedruckt.

Liebe Firma Marquard, lieber Ewald, liebe Leser,

es war nie meine Absicht, jemanden herabzusetzen. Vielleicht wäre es besser gewesen, die Sache stillschweigend zu beheben, statt zu publizieren. Ich dachte halt, es könnte unsere Leser interessieren, dass (und wie) wir dann die Fehlersuche machen im Verein. Welche an sich ja sachlich und prima gelaufen ist. Der Fehler konnte dadurch eingegrenzt und behoben werden.

Im Heft Zitate zu verwenden, ist ein Stilmittel, welches ich schätze. Wenn dabei Einfälle laut werden, die sich dann ausschließen lassen — umso besser. Herabsetzende Propaganda liegt mir fern. Bitte verzeiht mir den journalistischen Eifer.

Ich danke allen Beteiligten für ihr Engagement. mk

Umzug des Forth–Büros

Abbildung 2: Umzug des Forth–Büros Ende Januar.

Nach knapp 14 Jahren haben Andrea und Ewald Rieger das Forth–Büro an Carsten Strotmann abgegeben. Ende Januar wurden etliche Kartons mit aktuellen und historischen Akten der Forth–Gesellschaft an das neue Forth–Büro übergeben. Die neue Adresse des Forth–Büros ist nun:
Forth–Gesellschaft e.V.
Postfach 1030
48481 Neuenkirchen

Die E–Mail–Adresse ist mit `secretary@forth-ev.de` gleich geblieben. Das Forth–Büro hilft bei Fragen rund um die Mitgliedschaft in der Forth–Gesellschaft, verwaltet die Finanzen des Vereins und ist für den Druck und Versand des Forth–Magazins „Vierte Dimension“ zuständig.

Wir sagen vielen Dank an Andrea und Ewald für viele Jahre gute und verlässliche Arbeit für die Forth–Gesellschaft.

Historische Ausgaben der VD

Das Forth–Büro besitzt noch einen Schatz an historischen Ausgaben unseres Forth–Magazins „Vierte Dimension“ (VD). Die können von Mitgliedern unter der Angabe der Mitgliedsnummer nachbestellt werden. Preise (zzgl. Versandkosten):

Einzelhefte: 1 € pro Heft

10 Ausgaben: 8 €

20 Ausgaben: 14 €

Die Liste der noch verfügbaren Ausgaben findet ihr in unserem Wiki:
`https://wiki.forth-ev.de/doku.php/vd-archiv#historische_ausgaben`

Carsten Strotmann / Forthbüro

[1] Zur Erinnerung: In 1/2021 – 3/2021 war die Schrift der gedruckten Hefte gerastert; siehe Leserbrief Heft 4/2021.

[2] `https://repro-marquardt.de`

Krise

Corona, Erderwärmung und Artensterben sind die vorherrschenden Krisen, aber nicht die einzigen.

Derzeit wird meine kleine Firma gleich von vier Seiten aus China torpediert:

- Über die Online–Versandriesen flutet billige China–Modellbauelektronik unkontrolliert in Massen nach Deutschland, zu Versandkosten, zu denen ich noch nicht einmal nach Hamburg liefern kann. (Wedel ist ein Vorort von Hamburg). Dadurch gerät das ganze Preisgefüge durcheinander.
- Wenn wir aber an unseren Händler in China liefern, werden die Sendungen vom chinesischen Zoll regelmäßig abgefangen, weil unser Händler dort eine spezielle Import–Lizenz haben muss, die er aus irgendeinem Grund nicht hat oder nicht bekommt (die Mengen, die wir da exportieren, sind marginal, es ist mehr ein Achtungserfolg, dass unsere Sachen auch in China verkauft werden können — aber trotzdem).
- Chinesische Chip–Broker schnappen uns alle Halbleiter weg, die wir dringend in der Produktion benötigen und bieten sie dann zu unverschämten Preisen wieder an. Die exorbitanten Spekulationsgewinne werden offensichtlich sofort genutzt, um weitere Halbleiter aufzukaufen. Wenn das so weitergeht, wird sich die gesamte Bauteildistribution demnächst in chinesischer Hand befinden.
- Und last, but not least, wurde gerade aktuell das zweite von uns entwickelte Soundmodul in China kopiert, und jetzt wird versucht, diese Plagiate mit einer Kopie auch von unserem Typenschild, der Anleitung und der Produktbeschreibung über eBay anzubieten. In den kommenden Monaten werden diese Plagiate überall auftauchen, das kennen wir schon. Soundmodule machen bei uns rund ein Drittel des Umsatzes aus.

Bei der Abwehr dieser Machenschaften werden wir kleinen Unternehmen hierzulande leider in keiner Weise unterstützt und chinesischen Anbietern einfach zum Fraß vorgeworfen, im krassen Gegensatz zur chinesischen Staatspolitik, die ihre Wirtschaft und Exporte in jeder erdenklichen Weise fördert und schützt. Auf Solidarität deutscher Verbraucher braucht man auch nicht zu hoffen, „Geiz ist geil“ ist die Devise. Das *Amt der Europäischen Union für geistiges Eigentum*[3] geht für 2019 von 60 Milliarden Euro an Umsatzverlust und von 470.000 betroffenen (d. h. verlorenen) Arbeitsplätzen allein durch Plagiate in der EU aus. 27 % der Bevölkerung halten den Kauf von Plagiaten demnach für akzeptabel. Tja, solange es nicht der eigene Arbeitsplatz ist …

Noch halten wir uns über Wasser, aber die Liste der untergegangenen deutschen Anbieter im Modellbau ist lang. Gerade in den letzten Jahren hat es auch die Traditionsfirmen *Robbe* und *Graupner* erwischt, es existieren nur noch die Markennamen, die pikanterweise von einem österreichischen Händler gekauft wurden. Am stärksten unter Druck sind im Moment die Hersteller von Modell–Baumaschinen, es ist nicht immer nur die Elektronik.

Was tun?

Gegen die Plagiate auf eBay können wir mit dem Markenrecht vorgehen und werden tatsächlich von eBay sehr gut unterstützt — das Entfernen von unrechtmäßigen Angeboten dauert nur wenige Stunden. Haben die Chinesen dann aber erst einmal das Typenschild gewechselt, haben wir keine richtige Handhabe mehr. eBay und auch das Angebot der europäischen Modellbauhändler müssen wir jetzt permanent überwachen, um überhaupt einschreiten zu können. Gegen das Ungleichgewicht im Handel sind wir machtlos. Und bei den Chips?

Die Krise zeigt, wie gefährlich es sein kann, sich auf einen Hersteller und eine Chip–Bauform festzulegen. Da es bei Mikrocontrollern kein Second–Source mehr gibt (nicht täuschen lassen, das gilt auch für ARM–Prozessoren, jeder Hersteller hat proprietäre Peripherie um den Prozessorkern herum gebaut) kann man diesem Risiko wohl nur mit einer Modulbauweise entgehen. Eine eng bestückte Platine auf einen anderen Controller oder eine andere Gehäusebauform umzubauen, bedeutet eigentlich immer ein komplettes Redesign. Isoliert man den Controller auf ein Modul, muss bei einem Hersteller– oder Gehäusewechsel eben nur das Modul neu entwickelt und ggf. die Software angepasst werden, schlimm genug. Leider ist eine Modulbauweise immer auch mit höheren Kosten verbunden, da kann schon mal die Steckverbindung teurer sein, als der Controller. Und es entstehen wieder neue Abhängigkeiten, auch Steckverbindungen können nicht lieferbar sein. Außerdem muss sich ein Modul am kleinsten gemeinsamen Nenner der vorgesehenen Chipvarianten orientieren. Und wenn man Pech (oder Glück) hat, je nach Sichtweise, ist man mit dem Moduldesign gerade fertig und plötzlich ist alles wieder lieferbar. So ist es vielen in der Kondensator–Krise vor ein paar Jahren ergangen, die in der Not ihre Baugruppen von Keramik–Kondensatoren wo immer möglich auf Tantal umkonstruiert hatten — kaum war man damit fertig, waren die keramischen MLCCs[4] zu halbwegs normalen Preisen und in gängigen Bauformen wieder da.

Erzwungene höhere Kosten, ständig steigender Aufwand, Plagiate — China schafft es, unsere Wettbewerbsfähigkeit immer weiter zu verschlechtern. Es ist ein *Wirtschaftskrieg*. Und wir haben nichts Besseres zu tun, als den Warenverkehr in unserer eigenen EU durch Verpackungsgesetz und Elektroschrottverordnungen und Batteriegesetz und was weiß ich noch alles (in jedem EU–Land natürlich anders geregelt und getrennt zu beantragen) auch noch zu behindern.

In den letzten Monaten wurde es noch schlimmer. Meine „chinesischen Freunde“ haben jetzt auch Leistungstransistoren für sich entdeckt, rechtzeitig, nachdem ich für

[3] EUIPO `https://euipo.europa.eu`

[4] Multilayer Ceramic Capacitors

die Controller für das nächste Jahr einen Notfahrplan erstellt habe. Auch da ist für unsere Fahrtregler so gut wie nichts mehr zu bekommen, ich muss auf schlechter spezifizierte Typen ausweichen, noch dazu auf drei verschiedene Typen, um die nötigen Stückzahlen überhaupt zusammenzubekommen. Wir brauchen rund 1500 Mosfets jeden Monat ...

Wie lange die derzeitige Chip–Krise anhalten wird, scheint niemand verlässlich vorhersagen zu können. Oft ist von Ende 2022 die Rede, Durchhalten ist also die Devise. Schade, dass man Firmen nicht einfach in den Winterschlaf versetzen kann, der Zeitpunkt wäre günstig. Eine Prise Extraschlaf würde auch mir gut tun. Jörg Völker

Nachtrag: Dieser Leserbrief erreichte die Redaktion im 3. Quartal 2021. Wie Jörg nun, im März 2022, mitteilte, sei die Lage bei der Bauteilbeschaffung durch die jüngste Krise (Ukraine) noch schlimmer geworden. (mk)

Excurs

Als *Krisen* bezeichnet der Krisennavigator, das Institut für Krisenforschung, ein „Spin–Off“ der Christian–Albrechts–Universität zu Kiel, alle internen oder externen Ereignisse, durch die akute Gefahren drohen für Lebewesen, für die Umwelt, für die Vermögenswerte oder für die Reputation eines Unternehmens bzw. einer Institution. Unterschieden werden drei Arten von Krisen: Bilanzielle Krisen („Pleiten“), kommunikative Krisen („Skandale“) und operative Krisen („Störungen“). Pro Jahr ereignen sich nach den Erhebungen des Instituts im deutschsprachigen Europa rund 25.000 bis 40.000 bilanzielle Krisen sowie ca. 250 bis 280 (öffentlich gewordene) operative und kommunikative Krisen. `https://www.krisennavigator.de`

Ins Deutsche wurde das Wort von der lateinischen *crisis* entlehnt, erst in medizinischen Zusammenhängen vor allem fieberhafter Erkrankungen, wo es die sensibelste Krankheitsphase bezeichnete, der bei glücklichem Verlauf der Infektion (damals noch ohne die Möglichkeit der Antibiotikagabe) eine Entfieberung innerhalb eines Tages folgte und die endgültige Krankheitsabwehr einläutete. Hippokrates nannte diese Phase die *Entscheidungstage*. (Quelle: Wikipedia) mk

eForth — neues Template für MSP430

Das `430eforth-x-43n7vis.asm` ist ein Naken–Assemblerlisting, basierend auf dem Originalskript von Dr. Chen Hanson Ting, wie er es in seinem Buch beschrieben hat [1]. Ursprünglich für den *IAR–Assembler* geschrieben, wurde es zunächst ins *Code Composer Studio (CSS)* übertragen und später von mir für den frei verfügbaren *Naken–Assembler* angepasst.

Manfred Mahlow fügte Flash–Tools und Unterstützung für WORDLISTs, VOCs (VOCABULARY Prefixes), ITEMs und STICKY Words (VIS) hinzu, damit man dieses kleine Forth auch allein für sich genommen, z. B. auf dem TI MSP430G2553–Launchpad, verwenden kann. Es braucht damit eigentlich keine weiteren externen Tools mehr, um interaktiv im Target direkt in Forth zu programmieren. Eigentlich, weil natürlich irgendein serielles Terminal als Verbindung zur MCU vorhanden sein muss. Wir empfehlen dafür das *e4thcom* von Manfred zu verwenden, weil es das am besten angepasste Forth–Terminal für interaktives Arbeiten im Target ist, das es gibt![5]

Diese neue Vorlage enthält also nun den Code für *zwei* tolle MSP430–Modelle:

- Die *eForth Dream Machine* — MSP430FR5969 (64 K FRAM)
- Das *eForth Reference System* — MSP430G2553 (16 K Flash)

Beide Launchpads sind ja weiterhin für kleines Geld bei TI zu bekommen.

Manfred hat die jeweiligen spezifischen Teile des Codes so markiert, das mittels der Assembler–Konstante `MCU` für das eingestellte Target assembliert wird. Das ist ganz simpel und gut nachvollziehbar gestaltet, und verzichtet darauf, die spezifischen Teile in separate Dateien auszulagern, damit die innere Logik eines so einfachen Forthkerns möglichst nachvollziehbar bleibt.

Mit der Publikation dieses Listings sollen andere Forth–Benutzer ermutigt werden, sich das eForth anzueignen. Ihr werdet sehen, es sind nur wenige Stellen, die man da anpassen muss.

Das Projekt ist im Wiki der Forth–Gesellschaft abgelegt [4]. Ihr könnt uns gerne auch schreiben:
`mik.kalus@gmail.com`
`manfred.mahlow@forth-ev.de`

[1] Zen and the Forth Language: EFORTH for the MSP430 von Texas Instruments. Kindle Edition.
[2] Forth Magazin „Vierte Dimension“, Heft 4d2019–04, S. 11 ff — mit Glossar zum VIS und weiteren Links bzw. Literaturhinweisen. Im 4D–Archiv: `https://wiki.forth-ev.de/doku.php/vd-archiv`
[3] e4thcom — Ein Terminal für eingebettete Forth–Systeme `https://wiki.forth-ev.de/doku.php/projects:e4thcom`
[4] `https://wiki.forth-ev.de/doku.php/en:projects:430eforth-workbench:start` mk

Escom 0.1 — Forth Terminal für Win10

So, wie es aussieht, gibt es endlich auch für Windows ein Embedded–Forth–Terminal. Ende 2020 hat Ed Smallenburg es auf GitHub veröffentlicht, Update im Dezember 2021.

`https://github.com/Edzelf/escom`

Im Readme heißt es dazu:

> „`Escom` is a terminal program that runs under Windows. It can be used for communicating with an

[5] Ja, Linux! Neuerdings geht das auch unter Win10 mit dem *Escom* von „edzelf“ Ed Smallenburg genauso elegant.

embedded forth system, like `stm8ef` for the STM8 family or the `mecrisp` for the STM32 family. The connection between the Windows computer and the target Forth system is through a USB to Serial converter.

Escom is based on `e4thcom` by Manfred Mahlow. The program was converted to C for Windows compatibility.“

Wunderbar! Die wichtigsten Funktionen, die das *e4thcom* von anderen Terminals unterscheiden, sind im *Escom* nun auch implementiert, insbesondere das bedingte und unbedingte Hochladen von Quelltextdateien in das Forth–Zielsystem.

Die Unterstützung für weitere Zielsysteme, wie z. B. 430eForth, sollte sich recht einfach ergänzen lassen, so dass das 430eForth–Workbench–Projekt nun auch für Windows–User richtig interessant werden kann.

Und es gibt weitere Projekte, die bisher nicht so recht vorangekommen sind, weil eine einfache und praktische Entwicklungsumgebung für Windows fehlte.

Ich habe keine Gelegenheit, Escom zu testen, da ich ausschließlich Linux verwende. Also meine Bitte an alle interessierten Forth–Freunde der Windows–Welt: Testet das *Escom–Terminal*, berichtet über eure Erfahrungen und helft, es zu verbessern, falls das nötig sein sollte.

Manfred Mahlow

Making of Escom

Abbildung 3: Edzelf, forever young — like Forth ;)

Some time ago I discovered that it was possible to run Forth on very small microprocessors. Forth is an old computer language, developed for old computers with low memory. In that respect, modern microprocessors are quite comparable to the computers of that time.

The big advantage is that Forth has REPL (Read–Evaluate–Print–Loop). So you don’t need a development system and it is therefore suited for educational purposes.

I wanted to use Forth on microprocessors like the STM8 and the STM32.

On Hackaday I found an article about STM8 *eForth* and I installed the binary release of `stm8ef` on a STM8S103.

I tried the well–known *putty* program to contact the STM8. That worked. After that I found Mahlow’s *e4thcom*, which makes working with eForth a lot easier.

Because I often use Windows PCs for software development, I made *Escom*. Escom has the same functionality as e4thcom. The source and executable code is available on GitHub. The software is well documented in both the source code and the document in the doc directory.

To test Escom and refresh my Forth knowledge, I made some test programs, which are also available in my GitHub repository. Although I am not a real Forth programmer, with the help of Escom I was able to quickly develop software for the STM8.

I also tested with an STM32. This one has the advantage of the huge Flash and RAM memory over the STM8.

`esmallenburg@gmail.com`

Ed Smallenburg

Minimal Binary Seed

In der Ausgabe 1/2001, Seite 7, wird PlanckForth vorgestellt. Darauf bin ich gestoßen, als ich an meinem Projekt *Minimal Binary Seed* arbeitete. Dies hat sich zu einem Forth–Projekt entwickelt. Es enthält ein Kern–Forth, das für den x86–Prozessor nur 81 Bytes im 16–Bit–Modus und 101 Bytes im 32–Bit–Modus benötigt.

PlanckForth hat mich nochmal auf die „Ein Buchstaben Wörter“ gestoßen. Vor allem darauf, dass der Parser für sie ganz einfach ist: Er liest das nächste Zeichen. Erst dann hatte ich die Idee mit einer Tabelle für das Wörterbuch anstelle eines üblichen verlinkten Wörterbuchs. Das erlaubt dann ein derart kleines Kern–Forth.

Anfangs ist das Wörterbuch leer, aber auf dem Stack liegen 5 Execution–Tokens, –1 und eine Adresse (`here`). Der Forth–Quellcode definiert danach die ersten Forth–Worte. Diese sind anfangs nur ein Zeichen lang. Das Zeichen ist direkt der Index für die Wörterbuch–Tabelle. In der Tabelle steht direkt das Execution–Token (`xt`). So sind Interpreter und Parser ganz einfach:

```
Interpreter
  call key                      ; Parser liest ein Zeichen
  mov  esi, ContInterpreter     ; ForthIP := Interpreter
  jmp  [edi + ebx * CellSize ] ; ,find' and 'execute'
ContInterpreter:
  dd   Interpreter
```

Der Rest ist Initialisierung und die Implementation von 5 initialen Wörtern (s. u.). Der Forth–Quellcode definiert dann — anfangs mühsam — erste Code–Wörter:

```
2* 1+ over 0 cell next, here
```

Darauf aufbauend definiert der Quellcode ein „normales Forth“ im *Direct Threaded Code* (DTC) mit call, ein normales Wörterbuch und einen ersten Interpreter (das ist aktuell in Arbeit).

Die Ideen lassen sich leicht auf andere Architekturen bzw. CPUs übertragen, es sind:

- Wörter mit einem Zeichen Länge

- Wörterbuch als Tabelle mit 127 Einträgen (0 bis '~')
- DTC/call
- Initialer Stack mit `aHere -1 'psp@ '- '@ '! 'code-einfach`

Ich bin sehr gespannt, wie klein dies auf ARM etc. sein kann, und ob noch kleinere Implementationen möglich sind.

Das Ziel des Projekts ist nicht, ein möglichst kleines Forth–System zu haben, sondern mit möglichst wenig binären Daten ein System zu starten. Dafür ist dieses Kern–Forth (bislang) der beste Weg.

Dr. Stefan Karrmann

Ergänzung der Redaktion

Das MinimalBinaryBoot–Projekt von Dr. Stefan Karrmann steht unter der GNU–Public–License, Version 3, und ist unter `https://codeberg.org/StefanK/MinimalBinaryBoot` online zu finden. Ziel dieses Projektes ist es, ein System zu schaffen, durch welches auf einem leeren Computer durch eine minimale „binäre Saat", also ein Binär–Programm, welches direkt in den Speicher oder auf einen Datenträger eingegeben werden muss, eine Software–Entwicklungsumgebung neu aus Quellcode zu erschaffen, ohne auf bestehende Compiler/Assembler/Forth–Systeme zurückgreifen zu müssen. Der erste Schritt ist ein Minimalst–Forth, aus diesem entsteht danach ein relativ „normales" Forth, darauf aufbauend werden dann Assembler und Compiler für andere Sprachen (z. B. C) erstellt. Die Projekt–Webseite bietet eine Reihe interessanter Links zu diesem Thema, u. a. zu `https://bootstrapable.org`.

Charles Moore on Characters

Im Jahre 2020 gab es eine Videokonferenz der niederländischen Forth–Gruppe, zu der auch Charles Moore eingeladen war. Im Videoclip davon, ab der Minute 48:03, erzählt er von seinem jüngsten Projekt, dem Zeichensatz seines ColorForths.

> „One thing I have done recently is a new character set. You may be aware that I have done many many charactersets. This one is the prettiest, and the simplest, and it has a unique property. Normaly what I have been doing, I would generate a character as a raster library. I would have 48 symbols representing the characters stored in RAM. It finaly occured to me that computers are much faster than they need to be, so instead of storing the characters, I store the code that generates the characters, and when I want to display one, I compute it. Computers are pretty fast to do that, without perceptible delay, and it saves memory. It adds flexibility, you can add any characters on the fly, and you can invest a fair amount of work in calculating a character because it doesn't matter. So I got pretty characters available on the bench. I encourage people to write their own characters, its fun, its easy, and it gives you another customisable aspect to forth."

mk

Mecrisp Forth auf dem Raspberry Pico

> „Forth ist sicher nicht eine der modernsten Programmiersprachen, nimmt aber auf begrenzte Ressourcen besondere Rücksicht — genau das Richtige für den Raspi Pico."

So eröffnete Carsten Fulde seinen Beitrag „Programmierung für Mikrocontroller" am 09.03.2022 auf *Heise–Online* zum Forth *Mecrisp.* Und fuhr dann fort:

> „... Der Schöpfer von Mecrisp, Matthias Koch, wählte seinerzeit ganz bewusst eine Kombination von Forth mit Compileroptimierungen, um die für seine Doktorarbeit benötigten Algorithmen zur digitalen Signalverarbeitung interaktiv und in Echtzeit testen zu können. Vor mehr als einer Dekade, im August 2011, wurde die erste Version von Mecrisp für einen MSP430 unter der offenen GPL–Lizenz veröffentlicht. M–écriS–P, von französisch écris = schreiben, ist also ein Kunstwort für ‚Be–schreibe' auf dem MSP."

Das hat Herr Fulde sorgfältig recherchiert und prima zusammengefasst. Sowohl die inzwischen lange Geschichte des Mecrisp wird klar, als auch die Wortschöpfung von Matthias Koch für sein Forth–System wird deutlich. Matthias verwendet ja auch im Quellcode seiner Forthsysteme lustige Benennungen. So wurde das Macro für den „Header" eines Forthwortes zur „Wortbirne", angelehnt an das umgangssprachliche deutsche Wort „Birne" für „Kopf" — nicht ganz so leicht für andere Zungen, dahinterzukommen.

Und wie produktiv Matthias in diesen Jahren tatsächlich schon gewesen ist, wird ebenfalls deutlich:

> „Diesem Ur–Mecrisp folgten noch vier weitere Zweige: Mecrisp Stellaris für ARM Cortex CPUs schon im August 2013, Mecrisp Quintus für RISC–V– und MIPS–Prozessoren, Mecrisp Ice für die J1–Forth–CPU als FPGA–IP und Mecrisp Across speziell für winzig kleine Controller. Die Bandbreite der abgedeckten Systeme hier aufzuzählen, erspare ich mir an dieser Stelle; allein für die ARM–Plattform sind es über 60. Die Liste kann auf der Mecrisp–Seite auf Sourceforge eingesehen werden und beinhaltet natürlich auch die *Bluepill*, den STM32F103C8T6 ..."

Knapp formuliert, aber alles drin, so muss das. Ein erfrischender Beitrag!

Es folgt dann noch eine kleine Einführung in Forth an sich. Aber auch ein Testaufbau mit dem Raspi Pico und USB–Seriell–Adapter zur Kommunikation wird erklärt. Und schließlich würdigt er die erstaunliche Leistungsfähigkeit des resultierenden Systems, sehr schön.

Fuldes Artikel dort auf Heise–Online hat mir gut gefallen. Mein Dank an Carsten Strotmann für den Hinweis. Herr Fulde ist übrigens im Computermuseum Visselhövede tätig. Dort hat es eine Abteilung für Homecomputer. Hoffentlich darf es bald wieder öffnen … mk

Abbildung 4: Nicht schlecht für eine interaktive Sprache auf einem Mikrocontroller: Toggeln eines I/O–Pins mit über 6 MHz. (Quelle: Fulde, Heise–Online)

`https://www.heise.de/hintergrund/Programmierung-fuer-Mikrocontroller-Mecrisp-Forth-auf-dem-Raspberry-Pico-6544522.html`
`http://mecrisp.sourceforge.net/`
`https://www.computermuseum-visselhoeve.de/`

Einführung in die Programmierung mit UXN

In Heft 4d2021–4 hat Erich Wälde von der kleinen stack–basierten Sprache *TAL*[6] berichtet. Wer tiefer in die faszinierende Welt des *Permacomputings* mit UXN einsteigen möchte, für den gibt es nun das kleine Einsteiger–Handbuch „Introduction to UXN programming“ von „sejo vga“. In 7 Kapiteln wird die Programmierung in UXNTAL erklärt, von den ersten Schritten dieser stack–basierten Sprache (von Infix zu Postfix) über Ansteuerung des virtuellen Bildschirms, Tastatur, Maus und Sprachbestandteilen wie Variablen und Sprüngen. Am Ende steht die Programmierung eines kleinen *Pong–Spiels* und der Zugriff auf Dateien und die Tonausgabe.

Erhältlich ist dieses kleine eBook auf der Plattform `itch.io` für 8 US$ (oder mehr, wenn man den Autor unterstützen möchte). Das Buch ist als ePUB, im Mobi–Format und GEMPUB (ein eBook–Format auf Basis des Gemini–Protokoll–Markups) verfügbar. Für nur 2 US$ gibt es eine spanische Version des Buches.

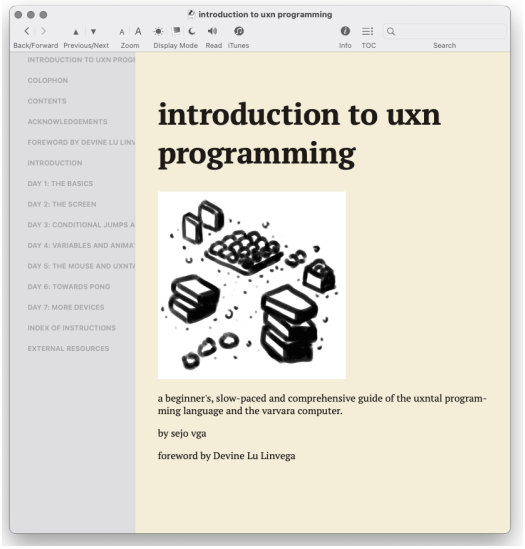


Abbildung 5: Das kleine UXN eBook.

`https://compudanzas.itch.io/introduction-to-uxn-programming`
`https://wiki.xxiivv.com/site/permacomputing.html`

UXN in Webassembly

Mittels *WEBUXN* von Bruno Garcia können UXN–Programme direkt im Webbrowser[7] ausgeführt werden. WEBUXN ist eine Portierung der UXN virtuellen Maschine in Webassembly.

Das im GitHub–Repo enthaltene Programm `rom2html` wandelt UXN–ROMs direkt in Webseiten, welche statisch von einem Web–Server ausgeliefert werden.

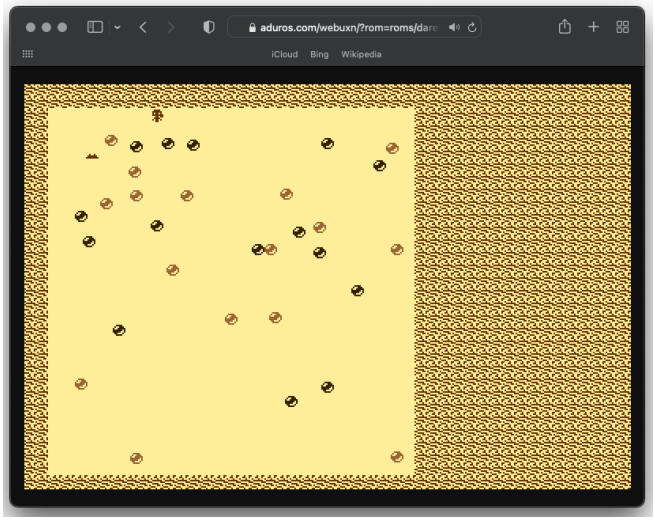


Abbildung 6: Ein in UXN geschriebenes Spiel im Web–Browser.

`https://github.com/aduros/webuxn`
`https://webassembly.org/`

[6] *UXN* ist die virtuelle CPU, *TAL/UXNTAL* die Programmiersprache und *Varvara* das virtuelle Computersystem rund um die UXN–CPU.
[7] Solange die Sicherheitseinstellungen des Browsers nicht zu streng eingestellt sind. So läuft WEBUXN z. B. nicht im TOR–Browser.

Es war einmal — eine CD–ROM

Wir schreiben das Jahr 2003. Eine kleine aufrechte Schar Forthler ließ verlauten:

> „Das CD–ROM–Projekt ist stolz darauf, die erste Version der CD–ROM der Forth–Gesellschaft e.V. anlässlich der Forth–Jahrestagung Forth'03 in Lambrecht vorstellen zu können."

Auf dieser CD–ROM fanden sich folgende Informationen: ein Abzug des Web–Servers `www.forth-ev.de` vom 8. April 2003, das komplette 4D–PDF–Archiv des Forth–Magazins „Vierte Dimension"[8], ausgewählte Forth–Systeme für einige Plattformen und sogar ein Acrobat Reader für einige Plattformen.

An Forth–Systemen war für DOS das F–PC drauf gepackt worden, sowie das volksForth.

> „*F–PC* ist das große Forth–System für DOS. Es ist ein vollständiges 16–Bit–Forth–83–Entwicklungssystem mit integriertem Hypertext–Editor für Stream–Files, Multitasker, Debugger und jeder Menge nützlichen Zusätzen. Durch seine spezielle Vokabular–Struktur kann es den gesamten 640–KB–DOS–Speicher verwenden."

> „*volksForth* ist das klassiche deutsche Forth–System. Es wurde Anfang der 1980er Jahre in Hamburg als ultraForth/volksForth entwickelt. Neben der 8086–DOS–Version existieren volksForth–Portierungen für 68000, 6502 und Z80. volksForth ist ein 16–Bit–Forth–83–System mit Block–Editor, Multitasker, Debugger und Meta–Compiler. Programme und Daten sind auf 64 KB begrenzt. volksForth ist im Vergleich zu etwa F–PC schlank und einfach zu verstehen und diente als Vorlage für eine Reihe weiterer Forth–Systeme, wie etwa *bigForth*.

Win32Forth, bigForth und auch schon Gforth gabs für Windows. Ob die `w32Forth_v6p01.exe` in Linux Wine wohl geht?

bigForth `bigforth-2.0.9.tar.bz2` von Bernd Paysan und Gforth sind besser auf Linux und waren dafür auch drauf auf der CD.

> „bigFORTH ist ein Maschinencode–Forth, das für Linux und für Windows 95/98 unter der GPL erhältlich ist. Sein herausstechendstes Merkmal ist seine graphische Benutzerschnittstelle MINOS und der Dialog–Editor Theseus."

Und dann das PFE `pfe-0.32.94.tar.gz` von Dirk–Uwe Zoller, ein in C implementiertes Forth für eingebettete Systeme. Die CD–ROM gibt es noch bei mir, falls wer interessiert sein sollte ...

Und es gab Links zu anderen Forth–Gruppen auf der HTML–Index–Seite der CD, zur Forth User Group Niederlande, zur FIG Russland und FIG of Taiwan sowie zu Forth@Google.

Die niederländische Gruppe `https://forth.hcc.nl/` ist bis heute sehr aktiv und trifft sich regelmäßig, nach wie vor — mal leibhaftig, mal im Videochat. Ihre jüngste PDF–Ausgabe der ForthWords hatte die Nummer 2022–02.

Selbst der Link zur `http://www.forth.org.ru/` funktioniert noch (13.03.2022) und man sieht dort, dass sie all die Zeit aktiv waren, zuletzt am 29.05.2021, also noch nicht so lange her. Shabronov teilte da eine neue Adresse mit: `http://90.189.213.191:4422`, und mit was man sich beschäftigt: examples of using Forth, Forth on spf4, Tachometer for a quadrocopter, Magnetometer, Scout WI–FI und „maggots".

Der Link nach Taiwan `http://www.figtaiwan.org/` hingegen war nun tot.

Tja, und dann hat das Internet hierzulande doch rasch solche Maßnahmen zur Verbreitung von Forth überflüssig gemacht. Forth findet man nun auch so. mk

volksForth — Erinnerungen

Das volksForth wird ja, wie ihr sicherlich wisst, u. a. von Philip Zembrod weiter gepflegt. Ausgelöst von der Nachricht, dass Georg Rehfeld Heilig Abend 2021 in Hamburg gestorben ist, ergab sich kürzlich ein Dialog zwischen Philip und Ulrich Hoffmann über die Geschichte des *volksForth*, weil Georg damals viel damit gemacht hatte.

PZ: Ich habe Georg nie persönlich kennengelernt, glaube ich zumindest, aber in den letzten anderthalb Wochen, während ich immer wieder weiter am volksForth gearbeitet habe, habe ich oft an ihn gedacht und mich über die Codebasis mit ihm verbunden gefühlt, speziell seit mir vor ein paar Tagen klar geworden ist, dass von ihm die C64–Portierung ist, auf der ich ja die allermeiste meiner Forth–Zeit verbracht habe.

UH: Ja, Georg hat sich um den C64 gekümmert. Inwieweit er auch die C16– und Plus4–Portierungen gemacht hat, weiß Claus Vogt sicher besser. Aber insbesondere das schräge Screen–Format von 24 x 41 und 1 x 40 des ultraForths[9] geht auf ihn zurück. Bernd Pennemann

[8] Natürlich auf dem damaligen Stand, obschon da einige Hefte noch fehlten, das wurde durch den Anlass dann aber rasch behoben. Heute ist das Archiv immer up to date.

[9] So hieß das volksForth auf dem C64.

hatte die 6502–Portierung von volksForth gemacht (auf eigener Hardware), dann Atari–ST–68000 zusammen mit Dietrich Weineck in Bremen. Ich habe mich um 8080 und Z80 gekümmert, CP/M 2.2, CPM 3, Schneider CPC und anderes. Ich erinnere mich auch gerne an die samstäglichen Forth–Treffen in der Berufsschule in Eimsbüttel und die sich daran anschließenden Abende beim Griechen. Calamari war Georgs Gericht und Retsina.

PZ: For the records ein Link auf die Kernel–Quelle des MSDOS–VF, das ein bisschen Geschichte enthält [1]. Dabei frage ich mich noch eins: Bernd Pennemanns erste 6502–Version — das war im Wesentlichen eine Neuentwicklung? Oder basierte das auch auf etwas anderem? F83? Ich vermute auch mal, daß Klaus Schleisiek parallel dazu den Metacompiler entwickelt hat?

UH: Klaus Schleisiek hatte sein eigenes Forth auf einem Compaq–PC — seinerzeit sensationell: mit Festplatte! Er war zuvor in Kalifornien gewesen und hatte den Forth–Geist von dort mitgebracht. Aus Klaus' Forth hat sich dann volksForth entwickelt, das für viele Rechner gleich verfügbar sein sollte. Bernd Pennemanns 6502–Version ist soweit ich weiß ein Ableger von Klaus' Forth. Der Metacompiler stammt von Klaus. Ich selbst hatte den Quellcode der 68000–volksForth–Version und habe den via F83 gebootstrappt.

PZ: Dann hast also du den volksForth–Meta–Compiler auf F83 portiert und damit mit den modifizierten 68000–VolksForth–Quellen das volksForth nach CP/M–80 portiert?

UH: Ja — es hat Zwischenversionen gegeben, da habe ich mit F83 (CP/M) den volksForth–Kern targetkompiliert. Leider fliegt man dabei lange im Dunkeln und es war ein Moment großer Freude, als der Kernel erfolgreich bootete und der Outer–Interpreter funktionierte. Die volksForth–Quellen haben für viele Worte auch High–Level–Implementierungen (etwa `(WORD` und das Buffer–Management). Die habe ich zunächst verwendet, damit das System auf die Beine kommt und sie dann später im laufenden CP/M–volksForth wieder in Code (8080/Z80–Assembler) re–implementiert, um die nötige Geschwindigkeit zu bekommen. Ich vermute mal, dass die Zwischenstände alle auf alten, heute unlesbaren Disketten sind und ich sie nicht rübergerettet habe.

PZ: Ach ja, richtig, es sind tatsächlich noch eine Reihe von alternativen Colon–Definitionen für Code–Worte selbst in den MSDOS–VF–Quellen vorhanden, allesamt mittels \\ auskommentiert, versteht sich, und ebenso in den CPM–VF–Quellen [2]. Scheint, dass du doch einiges an High–Level–Code für die Nachwelt erhalten hast.

UH: Der High–Level–Code stammt tatsächlich in den seltensten Fällen von mir und geht vermutlich auf Klaus, der zeitgleich die MSDOS–Portierung gemacht hat (ich hatte zu der Zeit keinen PC), zurück. mk

(Quelle: E–Mail; im März 2022)
[1] `https://github.com/pzembrod/VolksForth/blob/msdos-tests/8086/msdos/src/kernel.fth`
[2] `https://github.com/forth-ev/VolksForth/blob/master/sources/cpm/source.fth`

Excurs

Ein bestehendes Forth portieren, das kann man so machen:

1. Reduzieren
   Die Code–Definitionen[10] im bisherigen System weitgehend reduzieren, d. h. in Forth[11] umschreiben und das System auf der bisherigen Plattform immer wieder *meta–compilieren*, bis das alles fehlerfrei läuft.

2. Portieren
   Die verbleibenden unverzichtbaren Code–Definitionen[12] auf die neue Plattform anpassen, somit *cross–compilieren*.

3. Expandieren und „Tunen“
   Hat man ein laufendes Zielsystem auf der neuen Plattform, wird nur noch *meta–compiliert* und für eine bessere Performance zeitkritische Forth–Definitionen in den Code der neuen Plattform umgeschrieben.

uh

Die schlimmste Programmiersprache aller Zeiten?

Auf der NDC–Konferenz in Oslo Ende 2021 hielt Mark Rendle einen launigen Vortrag mit dem Titel „*The Worst Programming Language Ever*“. Ist hier von Forth die Rede? Schaut euch das Video an, vielleicht kommt ihr ja auch ins Schmunzeln … cs
`https://www.youtube.com/watch?v=vcFBwt1nu2U`

[10] „Low–Level–Forth“
[11] „High–Level–Forth“
[12] „Primitives“

Wireless Mesh Network

Willem Ouwerkerk, Henny Luijkx

The target of the project is a self constructing safe mesh network of small microcontroller nodes. For the transceiver we did choose the small and powerful nRF24L01+. *There is a basic driver file with a defined payload structure. On top of that is a uniform node controller that is identical on each node. The node handler is integrated in the noForth interpreter for easy handling and debugging.*

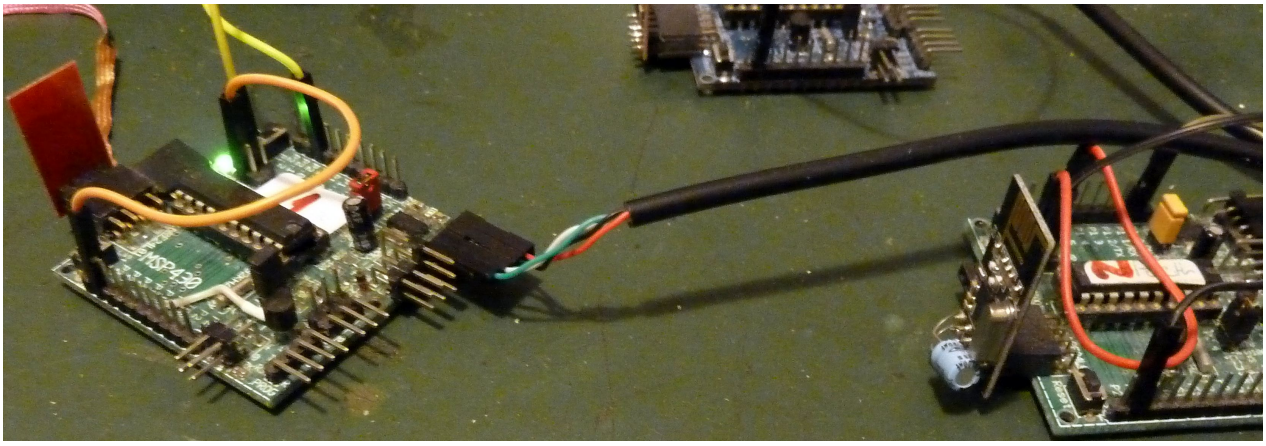


Figure 1: Node 1 and 2.

Options

- Network size adjustable with one constant
- Most node data is stored in a single bit
- Send & receive uses dynamic payloads
- Small usable command set
- Failing nodes are noted and suppressed
- Restoration of failed nodes
- Every node is identical and has all network information
- Node handler build in Forth's outer interpreter
- Each node can scan for (direct) nodes within reach
- Each node can register itself to an existing network
- Adjustable node settings after the source is compiled

The payload structure

Dynamic payload format from 2 to 32 bytes:

```
| 0      | 1    | 2    | 3        | 4    | 5...31   |
|--------|------|------|----------|------|----------|
|Command|Dest.|Orig.|Sub node|Admin|d00...d1A|
```

0 pay> = Command for dest.

1 pay> = Destination node

2 pay> = Origin node

3 pay> = Address of sub node

4 pay> = Administration byte

5 to 31 pay> = Data max. 26 bytes

Basic node functionality

There are several maps with data about the constructed network. There are maps for all nodes in the network, direct accessable nodes, indirect accessable nodes, failed nodes. Also there is data on not direct accessable nodes, hop data, and about the node types present in the network.

Some of the node commands:

.STATUS	Show current node settings & RF status
.ALL	Display the complete network map
SCANX	Start a scan for direct accessable nodes
REGISTER	Register myself to an existing network
SETRF	Replace & save all important node setup data
>NODE	Send a command to a given node
HOP	Gather node data from all direct nodes
INFO	Gather node type info from all direct nodes
OFF ON	(De)activate an output on given node
SWITCH?	Simple event handler that turns each node to a two–way switch that toggles all node outputs

Create an alternative path

Currently we are working on a way to create an alternative path to each node. This path will be used when the basic path does not function anymore because of the failure of a node. This path may not contain the same (hopping) node!

Fig. 2 gives an example of a network. The arrows show with which other nodes a node can communicate.

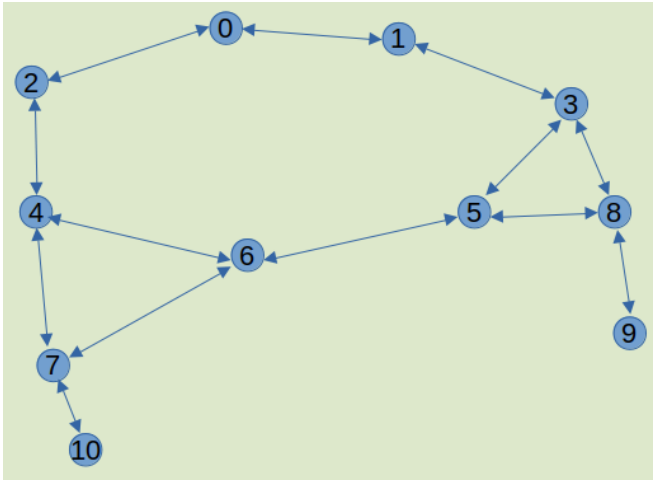


Figure 2: Network with 11 nodes.

Node 0 can communicate directly with node 1 and 2. If for example node 0 cannot communicate with node 1 this can be caused by:

- A broken node.
- A shield appeared inbetween the nodes.
- A node moved out of reach of the other node.

Up until now we only used fixed networks, so the last option is not possible (yet). Because it is not sure what causes the communication interrupt, an alternative route can be used to reconnect the nodes.

A procedure is developed whereby one node takes the lead. This is not democratic, but is the best to avoid a lot of collisions. The procedure is as follows:

1. The leading node sends a message to the first direct node.
2. This node increases the hub counter in the message and sends it further to his first direct node, etc.
3. The message is not send to the node where it came from.
4. When a node cannot send the message to another node (e.g. node 9) or the message is received by a node that already treated the message (e.g. the leading node) a message is send back to inform the previous node that the end of this route is found.
5. A node with the possibility to use another route will send the message to that route until the message has followed all possible routes.
6. The leading node will assign the next node in the network to send a message to all his direct nodes.

Because the hub counter in the message is increased by each node that forwards the message the shortest and second shortest route can be calculated by each node. In the end each node knows via which direct node another node in the network can be reached with as little communication as possible. With the same procedure a possible alternative route is found in case the first route is not functioning any more. In the example network, node 9 and 10 have no alternative route, but all the other routes have an alternative.

More info on:

`https://project-forth-works.github.io/`

Or specific:

`https://github.com/project-forth-works/project-forth-works/tree/main/Communication-Protocols/Wireless-Communication`

Tools at present

- A network build routine using the SCANX function.
- A ping routine to check (the efficiency) of connections.
- Routines to check disturbance on all or one given channel.
- Routines to check the range of a node (for easy placement).
- Some small demonstration routines that control all the outputs on the network.

Examples

Listing 1 is an example of the network build routine.

Listing 2 is a “running light” demo. This example controls all nodes outputs after a network is constructed. It can run on every node, even on multiple nodes! It can be stopped by entering a key or a stop command from any other node.

Listings

```
1   \ Listing 1
2   create NEW      #map allot      \ Scratch bit accu to find new hopping nodes
3   : BUILD     ( -- )
4       scanx  cr                   \ Search & note direct accessible nodes
5       direct >work  ch s >others  \ Perform SCANX on all new found nodes
6       hop  cr                     \ Ask node data at all new found nodes
7       direct >work  ch h >others  \ Perform HOP on all new found nodes
8       1  begin                    \ Start network building on indirect nodes
9           indirect count*         \ Count indirect nodes
10      tuck <> while               \ More then number on stack?
11          indirect new #map move  \ Yes, copy indirect node map
12          new >work  ch s >others \ Perform SCANX on all indirect nodes
13          hop  cr                 \ Gather node data at all nodes again
14          all >work  ch h >others \ Perform HOP on all nodes again
15      repeat  drop                \ Until no new nodes are found
16      finish ;                    \ Gather all node type info on each node
17
18  \ Listing 2
19  : RUNNER    ( -- )
20      run  begin                          \ Enable a free running program
21          all >user                       \ Copy all nodes to user accu
22          begin  user >next? while        \ Fetch & clear next node number
23            dup on  100 <wait>    \ Activate node output, wait 100 ms
24            off  30 <wait>              \ Deactivate node output, wait 30 ms
25          repeat
26      halt? until ;                \ Until a key or STOP command
```

The *nRF24L01+ 2.4GHz transceiver* is a cheap module with a low level part of the communication layer already in hardware available. Features of the nRF24L01+ are: adjustable auto retransmit, RF ACK handshake, a 1 to 32 byte payload with variable length (Dynamic Payload), Fifo of 3 deep, 120 selectable frequencies, adjustable output power, CRC, etc.

This transceiver software is already available in the *project forth works*:

- `USCIB SPI MSP v100.f`, SPI driver for MSP430G2553.
- `Basic 24L01dn G2553-01.f`, Basic transceiver routines using 'Dynamic payloads'.
- `Transmit test.f`, Transmit command, receive & display data.
- `Receive test.f`, On command increase a counter and sent data back.
- `Range checker G2553 usci.f`, Tools to help testing the range & placement of the transceivers.
- `MSP430`, Forth implementation(s) of the example code.
- `GD32VF`, Forth implementation(s) of the example code.

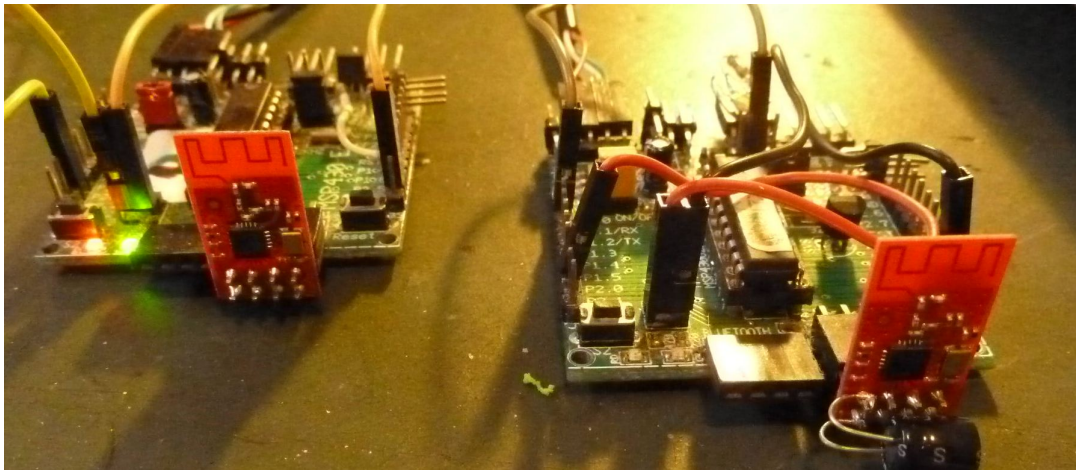


Figure 3: Bidirectional transmit & receive in action on the Egel kit.

FlexiFloat — Das Kochbuch

Jörg Völker

Der Artikel zum FlexiFloat–Format aus dem letzten Heft[1] *war noch nicht gedruckt, da kamen schon die ersten Anfragen, ob man das nicht auch für einen MSP430 implementieren könne. Natürlich kann man das, es ist nur eben eine gute Portion Arbeit. Um die zu erleichtern und auch eine Vorlage für andere Prozessoren zu geben, versuche ich mal die Algorithmen und Fallstricke allgemeingültig zu beschreiben. Ein STM8 Assembler–Listing wäre da, glaube ich, wenig hilfreich ...*

Man kann es nicht leugnen — ein Floating–Point–Paket ist erheblich aufwendiger als eine Ganzzahl– oder Fixpoint–Arithmetik. Das liegt gar nicht so sehr an den grundlegenden Algorithmen für Multiplikation oder Division, die sind eigentlich identisch, sondern an der beachtlichen Anzahl der Sonderfälle, die man abdecken muss. Und das gilt selbst dann, wenn man auf +–Infinity, NaNs[2] und die Unterscheidung zwischen +0 und –0 verzichtet.

Beim 32–Bit–FlexiFloat wird *nicht* nach jeder Operation normiert und die Mantisse ist *rechtsbündig* und im *Zweierkomplement* — das sind die wesentlichen Unterschiede zum IEEE754–Format. Daraus ergeben sich auch mehrere Unterschiede in der Implementierung.

Die Addition

Fangen wir mit Addition und Subtraktion an. Floating–Point–Numbers oder kurz FPNs kann man nur addieren oder subtrahieren, wenn die Exponenten gleich sind. Also müssen wir die FPNs erst einmal angleichen:

1. Prüfen, ob die Exponenten gleich sind. Sind sie gleich? Glück gehabt.
2. Solange möglich (d. h. kein Überlauf und Vorzeichenwechsel droht) bei der FPN mit dem größeren Exponenten die Mantisse links schieben, den Exponenten dekrementieren. Exponenten gleich? Dann gehts weiter.
3. Andernfalls bei der FPN mit dem kleineren Exponenten die Mantisse arithmetisch rechts schieben und den Exponenten inkrementieren, bis die Exponenten gleich sind.

Hier lauern gleich zwei Probleme: Punkt 3 macht nur solange Sinn, wie sich die Mantisse noch ändert. Ist sie vollständig mit Nullen oder Einsen gefüllt, kann man abbrechen; der Wert ist zu klein geworden, um am Ergebnis noch etwas zu ändern. Und bei IEEE754 sind die Mantissen positiv und ich glaube, jetzt weiß ich auch, warum: Schiebt man einen negativen Wert im Zweierkomplement nach rechts, wird er nie zu Null, sondern zum kleinsten darstellbaren negativen Wert in der jeweiligen Wortbreite. Ein weiterer Sonderfall, den man hier abfangen muss.

Derart angeglichen können die beiden Mantissen jetzt addiert oder subtrahiert werden. Dabei kann es zu einem Überlauf der Mantisse kommen, der durch Rechtsschieben der Mantisse und Inkrementieren des Exponenten repariert werden kann. Läuft jetzt der Exponent über, ist allerdings nichts mehr zu retten und es bleibt nur noch eine Fehlermeldung. Für die Subtraktion addiert man halt einen negierten Wert. Diese u. U. zeitraubende Angleichung der Exponenten entfällt aber komplett, wenn ganze Zahlen addiert oder subtrahiert werden, denn dann sind beide Exponenten im FlexiFloat–Format Null.

Vergleiche sind bei IEEE754 schneller. Da alles schon normiert ist, reicht in erster Näherung für größer/kleiner ein Vergleich der Exponenten und erst wenn die gleich sind, müssen die Mantissen verglichen werden. Das geht so beim FlexiFloat nicht, hier brauchen wir erst die Angleichung und dann eine Subtraktion, sind also in diesem Punkt u. U. erheblich langsamer. Man kann eben nicht alles haben. Auch hier ist das aber erst dann ein wirklicher Nachteil, wenn man den Ganzzahlbereich verlässt.

Die Exponenten verarbeiten

Jetzt fehlt uns noch zumindest die Division und Multiplikation. Dazu gleich ein Tipp: Da es hier ebenfalls zu Überläufen bei den 8–Bit–Exponenten kommen kann, die aber nicht gleich in jedem Fall fatal sind, ist es ratsam, die 8–Bit–Exponenten gleich in 16 Bit umzuwandeln, oder in den Schleifen mit einem temporären 8–Bit–Hilfsexponenten zu arbeiten. So können Überläufe in einem späteren Extraschritt erkannt und möglicherweise noch repariert werden. Noch einen Punkt muss man sich klarmachen: Da negative Werte üblicherweise schon in Forth in positive Werte umgewandelt werden (und ich würde erst einmal dabei bleiben), arbeiten die Multiplikations– und Divisionsalgorithmen hier tatsächlich nur auf positive 23–Bit–Mantissen. Ein „Überlauf“ liegt also u. U. bereits dann vor, wenn das Vorzeichenbit gesetzt ist, nicht erst beim Carry–Übertrag. Das so noch freie 24. Bit ist aber manchmal sehr nützlich, da es temporäre Überläufe aufnehmen kann. Um nicht mit zu vielen Details die grundlegenden Algorithmen zu verschleiern, sind die folgenden Grafiken etwas vereinfacht.

[1] *FlexiFloat — ein Floating–Point für Forth; 4d2021–04, S. 19 ff*

[2] Not a Number.

Die Multiplikation

FPNs multiplizieren heißt Exponenten addieren und Mantissen multiplizieren. Die Multiplikation zweier 23–Bit–Mantissen ergibt ein 46–Bit–Ergebnis, das schlussendlich wieder in eine 23–Bit–Mantisse eingepasst werden muss. Ich muss leider zugeben, dass auch hier IEEE754 einen Vorteil hat. Die linksbündigen Mantissen ergeben multipliziert bereits ein fertiges, ebenfalls linksbündiges Ergebnis. Bei FlexiFloat dagegen kann das Ergebnis gleich in die Mantisse passen oder eben nicht.

Der übliche Algorithmus aus Schieben und Addieren (`UM*`) arbeitet wie folgt (Abb. 1):

1. Ein Akkumulator bildet mit Faktor A ein doppeltgenaues Schieberegister für das Ergebnis
2. Ist das niederwertigste Bit von Faktor A gesetzt, wird Faktor B zum Akku addiert
3. Akku und Faktor A werden zusammen nach rechts geschoben
4. Das Ganze wird ab 2. n–mal wiederholt

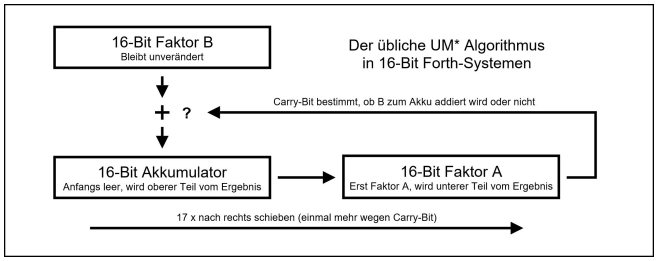


Abbildung 1: Der übliche `UM*`

Nach einigem Überlegen habe ich einen Weg gefunden, wie man die Multiplikation einerseits und das Einpassen in die Mantisse andererseits unter einen Hut bringen kann. Mich störte außerdem, dass der Standard–Algorithmus stur n–mal wiederholt wird, bei einer 24–Bit–Mantisse also selbst für `2*3` immer alle 24 Schleifendurchläufe stattfinden würden, denn das kostet viel Zeit.

Bei FlexiFloat wird die Multiplikation in zwei Schritte zerlegt. Da wir sowieso kein doppeltgenaues Ergebnis unterbringen können — oder wollen — sind Akkumulator und Faktor A getrennt. Der Algorithmus startet mittendrin mit dem 4. Schritt (!!!), das erspart im Schnitt 0,5 Programmsprünge je Durchlauf (Abb. 2):

1) Faktor B wird zum Akku addiert
2) Faktor B wird nach links geschoben
3) Ist bei Faktor B jetzt das Bit 23 (=Vorzeichen) gesetzt, weiter im zweiten Teil mit Schritt 12)
4) Faktor A wird nach rechts geschoben.
5) Ist das Carry gesetzt, weiter mit 1)
6) Ist Faktor A noch ungleich Null, weiter bei 2)
7) Ist im Akku Bit 23 (=Vorzeichen) gesetzt, Akku rechtsschieben und Exponent erhöhen
8) Fertig! Ergebnis steht im Akku

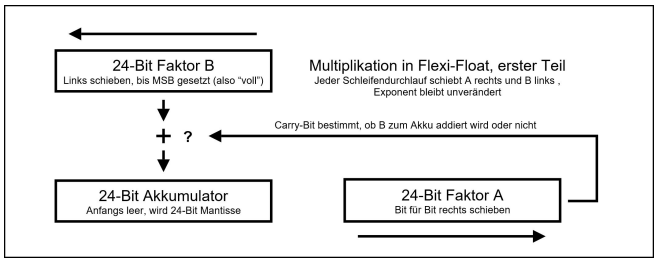


Abbildung 2: FlexiFloat–Multiplikation, Teil 1.

Anstatt also nach der Addition den Akkumulator nach rechts zu schieben, wird hier solange wie möglich Faktor B nach links geschoben. Vorteil: Solange das Ergebnis in 24 Bit passt, läuft nur dieser erste Teil, und das nur so lange, bis Faktor A Null ist. Bei `2*3` gibt es also nur zwei Schleifendurchläufe, und nicht 24! Der Akku kann nicht überlaufen, weil wir das 24. Bit in Reserve haben, das ist Teil des Tricks. Erst wenn Faktor B an seine Grenzen kommt, brauchen wir den zweiten Teil des Algorithmus. Los geht es hier mit Schritt 12) (Abb. 3):

9) Faktor B wird zum Akku addiert, ggf. wird dabei jetzt auch das Carry gesetzt
10) Der Akku wird nach rechts geschoben, mit Carry
11) Der Exponent vom Ergebnis bzw. Akku wird inkrementiert
12) Faktor A wird nach rechts geschoben.
13) Ist das Carry gesetzt, weiter mit 9)
14) Ist Faktor A ungleich Null, weiter bei 10)
15) Ist im Akku Bit 23 (=Vorzeichen) gesetzt, noch einmal weiter mit 10)
16) Fertig! Ergebnis steht im Akku.

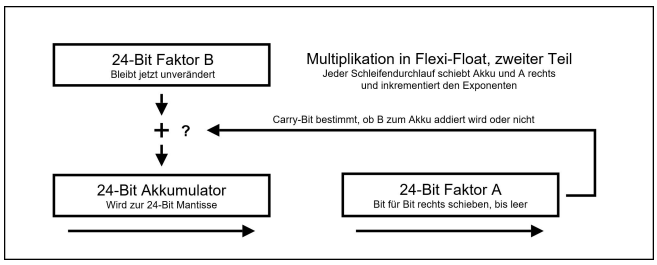


Abbildung 3: FlexiFloat–Multiplikation, Teil 2.

Dieser Teil kommt also zum Tragen, wenn der Akku = Matisse „voll“ ist. Dann müssen wir rechtsschieben und den (Hilfs–)Exponenten erhöhen. Ohne zusätzliches Schieben bzw. Normieren sieht die Mantisse jetzt genauso aus, wie sie sein soll. Hat ein Prozessor Multiplikations–Befehle, kann man die natürlich einbinden. Muss man dann allerdings das Ergebnis doch wieder hin– und herschieben, um es in die Mantisse einzupassen, ist der Vorteil zumindest zum Teil wieder dahin. Jetzt fehlt noch eine

Fehlermeldung beim Exponenten–Überlauf und Rechtsschieben der Mantisse, wenn der Exponent zu negativ geworden ist.

Die Division

FPNs dividieren heißt Exponenten subtrahieren und Mantissen dividieren. Für die Beschreibung der Division können wir die Exponenten am besten erst einmal völlig beiseitelassen. Ich tue einmal so, als hätten wir es mit zwei positiven und ganzzahligen 23–Bit–Mantissen zu tun, was natürlich so nicht immer stimmt, aber das Verständnis wegen der Ähnlichkeit zum Forth–Ganzzahl–Standard möglicherweise erleichtert. Im Folgenden ist deshalb nicht ganz korrekt von „Vorkomma“ und „Nachkomma“ die Rede.

Schon wieder benötigen wir zwei Verarbeitungsschritte für die Mantissen, eine „Vorkomma“–Division, die ggf. einen Rest zurücklässt und dem Forth `UM/MOD` weitgehend entspricht, und dann den „Nachkomma“–Teil, der den ggf. vorhandenen Rest weiterverarbeitet, solange noch Platz im Quotienten für weitere Ergebnisbits ist.

Der übliche Algorithmus aus Schieben und Subtrahieren — `UM/MOD` — arbeitet wie folgt (Abb. 4):

1) In einem Akkumulator werden die Subtraktionen durchgeführt — am Anfang steht hier die obere Hälfte des Dividenden, am Ende steht hier der Rest

2) Ein zweites Register (nennen wir es Register A) beinhaltet zunächst den unteren Teil des Dividenden — am Ende steht hier der Quotient

3) Register A wird nach links geschoben, das Carry–Bit wandert in den Akkumulator, der ebenfalls nach links geschoben wird

4) Vom Akkumulator wird der Divisor subtrahiert

5) ist das Ergebnis positiv, wird das niederwertigeste Bit in Register A gesetzt

6) Ist das Ergebnis negativ, wird der Divisor wieder addiert, und so die Subtraktion rückgängig gemacht

7) n–mal weiter mit 3)

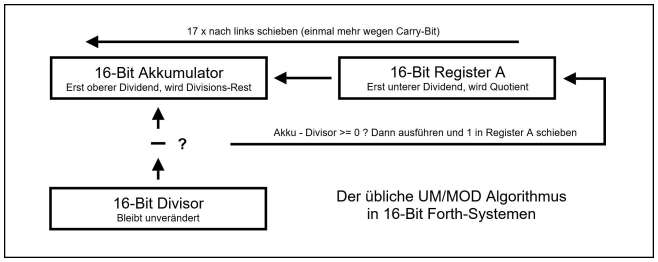


Abbildung 4: Der übliche `UM/MOD`

Bei 8–Bit–Prozessoren lassen sich Subtraktionen über Carry besser verketten als Vergleichsbefehle, ansonsten könnte man statt der Subtraktion auch vergleichen. Dieser Grundalgorithmus für den „Vorkomma“–Teil kann so übernommen werden, allerdings ist der Dividend bei FlexiFloat ja nur 24 Bit breit (Abb. 5).

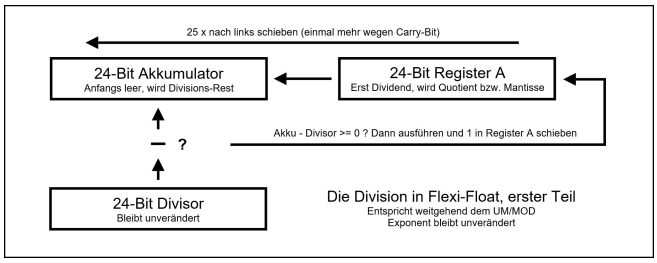


Abbildung 5: FlexiFloat–Division, Teil 1.

Es gibt schon hier eine Menge Tricks, wie man die Verarbeitung beschleunigen kann. So muss das Register A nicht 24 Bit breit sein, man kann es auch in drei Bytes zerlegen und diese Bytes nacheinander bearbeiten. Das spart auf 8–Bit–Prozessoren einiges an Schieberei.

Sind die Mantissen von Dividend und Divisor gut gefüllt, was bei FPNs ja der Regelfall ist, dann wird dieser erste Algorithmus viel herumschieben, bevor es zur ersten erfolgreichen Subtraktion kommt, wenn überhaupt. Das kann man mit ein paar Tests vorab erkennen und durch Kopieren ganzer Bytes vom Dividenden in den Akku die langsame Bit–für–Bit–Schieberei abkürzen.

Das Register A ist jetzt vollständig zum Quotienten geworden. Der zweite Teil der Division unterscheidet sich vom ersten Teil wie folgt (Abb. 6):

- Es gibt keine feste Anzahl von Schleifendurchläufen. Dieser Algorithmus läuft, bis der Quotient gefüllt ist oder der Akku bzw. Rest Null ist
- Der Akku bzw. Rest wird logisch links geschoben, unabhängig vom Quotienten
- Jedes weitere Linksschieben des Quotienten dekrementiert jetzt den Exponenten

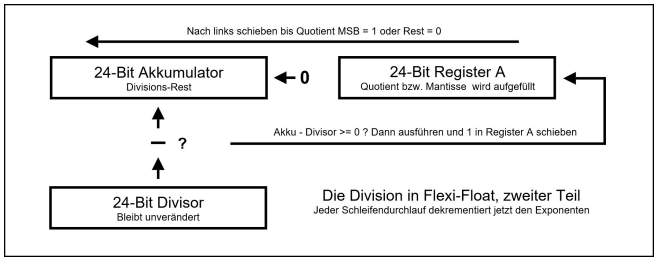


Abbildung 6: FlexiFloat–Division, Teil 2.

Dieser zweite Teil der Division ist die aufwendigste Schleife des FlexiFloat–Paketes, da es zwei bzw. sogar drei Abbruchbedingungen gibt.

Jetzt wird es endgültig Flexi

Beim Codieren der Division ist mir aufgefallen, dass man hier sehr einfach eine weitere nützliche Abbruchbedingung für die Nachkomma–Bits ergänzen kann. Wenn nicht gerade der Rest irgendwann zu Null wird, läuft die Division ja immer weiter bis zum bitteren Ende, bis in unserem Fall die Mantisse mit 23 Bit gefüllt ist. Das ist nicht immer sinnvoll. Wenn ich Sensoren mit Genauigkeiten im Prozent–Bereich habe, brauche ich nicht so viele Nachkommastellen. Außerdem kostet es unnötig Zeit.

Beim FlexiFloat gibt es deshalb einen Parameter, mit dem die Anzahl der Schleifendurchläufe bei der Nachkommadivision begrenzt werden kann. Das ist auch sehr „forthig“ und verbessert die Kompatibilität. Bei ganzen Zahlen und diesem Parameter auf Null werden z. B. überhaupt keine Nachkommastellen berechnet; Quotient und Rest werden zurückgegeben, ganz so wie bei `UM/MOD`, nur ohne doppeltgenauen Ausgangswert. Bei der Division spart die Beschränkung der Nachkommastellen bzw. des Exponenten Schleifendurchläufe und damit Zeit. Bei der Multiplikation klappt das so nicht, aber beim Addieren und Subtrahieren ergeben sich wieder Vorteile, wenn die Exponenten auf einen negativen Maximalwert geklemmt sind. Der Effekt ist eine Artihmetik mit den Eigenschaften einer Festpunkt–Arithmetik, die aber bei einem Überlauf nicht gleich Fehler produziert, sondern den Exponenten inkrementiert und damit quasi automatisch wieder zur Fließkommaarithmetik wird. FlexiFloat ist damit wirklich flexibel auf die jeweilige Applikation anpassbar und deckt mit seinen Algorithmen die ganze Palette von Ganzzahl– über Festpunkt– bis Fließkomma–Arithmetik ab — spannend!

FlexiFloat zusammengefasst

- Der Verzicht auf das kategorische Normieren nach jeder Operation erspart bei Addition und Subtraktion in der Praxis viele Schiebe–Operationen. Bleibt man z. B. bei ganzen Zahlen im Bereich +–2^23, wird überhaupt nie geschoben
- Ganzzahlige Werte können ohne Umwandlung zur Adressierung verwendet werden
- Logische Verknüpfungen der Mantisse ergeben Sinn, wenn der Exponent Null ist
- `True` und `False` sind darstellbar und funktionieren auch bei logischen Verknüpfungen wie gewohnt
- Die Floating–Point–Geschwindigkeitsnachteile kommen erst zum Tragen, wenn man den Ganzzahlbereich verlässt, also Floating–Point auch wirklich nutzt
- Der Faktor A (s. o.) bei der Multiplikation bestimmt die Anzahl der Schleifendurchläufe. Nimmt man den `TOS` als Faktor A, werden Adressberechnungen z. B. für Tabellen oder Strukturen, die ja mit kleinen ganzzahligen Werten multiplizieren, sehr effizient. Noch universeller: die Faktoren prüfen und dann ggf. tauschen
- Und der Forth–Vorteil: Die Zwitter–Eigenschaften dieses Formats erlauben den Verzicht auf etliche Forth–Arithmetik–Hilfsworte (so an die 50), wie im ersten Teil zu FlexiFloat im vorigen Heft gezeigt. Ein zusätzlicher Floating–Point–Stack ist nicht erforderlich

Nach zahllosen Varianten der Multiplikations– und Divisionsalgorithmen umfasst meine aktuelle Version für den STM8 derzeit die folgenden Worte:

```
+ - * /
OR XOR AND
FLIP SEX SHIFTR SHIFTL FLOAT?
< = > 0< 0= 0> 1+ 1- 2*
NEGATE ABS FLOOR FTRUNC
```

Der Assemblercode ist etwa 500 Zeilen lang und belegt 780 Bytes. Entgegen meiner eigenen Empfehlung werden dabei die Vorzeichen schon auf der Code–Ebene verwaltet. Könnte schlimmer sein!

Ausblick

An vielen Stellen kann jetzt noch optimiert werden. Bei der Addition und den Vergleichen kann man z. B. die Differenz der Exponenten betrachten und so vorab feststellen, ob das ganze Schieben überhaupt Sinn macht. Ähnliche Optimierungsmöglichkeiten gibt es zuhauf. Ob das immer Sinn macht, muss jeder selbst entscheiden. Letztlich wird das Programm ja auch immer länger, und die entscheidende Frage ist doch immer: Wie häufig greift die Optimierung in der Praxis? Ich habe schon viele „Optimierungen“ weggeworfen, weil der Effekt dann doch zu gering war und in einem schlechten Verhältnis zum Aufwand stand. Auch habe ich erst einmal konsequent auf das Runden verzichtet, denn auch das ist mit nennenswertem Aufwand und auch Rechenzeit verbunden.

Fazit

Also ein Wochenendprojekt ist das alles hier eher nicht. Aber das Resultat ist verlockend: Die Arithmetik im Forth wird gründlich vereinfacht, das Programmieren ist weniger fehlerträchtig und macht damit einfach mehr Spaß! In meinen Augen macht es einfach mehr Sinn, sich einmal mit einer etwas komplexeren Implementierung zu beschäftigen und dann in vielen Projekten davon zu profitieren, als umgekehrt eine einfache Implementierung zu feiern und sich dann in vielen Projekten mit einer in der Anwendung umständlichen Arithmetik herumzuschlagen.

Ich bin auf das Feedback gespannt.

eForth für den MSP340FR5739

Michael Kalus

Das neue Template des eForth — `430eforth-x-43n7vis.asm` — wurde benutzt, um es für den „kleinen Bruder“ der „eForth Dream Machine“ anzupassen. Anders als von Manfred Mahlow vorgesehen, hab ich die Umstellung mit den Mitteln gemacht, die mir in Windows 10 zur Verfügung standen. Das ging ohne Probleme gut: Notepad++, `naken-asm.exe` ausgeführt in der `CMD.exe`, und als Programmer den Lite–FET–PRO430 von Elprotronic. Den hatte ich mal gekauft, und der ist mir einfach vertrauter und irgendwie auch komfortabler als das Uniflash von TI, welches auch anstandslos das Target geladen hat und obendrein kostenlos bei TI zum Download steht. Als serielle Verbindung zum Target habe ich nach wie vor Teraterm in Benutzung. Also, ihr könnt auch in einer W10–Umgebung ganz zwanglos mit eForth starten.

IDE oder was?

Die Vorlage existiert ja nun, dank Manfred Mahlows akribischer Vorarbeit, bereits für zwei Typen von MSP430–MCUs: für die mit 16K–Flash– und die mit 64K–FRAM–Memory. Ich hätte das eForth gerne auch für mein Launchpad mit dem kleineren FR5739 mit seinen 16K–FRAM gehabt.

Manfred gab mir zu verstehen: „Na, dann mach dir das doch!“ So ist das wohl noch immer in der Forth–Welt: Brauchst du ein Forthsystem, mach dir halt eins.

Zunächst galt es zu verstehen, was er da gemacht hat. Von einem anderen kleinen Forthsystem, dem Camelforth, kannte ich es noch so, dass der MCU–spezifische Code in eine eigene Datei ausgegliedert ist und auch der Forthkern selbst dann noch über mehrere Dateien verstreut liegt. Da gibt es den High–Level–Teil, die Primitives[1], die Sprungtabelle für Interrupts (Vector Area) und noch spezielle Speicherbereiche (InfoB Area z. B. im MSP430). Und man braucht dann den Linker einer IDE, um all die Stücke korrekt zusammenzusetzen. Diese IDEs richtig einzustellen, war für mich immer eine hohe Hürde.

Hier ist nun alles aus einem Guss im Assembler–Listing untergebracht. So kommt man mit seinem Editor aus. Manfred bewegt sich in seiner Linux–Umgebung traumwandlerisch sicher. Ich nicht einmal auf Windows 10. Dennoch ist es mir gelungen!

Notepad++ erwies sich dabei wieder als sehr vielseitig. Die Ansichten der Dateien in zwei Fenstern, ein komfortables „Find“, welches man sich für seinen Zweck anpassen kann, auch die Betrachtung ganzer Folder als „Workspace“ sowie eine „Map“ der aktuellen Datei am rechten Rand, all das entspricht eigentlich bereits einer ausgewachsenen IDE. Was dabei natürlich fehlt, ist ein „Build“ wie in so einer richtigen IDE. Notepad++ weiß ja nichts davon, welcher Assembler für welches Target verwendet werden wird. Da ist man also frei in seiner Wahl. Ich habe eForth nun mit dem Naken–Assembler generiert, der als `naken_asm.exe` auch für WIN10 da ist und prima funktioniert. Dabei stellte sich heraus, dass das vom Notepad++ ganz praktisch unterstützt wurde. Denn klickt man in die Ansicht seines Workspace, bringt einen ein Rechtsklick mit der Maus sogleich in die `CMD.exe` mit dem richtigen Pfad zum Workspace. Und legt man sich dort sein Batchfile hin, sind es nur zwei Klicks zum „Build“.

Mein `make.bat` lautete:

```
naken_asm.exe
-l 430eforth-fr5739-43n7vis.asm
-o 430eforth-fr5739-43n7vis.hex
> error.log
```

Natürlich alles in einer Zeile, das ist hier nur wegen des Spaltenformates umgebrochen.

Eigentlich ist `out.hex` die Default–Ausgabe des `naken_asm` für das Image. Aber durch die Option `-o` kann man das nennen und ablegen, wie und wo man will.

Notepad++ und `CMD.exe` arbeiten da also recht gut zusammen. Naken macht auch immer brav ein Listing aller Forthworte, weil sie auch die Assembler–Labels sind. So kann man gut erkennen, ob Naken irgendwo „hängengeblieben“ ist beim Übersetzen. Das Ergebnis seiner beiden Durchläufe schreibt `naken_asm` in die Konsole. Durch Umleitung in ein `error.log` bleibt es gespeichert.

War der Assembler fehlerfrei durchgelaufen, kam die Erprobung im Target. Mit dem Programmer mal eben das Image ins Target geladen, und — nix. Noch keine LED an. Und auch im Terminal noch keine Reaktion. Das ist der frustrierende Teil, denn es gibt keinen Debug–Mode, so wie in einer ordentlichen IDE. Aber mit etwas Diskussion und geschicktem Setzen von Kontrollpunkten, wie LED an/aus und `KEY` und `EMIT` im Bootvorgang, hangelt man sich durch, bis alles funktioniert, das kennt man ja schon.

Bei dieser Portierung ging das auch nicht ganz glatt, aber der Fehler war schnell eingekreist, siehe unten.

[1] Das sind die schnellen, weil assemblierten, Code–Stückchen; Low–Level–Forth–Worte genannt.

[2] Das klingt jetzt wie: erst das eine, dann das andere machen. So ist es aber in Wirklichkeit nicht. Tatsächlich hangelt man sich da so nach und nach hoch. Mal ein Assembler–Probelauf, Fehlermeldungen, Fehlersuche, verstehen, was passiert sein könnte, Optimierung des Durchlaufs usw., bis die „Tools“ verstanden worden sind und man produktiver wird.

Was war dann zu tun?

Hat man seine Entwicklungsumgebung soweit stehen, geht es ans Studieren der Quelldatei. Was muss umgeschrieben werden?[2] Gleich nach dem Vorspann wird man schon fündig:

```
MCU equ 0
; 0 : FR5969
; 1 : G2553

; Include MCU specific register data
.if MCU
.include "msp430g2553.inc"
.else
.include "msp430fr5969.inc"
.endif

.include "header.inc" ; add header macros

; Direct Thread Model of eForth
...
```

Um vom FR5969 auf den FR5739 umzubauen, muss zunächst überprüft werden, ob die ganzen Adressen der Peripheriebausteine wie Ports, UART usw. sich unterscheiden. Das `msp430fr5969.inc` unterschied sich nicht sehr vom `msp430fr5739.inc`. Man kann das manuell nachhalten. Ich habe in dem Fall eine Anleihe beim CCS gemacht. Da gibt es eine komplette inc–Datei für jeden MCU–Typ, die man sich da rauskopieren kann. Dann sind schon mal alle Registernamen und ihre Adressen richtig. Hat man sein `inc` gefunden, klar, kommt da

```
.include "msp430fr5739.inc"
```

an die Stelle. Man tut gut daran, das, was da vorher stand, stehen zu lassen. Also so zu verfahren:

```
;.include "msp430fr5969.inc"
.include "msp430fr5739.inc" ;mk
```

Ich mach mir auch immer ein Kennzeichen „an den Rand“, das `;mk`, um alle Stellen wiederfinden zu können, an denen was geändert wurde. So kann man sich später auch mit Winmerge einen Überblick darüber verschaffen, was geändert worden ist.

Nun kann man sichten, was sonst noch alles zu tun wäre. Eine Suche nach „`.else`“ bringt einen zu all den Stellen, an denen das FRAM– anders als das Flash–Modell ist. Das sind so 20 Stück an der Zahl, also gar nicht sooo viele.

Was sind das für Stellen?

Memory Map

```
; .else ; FR5969 ;MM+220119
.else ; FR5739 ;MK+220210
SOR equ 01C00H ;start of SRAM FR5739 == FR5969
;EOR equ 023FFH ;end of SRAM FR5969
EOR equ 01FFFH ;end of SRAM FR5739  ;mk
...
```

Na klar, der „kleine Bruder“ hat weniger RAM, das startet aber an der gleichen Stelle. Ob 1024 Bytes reichen? Wir werden sehen. Die Zuweisungen der Stackgrößen wurden übernommen.

Und es gibt weniger FRAM:

```
;start of Forth code/data in FRAM FR5969
;CODEE equ 04400H

;start of Forth code/data in FRAM FR5739
CODEE equ 0C200H
```

Das FRAM liegt aber, wie beim großen Bruder auch, am Ende des Adressraumes, startet also einfach nur an einer höheren Adresse.

Und auch die Lage und Größe des INFO–FRAMs sind gleich, keine Änderung nötig.

```
;save area for the user variables
INFOD equ 01800H ; FR5739 == FR5969
.endif
```

Das war einfach. Die Werte findet man im Datenblatt der speziellen MCU, welches bei TI zum Download frei zugänglich da lag, angenehm.

I/O

Als Nächstes landen wir dann schon beim I/O. Die Forthworte `?KEY KEY EMIT` arbeiten erfreulicherweise bei beiden FRAM–Varianten über denselben UART. Hier ist also keine Anpassung nötig. Doch die Initialisierung dieses I/O–Devices unterschied sich dann doch. Im `!IO` musste diese eine, aber entscheidende Stelle modifiziert werden:

```
; DCO=8MHz, default FR5969
; mov #0x000C,&CSCTL1

; DCO=8MHz, default FR57xx
mov #0x0006,&CSCTL1
```

Danach habe ich dann suchen müssen! Dass dieser Default nicht stimmte, war der Grund dafür, dass zunächst kein `OK` vom eForth kam, obschon Naken schon längst alles fehlerfrei übersetzt hatte.

Hat man schon mal Forth auf eine andere Maschine portiert, kennt man diese Fallstricke allerdings schon: Da stimmt was mit der Baudrate nicht. Und dann heißt es: eintauchen in das Clocksystem der Maschine, um zu finden, was da justiert gehört. Das kann dauern. Glücklicherweise stimmten die übrigen Registereinstellungen alle überein. Und dann lief es auch, das ersehnte `OK` war da. Natürlich waren alle folgenden Stellen da schon richtig.

Was speichern!

Das grundlegende Problem war schon gelöst von Manfred. Statt der komplizierten Art, das Flash im G2553 zu beschreiben, verhält sich das FRAM wie ein RAM. Das `I!` des eForth, das in den nichtflüchtigen FRAM–Bereich schreiben kann, ist beim großen und kleinen Bruder identisch. Da musste auch nichts gemacht werden, außer dieses festzustellen.

```
; I! ( n a -- )
; Store n to address a in code dictionary.
HEAD( 2,"i!" )
ISTORE:
.if MCU ; G2553
mov #FWKEY,&FCTL3 ; Clear LOCK
mov #FWKEY+WRT,&FCTL1 ; Enable write
; call #STORE
mov.w @stack+,0(tos)
pops
mov #FWKEY,&FCTL1 ; Done. Clear WRT
mov #FWKEY+LOCK,&FCTL3 ; Set LOCK
.else ; FR5969 == FR5739
mov.w @stack+,0(tos)
pops
.endif
INEXT
```

Hier hat es also genügt, sein `FR5969 == FR5739` an den Rand zu schreiben, damit das abgehakt war. Das Gleiche galt für das Wort `APP!` , mit dem ein Eintrag in das InfoD–FRAM vorgenommen wird, damit eine Anwendung beim Booten starten kann. Das hieß früher mal „make a turnkey applikation“, also das Endprodukt „schlüsselfertig“ wie ein Fertighaus zu übergeben.

`NOAPP?` war dann die nächste Stelle, an der es nichts zu tun gab. Diente das im G2553LP noch als Bypass, um, ohne die APP zu starten, direkt ins eForth zu kommen, ist das wohl in der FRAM–eForth–Version fallengelassen worden. Hier war also auch einfach nur `FR5969 == FR5739` zu vermerken. Immerhin kann man sich da schonmal Gedanken machen, wie man das dereinst in seinem Forth eigentlich haben will. Das kann eine kritische Stelle sein: Will man zulassen, an einer APP vorbei ins Forth zu dürfen? Beim MSP430G2553LP für Studenten fanden wir: Ja! Man kann schließlich beim Erlernen des interaktiven Forthprogrammierens einiges falsch machen, und sich mit seiner App das Maschinchen „zuzumachen“, obschon ja das Forth darin noch intakt ist, geht. Da ist es schon gut, einen Boot–Bypass zu haben, mit dem das intakte Forth zurückzuholen ist, und man nicht bei jedem Fehler den Chip neu programmieren muss. Beim FRAM ist es insofern anders, als der eForth–Kern da ohne weiteres flott überschrieben werden kann. So, dass die Chance auf einen intakten Forthkern nach Absturz seiner `APP` wirklich nicht groß ist. :) Man kann FRAM aber auch schreibschützen. Das muss wohl noch als Feature dazu!

`INIT` und `COLD`

Klassischerweise fährt Forth sein System hoch, indem `COLD` ausgeführt wird. Das ist eine High–Level–Routine und gewöhnlich nicht mehr maschinenabhängig. Und so ist es auch in unserem Fall. Doch bevor `COLD` loslegen kann, so einiges in den Grundzustand zu versetzen, muss es seine beiden Zeiger für die Stacks im RAM, den Daten und den Returnstack, über die Forth ja arbeitet, initialisiert haben, sonst läuft da nichts. Erfreulicherweise unterscheiden sich die beiden FRAM–Typen auch an dieser Stelle nicht voneinander, also nochmal `FR5969 == FR5739`

```
;mk  jmp was out of range.
;mk  put it closer to cold.
init:
mov #RPP,SP ; set up stack
mov #SPP,stack
clr tos
mov #DOLST,R15 ; MM-191024
.if MCU ; G2553
; Stop watchdog timer
mov.w #WDTPW+WDTHOLD,&WDTCTL
bis.b #041h,&P1DIR ; P1.0/6 output
.else ; FR5969 == FR5739
; disable watchdog
mov.w #WDTPW|WDTHOLD,&WDTCTL
.endif
jmp COLD
```

Es gibt am Ende von `COLD` noch eine Überlegung, ob eForth eine boardspezifische Aktion machen soll:

```
.if MCU ; G2553
.dw FSCAN,CR ;MM++180629
.else ; FR5969 == FR5739
.dw CR
.endif
.dw QUIT ;start interpretation
```

Das war hier nun auch ersteinmal nicht der Fall. Und nun sind wir auch schon fast durch mit der Anpassung. Eine MCU–ID soll noch erfolgen, damit Mensch und Maschine später wissen, was da drin ist.

```
; MCU-ID ( -- ) ;MM++220123
MCUID:
.if MCU ; G2553
HEAD(IMEDD+7,"-G2553-" )
.else ; FR5739
HEAD(IMEDD+8,"-FR5739-" )
.endif
INEXT
```

Damit kann das Terminal `e4thcom` feststellen, was an Forthsource nachgeladen werden könnte, oder was nicht passen wird.[3]

[3] Ja, in der Linuxwelt ist da mehr möglich, als im Win10. Wie man das eForth dort benutzt, ist beim Terminal `e4thcom` beschrieben.

Eine wichtige Funktion von COLD ist es, die User–Area in den Grundzustand zu setzen. Dort sind alle User–Variablen versammelt, die das Forth selbst zum ordentlichen Betrieb benötigt.

Da eForth diese Tabelle bei allen MSP430–Typen (bisher) im gleichen InfoD–Bereich hat, und der an identischer Stelle im Adressbereich geblieben ist, war auch hier nichts zu tun — siehe weiter unten.

Schön zu sehen ist an der User–Area, dass sich etwas geändert hat vom ursprünglichen eForth hin zum eForth mit VIS: Es gibt nun Pointer für CURRENT und CONTEXT und VOC und Register für CSR und LBB — siehe dazu Manfred Mahlows Beschreibung des VIS in [2].

Und auch der Bootvektor selbst liegt bei allen Modellen ganz oben im Speicher.

```
.org 0FFFEh
DC16 init ; FFFE - Reset
```

Auch da gab es also nichts weiter zu tun. Ich erwähne diese Stellen nur, um zu zeigen, wo man auch mal hinschauen könnte auf der Suche nach Spezifischem.

So sind wir nun mittels der Suche nach „.else“ durch den ganzen Assemblerquelltext gesprungen, hatten dort aber eigentlich wenig zu tun.

„Keep the user happy“–LED

Doch eine Stelle fehlt noch: Am Ende von !IO versteckt sich noch ein Stückchen Code, der strenggenommen ja auch nicht zum eForth selbst gehört, sondern dem Board geschuldet ist, auf dem es laufen soll. Je nachdem, was da schon beim Reset angesteuert werden soll, muss man das dort noch schnell machen.

```
;mk  Enable and set LED1..8
;    of MSP-EXP430FR5739 LP
BIS.B #0x0f,&PJDIR ; LED1..4
BIC.B #0x0f,&PJOUT ; LEDs off
BIS.B #0x08,&PJOUT ; LED4 on
BIS.B #0xf0,&P3DIR ; LED5..8
BIC.B #0xF0,&P3OUT ; LEDs off
BIS.B #0x10,&P3OUT ; LED5 on
```

Sonst bleibt das Launchpad „reglos“ da liegen und tut keinen Mucks, obschon alles innen funktioniert. Dieses Zeichen, dass der Reset richtig durchlaufen wurde bis hin zum !IO, ist doch immer wieder beruhigend und schließt schon mal einiges an Fehlern aus, die man da gemacht haben könnte. :)

So, nun ist es geschafft. Unser eForth hat sein OK auch für den FR5739 gegeben. Bleibt auszuprobieren, wie es sich darauf bewährt. Auf dem großen Bruder hat es sich bereits bewährt.

Literatur und Links

[1] Zen and the Forth Language: EFORTH for the MSP430 von Texas Instruments. Kindle Edition.

[2] Forth Magazin „Vierte Dimension“, Heft 4d2019–04, S. 11 ff — mit Glossar zum VIS und weiteren Links bzw. Literaturhinweisen. Im 4D–Archiv: https://wiki.forth-ev.de/doku.php/vd-archiv

[3] e4thcom — Ein Terminal für eingebettete Forth–Systeme https://wiki.forth-ev.de/doku.php/projects:e4thcom

```
UZERO:    ;; User variables
;  init value  ; offset to UPP
  .dw HI       ; 000H boot routine
  .dw BASEE    ; 002H BASE
  .dw 0        ; 004H tmp
  .dw 0        ; 006H #TIB
  .dw 0        ; 008H >IN
  .dw 0        ; 00AH HLD
  .dw INTER    ; 00CH 'EVAL
; .dw COLD-6   ; 00EH CONTEXT pointer                          ;MM--180713
  .dw MCUID+2  ; 00EH DIC    ; points to na of last word in the dictionary
  .dw CTOP+8   ; 010H CP
  .dw DPP      ; 012H DP
; .dw COLD-6   ; 014H LAST
  .dw MCUID+2  ; 014H LAST
  .dw 0        ; 016H CURR   ;MM++180712  new CURRENT pointer
  .dw 0        ; 018H CONT   ;MM++181003  new CONTEXT pointer
  .dw 0        ; 01AH VOCP   ;MM++180713  new VOC context pointer
  .dw 0        ; 01CH CSR    ;MM++181124  context switch request, wid(voc) or 0
  .dw 0        ; 01EH LBB    ;MM++181208  buffer to collect lexicon bits
ULAST:
```

Stringstack

Michael Kalus

Grundlegende Betrachtungen dazu hatte Klaus Schleisiek *angestellt und beispielhaft in Forth codiert. Sein Beitrag kann im Forth–Magazin „Vierte Dimension“ von 1986, Heft 1, nachgelesen werden. Er kam zu dem Schluss: „Strings haben ein solch eigentümliches Format, dass man am besten einen eigenen Stack nebst den notwendigen Operatoren anlegt, um Strings bequem und flexibel zu manipulieren.“ Um hier gut folgen zu können, empfehle ich den Quellcode kopiert neben den Beitrag zu legen zum Mitlesen darin. Ihr werdet sehen, wie einfach es ist, diese Operatoren zu machen, sobald deren Komponenten einmal gefunden worden sind. Im folgenden Text werden die Operatoren zusammen mit ihrem Stack–Bild vorgestellt, damit man gleich sieht, was sie machen. Es zeigt zuerst den Datenstack und danach den Stringstack.*

Klaus hatte damals angegeben, welche Operatoren das sein müssten.

- Stackverwalter — nötig, um die eigentlichen Stringoperatoren zu bauen, um die sich aber der Benutzer des Stringstacks nicht zu kümmern braucht — `TOS$ PICK$ ...`
- Stringbeweger — `"DUP "SWAP ...`
- Stringmanipulatoren — `"SPLIT "JOIN ...`
- Stackbeschicker — `"PUSH "POP ...`

Die Strings werden in Klaus' Modell tatsächlich im Speicher hin und her bewegt, dort geteilt, geschoben, vereinigt. `CMOVE` und `PLACE` sind daher oft genutzte Funktionen. Konrad Scheller war das zu langsam, er hat das schneller gemacht [3]

Aufbau des Stringstacks

Weil Klaus das Grundlegende schon geklärt hatte, kann ich mich nun darauf beschränken, meine zum Vergnügen angefertigte Codierung seiner Ideen vorzustellen.

Der Speicherbereich für den Stringstack ist mitten im Dictionary und wird mit `CREATE... ALLOT` zugewiesen. Mit den beiden Zeigern `TOS$` und `BOT$` wird der Stack verwaltet, sie sind in der Datenstruktur am oberen Ende eingebettet.[1] Sie werden bei der Kreation des Stringstacks initialisiert — in `create-stringstack` sind das diese seltsamen Passagen mit `here` und so.

Die globale Variable `CSP$` würde dabei helfen, mehrere Stacks zu halten und kann getrost ignoriert werden, da wir es heute nur mit *einem* Stack zu tun haben werden. Der Vector `'SP$` würde die Stackumschaltung machen, das kann heute auch mal ignoriert werden.

Ob man sich ein eigenes „Defining Word“ macht, um damit weitere Stringstacks zu generieren, oder nur einen einzelnen Stringstack anlegt, hängt sicherlich davon ab, was man in seiner Anwendung erreichen möchte. Ich hatte damals übungshalber den Weg über das Defining Word genommen. Und das auch erst im zweiten Durchgang, einfach, um mal zu sehen, wie das so geht.

Einfach so im Speicher ließe sich das aber auch machen:

```
here  &1000 allot align
here  constant SP0$
      constant bot$
      variable SP$
```

Oder so ähnlich. Die beiden `here` merken sich Anfang und Ende des Speicherbereichs und werden als Konstanten aufgehoben. Dann kann man mit diesen alle Zugriffe in den Stringstack–Bereich berechnen und globale Zeiger mitführen. Das geht ja auch.

Nachdem der Stringstack also angelegt ist, so oder so, braucht es eine Reihe Verwaltungs–Operatoren, um darauf zuzugreifen. Diese drei — `SP$ TOS$` und `BOT$` — geben Auskunft darüber, wo was ist. `SP$` zeigt auf den Anfang der Struktur, die ja *nach unten* wächst, also ist es die hohe Adresse, `TOS$` hingegen zeigt auf den String, der zuoberst auf dem Stapel ist, wandert also mit jedem String, der dazukommt, nach unten. Und `BOT$` zeigt an, wo das Ende des reservierten Platzes ist.

Die ganze Sammlung der `?xxx`–Worte dient der Fehlerbehandlung, wie Über– oder Unterlauf des Stacks und Passungen der Strings. Da ist der Fantasie keine Grenze gesetzt, falls ihr noch mehr abfangen wollt.

Der Stringstack wächst nach unten!

Sein Aufbau ist so:

```
top of allocated memory
 bot$ <-- 1 cell for bottom addr
 sp$  <-- 1 cell for stack pointer
  ... place top string
 top$ <-- top string count byte
  ... place second string
 sec$ <-- second string count byte
  ... place more strings
  ...
  ...
 last$ <-- last string count byte
bottom of allocated memory
(sp$) <-- 1 cell pointer to sp$
header of created stringstack
```

[1] Daher könnten mehrere Stringstacks angelegt werden. Ob das unnötig kompliziert ist, sei nun mal dahingestellt. Es geht jedenfalls, nehmen wir es mal als Machbarkeitsstudie.

Die $–Positionen

Um im Stack irgendwas bewegen zu können, muss man wissen, wo man ist. Die Positions–Operatoren liefern die gesuchten Adressen.

Zum „Top of Stringstack“ bringt uns `TOS$`. Um von da aus zum nächsten String zu kommen, müssen wir aber immer dessen Länge berücksichtigen — richtig: Es wird dynamisch zugewiesen, nicht mit fixen Längen. `SKIP$` bringt einen zum Anfang des nächst tieferen Strings. Damit ist

```
PICK$ ( n -- adr )
```

auch gelöst. Der Bequemlicheit halber gibt es auch `SEC$` und `LAST$`, und natürlich muss man auch mal die Stacktiefe wissen, also auch `DEPTH$` ist da.

Ein einzelnes Zeichen im `TOS$` verändern

Im obersten String des Stapels kann man damit nun schon einzelne Zeichen manipulieren. Ein

```
GET$ ( n -- char )
```

und ein

```
PUT$ ( char n -- )
```

machen das — `TOPCOUNT$` und `TOPLENGTH$` und `TOPLOC$` sind lediglich die ausfaktorisierten Teile davon, man wird sie kaum eigenständig benutzen, auch `EXTRACT$` und `PATCH$` dienen nur der Adressberechnung für die eigentlichen Operatoren `"PATCH` und `"EXTRACT` — wir kommen gleich dahin.

Strings auf den `TOS$` legen und wieder entfernen

Bisher haben wir gesehen, wie die Positionen im Stack bestimmt werden können, aber noch nicht viel damit gemacht. Denn dazu muss man ja überhaupt erst einmal Strings auf den Stack schaffen.

Der Operator

```
"PUSH ( adr len -- ) ($ -- s )
```

macht das. Damit kann ein typischer „Counted String“, dessen Adresse und Länge bekannt ist, auf den Stringstack kopiert werden.

So eine klassische Terminal–Eingabe wie:

```
s" eins" "push <ret>
```

bringt uns den String „eins“ in den `TOS$`.

Und dementsprechend befördert

```
"POP ( -- adr len ) ($ s -- )
```

ihn wieder herunter (ist dabei aber so nett, uns zu sagen, wo der String denn gewesen ist, falls wir den noch woanders hintun wollen.)

Dann gibt es noch ein paar spezielle Operatoren in dieser Gruppe, die ich mir damals ausgedacht habe. Ob die nützlich sind, weiß ich auch nicht — waren ja „Fingerübungen“. Ihr könnt sie also auch ignorieren.

So gibt es da ein

```
"CHAR ( char -- ) ($ -- s ),
```

das ein einzelnes Zeichen zum String der Länge 1 im `TOS$` macht. Oder das

```
"@ ( adr -- ) ($ -- s ),
```

welches das andere Stringformat benutzt, also nur die Adresse bekommt, an der ein Counted String liegt und den dann auf den Stack packt; und natürlich das Gegenstück

```
"! ( adr -- ) ($ s -- ).
```

Und ein

```
"COPY ( adr -- ) ($ s -- s ),
```

um dasselbe zu bewerkstelligen, ohne den `TOS$` zu löschen.

Strings auf dem Stack bewegen

So, nun sind wir endlich bei den eigentlichen String–Operatoren angelangt, die ganz wie die alten Bekannten vom Datenstack des Forth arbeiten. Da ihr ja schon die ganze Zeit im Quellcode mitgelesen habt, wird euch an dieser Stelle auffallen, wie *einfach es auf einmal war*, diese Operatoren zu formulieren. So ein `"PICK` ist plötzlich nur noch ein simples `PICK$ "@` und ein `"OVER` schrumpft zum `1 "PICK` zusammen. Das sorgfältige *Ausfaktorisieren* zuvor, das zunächst verwirren mag, hat's gebracht! Und das kam auch nicht einfach so als plötzliche Eingebung, sondern hat sich so nach und nach eingestellt, indem der anfänglich noch wirre Code immer wieder daraufhin durchgesehen wurde, was noch faktorisiert werden könnte. Ich weiß nicht mehr, wie viele Faktorisierungszyklen das gewesen sind, aber es hat Tage gedauert, bis ich das Prinzip erkannt hatte. Und wenn man das dann hat, kommt richtig Freude auf beim Codieren der Operatoren, die dann ganz einfach zu machen sind.

`"EMPTY "CLEAR "DROP "DUP "ROLL "ROLLDOWN "SWAP "ROT` — das flutschte dann nur so, man möchte gar nicht aufhören, neue Operatoren zu kreieren. :)

Manipulieren der oberen beiden Strings

So ausgerüstet, wagt man sich dann an noch mehr Stringmanipulationen. Das klassische

```
"JOIN ( -- ) ($ a b -- ab )
```

braucht dann schon etwas mehr Adressjonglage und Zeichenschieben, aber ist nun auch nicht mehr sooo die Kunst mit den Positionsbestimmungswerkzeugen, die ja nun schon gemacht sind. Und auch das

```
"SPLIT ( n -- ) ($ ab -- a b )
```

fällt dabei ab. Und ein

```
"PATCH ( n -- ) ($ abcd xx -- axxd )
```

und ein

```
"EXTRACT ( n1 n2 -- ) ($ asb -- s )
```

sind dann plötzlich kein Hexenwerk mehr, und auch das

```
"INSERT ( n -- ) ($ ab s -- asb )
```

gelingt. Man kann ganz in Forth denken, als ob man auf dem Datenstack herumturnen würde — herrlich, oder? Ihr habt ja den Code neben euch liegen, um darin mitzulesen? :) Und das

```
"FILL ( c -- ) ($ s -- cc )
```

und

```
"BLANK ( -- ) ($ s -- bl )
```

muss man nun nicht mehr erklären, oder? Ob man so etwas braucht und einbauen sollte, mag jeder für sich entscheiden.

Was man aber haben möchte, sind Operatoren, um Zeichen an einen String anzufügen, hintendran mit

```
"APPEND ( c -- ) ($ s -- sc )
```

und vorn mit

```
"INFRONT ( c -- ) ($ s -- cs ).
```

Lustig fand ich das

```
"ENROL ( char -- ) ($ s - s' ).
```

Dabei wird dem String links ein Zeichen abgeschnitten und rechts ein Zeichen angefügt. In einer Schleife eingebaut gäbe es eine Laufschrift.

Analog dem `blank` für die Ausgabeformatierungen im Forth, dachte ich,

```
"BLANKS ( len -- ) ($ -- s )
```

könne nützlich sein. Es macht einen String mit Leerzeichen, der dann eben leer aussieht bei der Ausgabe, aber die Länge `len` hat.

```
"SUPP ( len -- ) ($ s -- s bl )
```

hingegen erzeugt einen String mit Leerzeichen, der *zusammen* mit dem Topstring die Länge `len` hat. Das ist eigentlich nur der Faktor für `"L` und `"R`, mit denen man eine bestimmte Anzahl `blanks` links oder rechts an einen String anfügen kann.

Die obersten beiden Strings vergleichen

Diese Operatoren benutzen das `compare` des Forthsystems, in dem man seinen Stringstack macht, in der Hoffnung, dass es schon das richtige sein wird. So liefert

```
COMPARE$ ( -- n ) ($ s1 s0 -- s1 s0 )
```

die Flags ebenso, wie es das Forthsystem beim Stringvergleichen macht, wenn es Worte in seinem Dictionary sucht. Wie man sieht, bleiben die Strings unverändert erhalten und das Ergebnis des Vergleichs erscheint auf dem Datenstack. Ob das

```
"COMPARE ( -- n ) ($ s1 s2 -- )
```

dann überhaupt Sinn macht, bei dem nach dem Vergeich beide Strings weg sind? Im Forth ist das auf dem Datenstack ja so gemacht, dass die zu vergleichenden Daten durch den Vergeichsoperator verbraucht werden. So habe ich das hier denn auch mal nachgestellt und `"= "<` und `"<=` liefern ihr Flag, doch die Strings sind anschließend futsch. Ihr könnt das verwerfen, denke ich. Man wird mit dem nicht–destruktiven `COMPARE$` auskommen.

I/O am Stringstack

So, schließlich will man die Ergebnisse der Stringbearbeitung ja auch sehen. Auch diese Operatoren ergaben sich fast wie von selbst, simple Einzeiler dank der schon erwähnten Faktorisierung. Es sind die Pendants der Standard–Forth–Ausgabeworte.

```
"TYPE ( -- ) ($ s -- s )
```

druckt nicht–zerstörend den `TOS$`, während

```
". ( -- ) ($ s -- )
```

den `TOS$` verbraucht.

```
".R ( len -- ) ($ s -- )
```

und

```
".L ( len -- ) ($ s -- )
```

machen das rechts– bzw. linksbündig. Und das

```
"AT ( col row -- ) ($ s -- )
```

druckt den `TOS$` im Terminal an die angegebene Stelle. Und schließlich dachte ich, ein

```
"EXPECT ( len -- ) ($ -- s )
```

könne auch nicht schaden. Das stellt im `TOS$` einen Puffer der Länge `len` bereit, der mit Leerzeichen vorbelegt ist. Dort hinein schreibt das Forth `expect` bis zum `cr` und anschließend wird dieser Puffer auf das tatsächlich Geschriebene gekürzt und als `TOS$` hinterlegt — nützlich? Keine Ahnung, eben eine „Fingerübung“ :)

Debugging

Bei der Entwicklung dieses Stringstacks war es natürlich nötig, nachzusehen, was sich denn auf dem Stringstack gerade so getan hat, so wie man das im Forth mit dem Datenstack ja auch macht.

Den Stack drucken, das braucht man. Und auch ein Dump einzelner Strings oder des gesamten Bereichs. So gibt es im Listing auch noch meine Hilfen dafür. Da hat sicher jeder noch andere Wünsche und bessere Werkzeugideen.

Das `".S`, um den Inhalt des Stringstacks nicht–destruktiv zu drucken, fand ich sehr hilfreich. Es zeigt ja auch, dass die Zeiger funktionieren und der Stack richtig arbeitet.

Auch das

```
"DUMP ( -- ) ($ s -- s )
```

für den TOS$ kann man gebrauchen, denke ich, um Fehler in den Zeichenmanipulationen aufzuspüren.

Und den ganzen Stack anzusehen, brauchte ich auch hin und wieder. Also gibt es da auch das

```
DUMP-S$ ( -- )
```

(was natürlich etwas geschummelt ist, weil die Stapelgröße darin nur für das Beispiel gemacht ist, und nicht selbst nachschaut, wie groß der Stapel denn ist; aber man muss es ja nicht übertreiben).

Beispieleingabe

Meine simple testweise Stringeingabe am Terminal, an der ich vieles ausprobieren konnte, war das hier:

```
s" eins" "push
s" zwei" "push
s" drei" "push
s" vier" "push
".s .s
```

So, und nun viel Vergnügen bei euren Stringstackübungen und den Fingerübungen mit Forth.

Und wer wissen will, wie es ANS–konform, eleganter, eben professioneller geht, liest noch ein bisschen weiter ...

Der Stringstack der Profis

Ulrich Hoffmann hat 2015 eine professionelle Weiterentwicklung erstellt, die seitdem viel in Gebrauch ist. Das Paket ist hinterlegt bei theforth.net [4].

> „This package provides an implementation of a string stack to Forth–94 and Forth–2012 systems. It is based on a string stack implementation by Klaus Schleisiek from 1986 but instead of character counted strings it uses *cell counts*. No interpretation of the strings takes place so it can as well process arbitrary binary data.“

Kurz was zur Handhabung: Es reicht schon das einfache „Tüttelchen oben“, um einen String auf den Stringstack zu befördern, kein umständliches "PUSH mehr nötig. Gedruckt wird aber auch mit ". , wie oben schon erwähnt. Eine Terminaleingabe liest sich dann z. B. so:

```
" a string" ". a string ok
```

oder so:

```
" Forth String World!"
" Hello, " "join
". Hello, Forth String World! ok
```

Und da sind auch kompliziertere Operatoren:

```
" name=$(name)"
" $(name)"
" Forth"
"substitute
". name=Forth ok
```

Genug Appetitanregung?

Fragt gerne nach, Ulli oder Klaus erklären gerne ihren moderneren Ansatz. In Kontakt kommt ihr über das Forth–Büro (siehe Impressum).

Literatur und Links

[1] Schleisiek, Klaus; Stringstack; 4d1986–01S. 22 ff

[2] Konrad, Karsten; Read–only–Stringfelder in FORTH; 4d1989–03 S. 30

[3] Scheller, Konrad; Neuer Stringstack mit Heap; 4d1989–04 S. 24 ff
Die PDFs sind im VD–Archiv [6].

[4] https://theforth.net/package/stringstack

[5] https://wiki.forth-ev.de/doku.php/examples:stringstackmk

[6] https://wiki.forth-ev.de/doku.php/vd-archiv

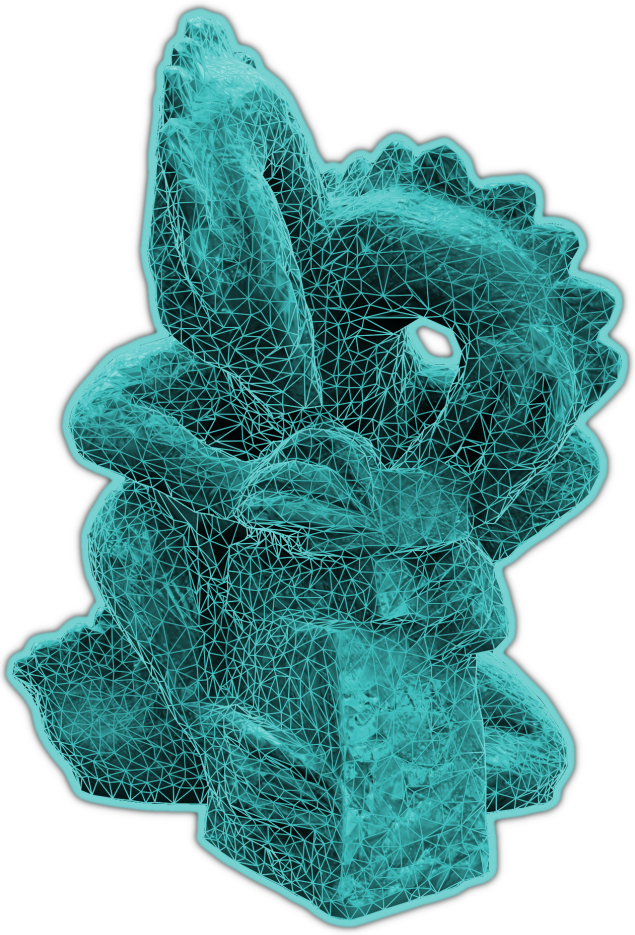


Abbildung 1: Stacks, Stacks, Stacks ... in Forth ganz leicht zu machen. (Da seht ihr übrigens den ganzen Titelbild–Swap.)

Listing

```
\ string stack - f83 1986 mka
\ 06.03.2013: adapted to gforth
\ 09.03.2013: geht im Prinzip wieder. debugging...
\ 10.03.2013: alles ok.
\ 04.03.2022: redaktionelles; ok in gforth

vocabulary stringstackwords
stringstackwords definitions

decimal

: ($   postpone ( ; immediate
\ ($ -- ) is a stringstack comment
\ ( -- )  is a data stack comment

variable 'SP$       \ string stack vector
variable CSP$       \ current stack pointer

: create-stringstack    ( "name n -- )
  \ stack defining word - why not :)
   create
   >r
   here 0 ,         \ does> adr
   r@ allot align   \ n allot == stringstack
   dup r> erase     \ erase stringstack
   here over !      \ top to does> adr
   here 'sp$ !      \ init stringstack vector
   here ,           \ init sp@ to itself
   cell+ ,          \ save bottom adr
   does> @ 'sp$ !   ; \ on calling init vector

1000 create-stringstack $S
\ $S  \ run $S to switch to this stringstack

\ stringstack administration
: SP$   ( -- adr )  'sp$ @ ;
: TOS$  ( -- adr )  sp$ @ ;
: BOT$  ( -- adr )  sp$ cell+ @ ;
: !CSP$ ( -- )
  tos$ csp$ ! ;  \ store current stack pointer

\ error messages
: ?CSP$ ( -- )
  tos$ csp$ @ <> abort" $stack changed" ;
: ?OFL  ( adr -- )
  bot$ u< abort" $overflow" ;
: ?LIM  ( len -- )
  $FF00 and abort" $toolong" ;
: ?MTY  ( adr -- )
  sp$ >= abort" $underflow" ;
: ?FIT  ( len -- )
  tos$ c@ u> abort" $does'nt fit in tos$ " ;

\ stack positions
: SKIP$ ( adr1 -- adr2 ) count + ;
: PICK$ ( nth -- adr )
  tos$
  begin dup ?mty swap ?dup
  while 1- swap skip$
  repeat ;
: TOP$  ( -- adr )  0 pick$ ;
: SEC$  ( -- adr )  1 pick$ ;
: DEPTH$  ( -- n ) ($ sn..s0 )
  0 tos$
  begin dup sp$ -
  while skip$ swap 1+ swap
  repeat   drop ;
: LAST$    ( -- adr )  depth$ 1- pick$ ;

\ char addressing in top string
: TOPCOUNT$  ( -- adr+1 len ) ($ -- ) top$ count ;
: TOPLENGTH$ ( -- len ) ($ -- ) top$ c@ ;
: TOPLOC$  ( +n -- adr ) ($ -- ) top$ + ;
: GET$  ( n -- char ) toploc$ c@ ;
: PUT$  ( char n -- ) toploc$ c! ;

\ convert counts into address-length format
: EXTRACT$  ( n1 n2 -- adr len )
    toplength$ umin over - 1+ swap toploc$ swap ;
: PATCH$    ( n1 n2 -- adr len )
   2dup + ?fit toploc$ swap ;

\ move strings in stack
: "PUSH ( from.adr len -- ) ($ -- s )
  \ push string on stringstack.
  dup ?lim   tos$ over - 1-
  dup ?ofl   dup sp$ !   place ;
: "POP    ( -- from.adr len ) ($ s -- )
  \ pop string from stringstack.
  topcount$ 2dup + sp$ ! ;
: "CHAR   ( char -- ) ($ -- s )
  \ push character as string.
  here 1 "push  \ dumy string
  topcount$ drop c! ; \ place char in dummy
: "@      ( adr -- ) ($ -- s ) count "push ;
: "!      ( adr -- ) ($ s -- ) "pop rot place ;
: "COPY  ( adr -- ) ($ s -- s )
  \ non-destructive copy of top$ to buffer
  topcount$ rot place ;
: "EMPTY  ( -- ) ($ sn..s0 -- )
  sp$ dup ! ;
: "CLEAR  ( -- ) ($ sn..sm..s0 -- sn..sm )
  csp$ @ sp$ ! ;
: "DROP   ( -- ) ($ s -- )
  "pop 2drop ;
: "PICK  ( n -- ) ($ sm..sn..s0 -- sm..sn..s0 sn )
  pick$ "@ ;
: "DUP    ( -- ) ($ s -- s s )
  0 "pick ;
: "OVER   ( -- ) ($ a b -- a b a )
  1 "pick ;
: "ROLL   ( n -- ) ($ sn..s0 -- sn-1..s0 sn )
  pick$ dup "@  tos$ tuck - "pop + swap cmove> ;
: "ROLLDOWN ( n -- ) ($ sn..s1 s0 -- s0 sn .. s1 )
  pick$ skip$ dup topcount$ + tuck - tos$ swap
  "dup cmove "pop rot over - 1- place ;
: "SWAP  ( -- ) ($ a b -- b a )
  1 "roll ;
: "ROT   ( -- ) ($ a b c -- b c a )
  2 "roll ;

\ manipulate top and sec string
: "JOIN ( -- ) ($ a b -- ab )
  tos$ dup >r "pop dup toplength$ + r> c!
  over sp$ ! 1+ cmove> ;
: "SPLIT ( n -- ) ($ ab -- a b )
  toplength$ over - over toploc$ >r >r
  "pop drop swap over 2 -  dup sp$ !
  place r> r> c! ;
: "PATCH    ( n -- ) ($ abcd xx -- axxd )
  "pop rot patch$ cmove ;
: "EXTRACT  ( n1 n2 -- ) ($ asb -- s )
  extract$ "drop "push ;
: "INSERT   ( n -- ) ($ ab s -- asb )
  "swap "split "rot "swap "join "join ;
: "FILL     ( c -- ) ($ s -- cc )
  \ replace characters with c
  topcount$ rot fill ;
: "BLANK    ( -- ) ($ s -- bl )
  \ replace characters with blanks
     bl "fill ;
: "APPEND   ( char -- ) ($ s1 -- s2 )
  "char "swap "join ;
: "INFRONT  ( char -- ) ($ s1 -- s2 )
  "char "join ;
: "ENROL    ( char -- ) ($ s - s' )
  topcount$ 1- 2dup >r
  dup 1+ swap r> cmove + c! ;
: "BLANKS   ( len -- ) ($ -- s )
  \ make blank string
```

```
151       here swap "push "blank ;
152   : "SUPP     ( len -- ) ($ s -- s bl )
153     \ make supplement blank string
154       toplength$ - 0 max "blanks ;
155   : "L        ( len -- ) ($ s -- s_bl )
156     "supp "swap "join ;
157   : "R        ( len -- ) ($ s -- bl_s )
158     "supp "join ;
159
160   \ special string types (experimental)
161   \ : ""         ( -- ) ($ -- s ) 0 0 "push ;
162   \ : "D         ( d -- ) ($ -- s ) (d.) "push ;
163   \ : "0         ( -- ) ($ -- s )  0 0 "d ;
164   \ : "NUMBER    ( -- d ) ($ s -- )
165   \   lenght$ toploc$ c@ bl = not IF bl "append THEN
166   \   "pop drop number ;
167   \ : (D.PRICE)  ( d -- adr len )
168   \   tuck dabs <# # # ascii . hold #s rot sign #> ;
169   \ : "PRICE ( -- ) ($ s -- $US )
170   \   "number (d.price) "push ;
171
172   \ string comparators
173   : COMPARE$ ( -- n ) ($ s1 s0 -- s1 s0 )
174     top$ count sec$ count  compare  ;
175   : "COMPARE ( --- n ) ($ s1 s2 -- )
176     compare$ "drop "drop ;
177   : "=       ( -- f ) ($ s1 s2 -- )
178     "compare 0= ;
179   : "<       ( -- f ) ($ s1 s2 -- )
180     "compare 0< ;
181   : "<=      ( -- f ) ($ s1 s2 -- )
182     "compare dup 0< swap 0= or ;
183
184   \ stringstack I/O
185   : "TYPE  ( -- ) ($ s -- s )
186     \ non destructive info of tos$.
187     topcount$ type ;
188   : ".     ( -- ) ($ s -- )   "pop type  ;
189   : "AT    ( col row -- ) ($ s -- )  at-xy ". ;
190   : ".R    ( len -- ) ($ s -- )  "R ". ;
191   : ".L    ( len -- ) ($ s -- )  "L ". ;
192   : "EXPECT   ( len -- ) ($ -- s )
193      "blanks  topcount$ expect
194      "pop drop span @ "push ;
195
196   \ some debugging tools
197   : ".S   ( -- ) ($ -- )
198     \ show strings on stringstack
199     depth$ 0= IF ." Empty$ " exit then
200     tos$  BEGIN depth$ while cr ". REPEAT
201     sp$ ! space ;
202   : ?$  ( adr  -- ) ($ s -- s )  \ check string
203     count type ;
204   : "DUMP  ( -- ) ($ s -- s )    \ dump top$
205     top$ toplength$ 1+ dump ;
206   : DUMP-S$ ( -- ) \ dump entire stringstack
207     [’] $S 1100 dump ;
208
209   ( finis)
```

synonym vs. alias

Im Listing zum Stringstack wurde damals die Definition eines „Stackkommentars“ mittels `postpone` gemacht [Zeile 12], `synonym` kannte Gforth 0.7.3 noch nicht. Heute würde man einfach schreiben:

```
synonym ($ (
```

Das `alias` gab es damals zwar schon, doch hatte es genauso seine Tücken wie jene postpone–Konstruktion, denn die nehmen nicht alle Flags mit. Folgende Definition scheitert beim Compilieren, ließe sich also *nicht* als Kommentar *in* einer Definition verwenden.

```
’ ( alias (s ok
: test (s a b -- ) ;
*the terminal*:2:11: error: Undefined word
: test (s >>>a<<< b -- ) ;
Backtrace: ...
```

Ein `immediate` ist in dem Fall also notwendig, dann würde es gehen:

```
’ ( alias (s  immediate
: test (s a b -- ) ;  ok
```

Und so etwas ist besonders trügerisch (!):

```
: x" postpone s" ; immediate ok
: test1 x" bla" type ; ok
test1 bla ok
```

Soweit, so gut. Aber dann:

```
x" bla" type
*the terminal*:6:9: error: Stack underflow
x" bla" >>>type<<<
Backtrace: ...
```

`synonym` ist inzwischen Standard. Da es die Flags und alle anderen Merkwürdigkeiten, wie verschiedenes Verhalten im Interpreter und Compiler, mitnehmen kann, ist es klar besser als `alias`. Bernd Paysan

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

CollapseOS — Ein Betriebssystem (nicht nur) für die Zombie–Apokalypse

Carsten Strotmann

Wie werden wir Computer benutzen, wenn Internet, das Stromnetz, das Telefonnetz und auch die Zivilisation zusammenbrechen?

Das CollapseOS–Projekt erforscht die Möglichkeiten, mittels aus Alltagsgegenständen wie Waschmaschinen oder Heizungssteuerungen ausgebauten Mikroprozessoren einfache Computer zu erstellen. Dabei sollten diese Systeme komplett autark funktionieren, d. h., dass sich die Rechner direkt ohne weitere Hilfsmittel selbst programmieren und erweitern lassen. Das System soll simpel genug sein, so dass eine Person dieses vollständig verstehen und meistern kann.

Die erste Version von *CollapseOS* war in Z80–Maschinensprache geschrieben. Eines der Ziele ist die Portierbarkeit des Systems auf möglichst viele CPU–Architekturen, und das funktioniert mit Assembler bekanntlich nur bedingt. CollapseOS–Entwickler Virgil Dupras schreibt unter „Why Forth?“:

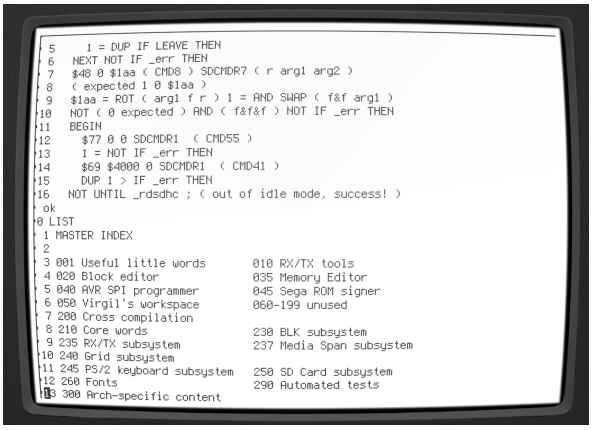
> „CollapseOS' first incarnation was written in Z80 assembler. One of the first feedbacks I had after it went viral was 'why not Forth?'. I briefly looked at it and it didn't seem such a great choice at first, so I first dismissed it. Then, I had what alcoholics refer to as a 'Moment of clarity'.“

CollapseOS ist ein kleines System, 200 Kilobyte Quellcode. Nach der ersten Version in Assemler wurde die Forth–Version auch schon das eine oder andere Mal komplett neu geschrieben und verschiedene Implementierungsdesigns ausprobiert (ITC, DTC, Superinstructions etc.) Die Evolution von CollapseOS erinnert ein wenig an die Evolution von Forth selbst, von den Anfängen in den 70er Jahren bis zu den heutigen Systemen.

CollapseOS, wie der Name schon vermuten lässt, ist ein komplettes Betriebssystem mit Forth–Kern. Anstatt eines Dateisystems werden traditionell Forth–Blöcke verwendet. Die Blöcke beinhalten den plattformunabhängigen und auch den –spezifischen Quellcode. Unterstützt werden derzeit

- Z80 (diverse Single–Board–Computer, Taschenrechner, Homecomputer)
- 6809 (TRS–80 Color Computer 2)
- 6502 (Apple II)
- 8086 (PC–Architektur)

Weitere CPU–Architekturen (ARM) und Ports (z. B. Atari 8–Bit) sind in Planung.



Von jeder Plattform kann das CollapseOS in eine der anderen Plattformen cross–compiliert werden. Dazu kommen Disassembler (6502 und 6809), ein Text–Editor nach Unix „vi–Art“ sowie Emulatoren (6502 und 6809), direkt im CollapseOS–Forth implementiert.

Das Forth des CollapseOS' ist eigenwillig, der Einsteiger muss sich mit den Eigenheiten des Forth auseinandersetzen. CollapseOS ist kein Team–Projekt, Virgil Durpas arbeitet an seiner Version, das System ist Open Source und jeder ist eingeladen, seine eigene Version zu erstellen. Ideen werden auf der Mailingliste diskutiert, aber es gibt keine Garantien, dass Änderungen von anderen CollapseOS–Entwicklern in das System eingebaut werden. Die Mailingliste ist eine geschlossene Gruppe, eine Anmeldung ist per E–Mail an Virgil Durpas möglich (Anleitung auf der Webseite).

Neben den „Bare-Metal“–Versionen des CollapseOS' gibt es auch Emulatoren für verschiende Betriebssysteme (Windows, Unix, MacOS), und auch eine Implementation in JavaScript, welche in einem Browser ausgeführt wird.

Hilft CollapseOS, den Zusammenbruch der Zivilisation abzuwenden? Wahrscheinlich nicht. Aber das ist auch gar nicht so wichtig. Es ist ein spannendes Projekt, in dem viele Ideen (alte und neue) rund um Forth ohne Berührungsängste ausprobiert werden.

`https://schierlm.github.io/CollapseOS-Web-Emulator/`

`http://collapseos.org/`

Forth–Gruppen regional

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

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

München **Bernd Paysan**
Tel.: (0 89) – 41 15 46 53
`bernd@net2o.de`
Treffen: Jeden 4. Donnerstag im Monat um 19:00 auf `http://public.senfcall.de/forth-muenchen`, Passwort `over+swap`.

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

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

Dienste der Forth–Gesellschaft

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

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

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

μP–Controller–Verleih **Carsten Strotmann**
`microcontrollerverleih@forth-ev.de`
`mcv@forth-ev.de`

Spezielle Fachgebiete

Forth–Hardware in VHDL microcore (uCore)	**Klaus Schleisiek** Tel.: **(0 58 46) – 98 04 00 8**$_p$ kschleisiek@freenet.de
KI, Object Oriented Forth, Sicherheitskritische Systeme	**Ulrich Hoffmann** Tel.: **(0 41 03) – 80 48 41** `uho@forth-ev.de`
Forth–Vertrieb volksFORTH ultraFORTH RTX / FG / Super8 KK–FORTH	Ingenieurbüro **Klaus Kohl–Schöpe** Tel.: **(0 82 66)–36 09 862**$_p$

Termine

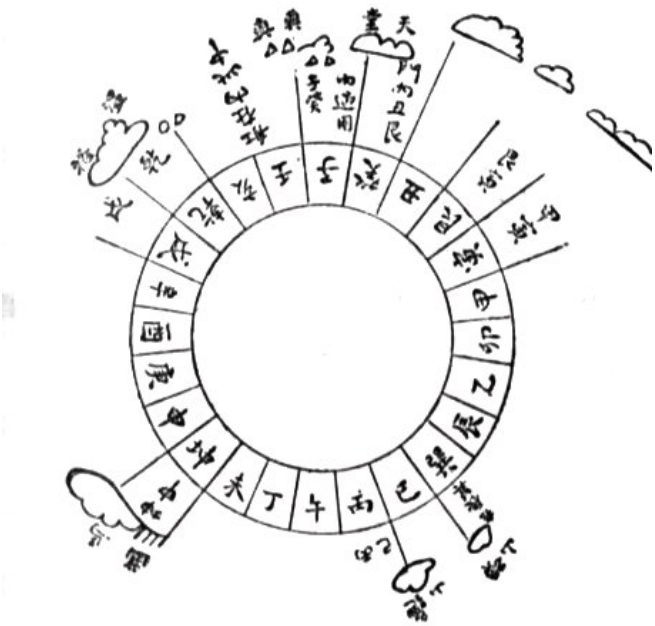
Donnerstags ab 20:00 Uhr
Forth–Chat net2o `forth@bernd` mit dem Key `keysearch kQusJ`, voller Key:
`kQusJzA;7*?t=uy@X}1GWr!+0qqp_Cn176t4(dQ*`

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

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

Forth–Tagung (online) 6.–8. Mai 2022
`https://tagung.forth-ev.de`

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.

Forthtagung online, 6.–8. Mai 2022

Direktorium

Die Bundesregierung hat die Pandemie jetzt (am 20.3.2022) offiziell „beendet“. Also, die Maßnahmen (Freiheit!). Nun ist nicht alles, was erlaubt ist, auch eine gute Idee, und manche Antworten auf Fragen, die man jetzt stellen sollte, würden die Bevölkerung verunsichern. Unsere Vereinssatzung erlaubt zum Glück eine Online–Tagung auch ohne Infektionsschutzgesetz (Eigenverantwortung!). Denn das Virus ist ja nicht weg, sondern im Gegensatz präsenter und ansteckender denn je.

Im Heft 4d2021–01 hatten wir unsere Mitglieder dazu aufgerufen, ihre Meinung mitzuteilen, in welcher Form in der pandemischen Situation das Jahrestreffen gewünscht wird. Mitglieder, welche an der Abstimmung teilgenommen haben, wünschten sich mehrheitlich eine *Online–Tagung mit Mitgliederversammlung*. Die Auszählung hatten wir im Heft 4d2021–02 an dieser Stelle publiziert. Inzwischen hat es ja eine erste solche Konferenz gegeben, einschließlich Mitgliederversammlung am 14.11.2021 und Drachenratssitzung; das Protokoll stand im letzten Heft (4d2021–04). Dort wurde beschlossen, im Frühjahr 2022 noch mal eine Tagung in diesem Format anzubieten.

Wir haben es ja schon mehrfach geübt und sind alle schon ganz „cyber cyber“ geworden inzwischen. Wird nicht so toll wie IRL[1], aber trotzdem schön! Sollte die Pandemie es zulassen, so haben wir sogar gemutmaßt, man könnte ein physisches Treffen im Sommer machen.

Doch was tun bis dahin? Wir bitten euch um Ideen und Vorträge, was ihr so in und um Forth erlebt habt.

Dieses Jahr planen wir auch eine kleine Exkursion: Zu einem physischem Treffen kann man sein(e) Bastel–zimmer/–ecke/–tisch/–kiste nicht so leicht mitnehmen. Aber gecybert geht das ganz flott. Wer mitmachen will, bereitet bitte 3 Bilder vor und ca. 1 min Plaudern dazu. So sind wir uns doch ein bisschen näher in der Ferne ;)

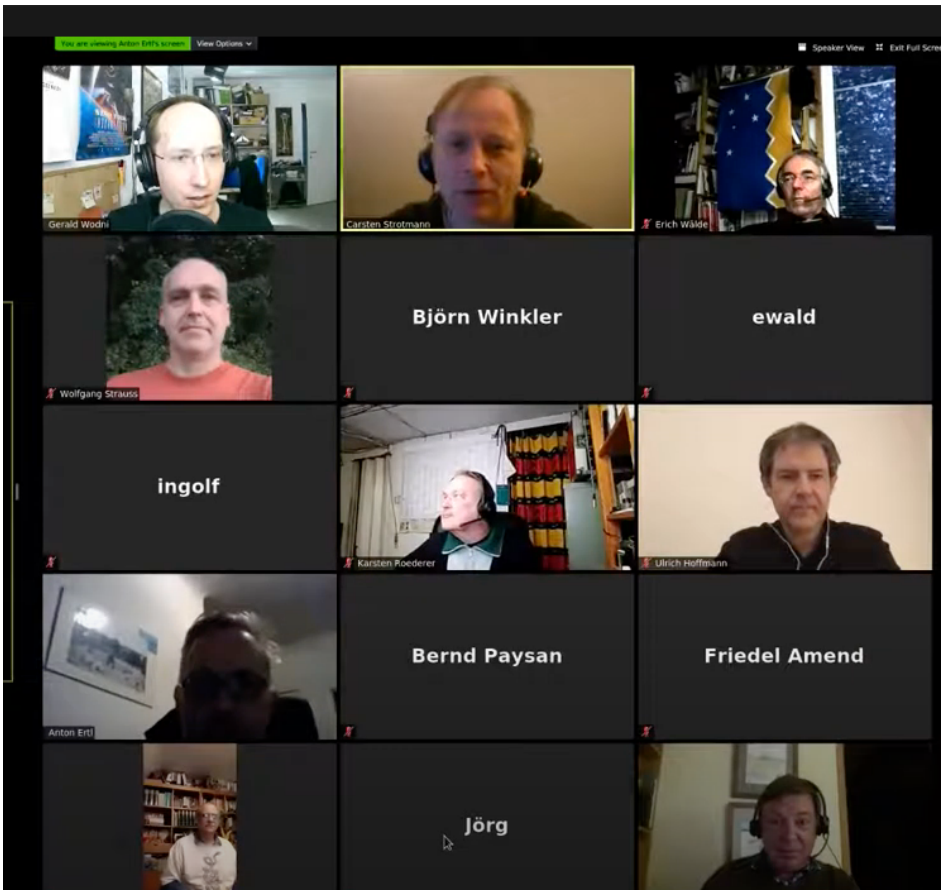
Eure Ideen und Vorschläge sendet bitte *nun* an `direktorium@forth-ev.de` oder per Brief an die *neue* Büroadresse der Forth–Gesellschaft.

Wir freuen uns auf rege Rückmeldungen.

Gleich anmelden!

Das Anmeldeformular zur Tagung ist unter `https://tagung.forth-ev.de` bereits da, zusammen mit näheren Informationen zur erforderlichen Technik und den Abläufen.

Ulrich Hoffmann, Bernd Paysan, Gerald Wodni



[1] engl.: in real live