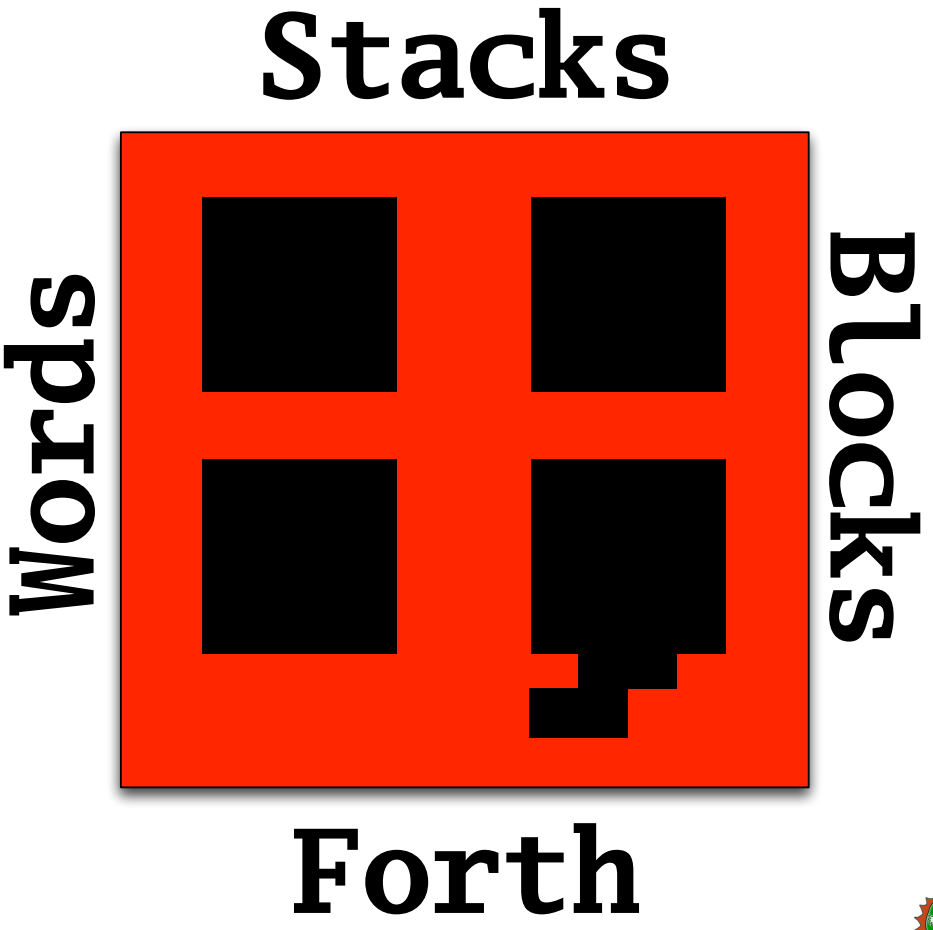
Vierte Dimension

3/2017

Das Forth–Magazin

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



In dieser Ausgabe:

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische
Informatik

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

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

LEGO RCX–Verleih

Seit unserem Gewinn (VD 1/2001 S.30) verfügt unsere Schule über so ausreichend viele RCX–Komponenten, dass ich meine privat eingebrachten Dinge nun Anderen, vorzugsweise Mitgliedern der Forth–Gesellschaft e. V., zur Verfügung stellen kann.

Angeboten wird: Ein komplettes LEGO–RCX–Set, so wie es für ca. 230,–€ im Handel zu erwerben ist.

Inhalt:

1 RCX, 1 Sendeturm, 2 Motoren, 4 Sensoren und ca. 1.000 LEGO–Steine.

Anfragen bitte an
Martin.Bitter@t-online.de

Letztlich enthält das Ganze auch nicht mehr als einen Mikrocontroller der Familie H8/300 von Hitachi, ein paar Treiber und etwas Peripherie. Zudem: dieses Teil ist „narrensicher"!

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

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

Ingenieurbüro
Klaus Kohl–Schöpe

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

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

KIMA Echtzeitsysteme GmbH

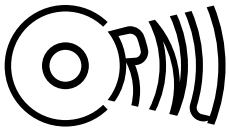
Güstener Straße 72 — 52428 Jülich
Tel.: 02463/9967-0 — Fax: 02463/9967–99
`www.kimaE.de` — info@kimaE.de

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

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

Bergstraße 10 — D–18057 Rostock
Tel.: +49 381 496800-0 — Fax: +49 381 496800-29

PC–basierte Forth–Entwicklungswerkzeuge, comFORTH für Windows und eingebettete und verteilte Systeme. Softwareentwicklung für Windows und Mikrocontroller mit Forth, C/C++, Delphi und Basic. Entwicklung von Gerätetreibern und Kommunikationssoftware für Windows 3.1, Windows95 und WindowsNT. Beratung zu Software–/Systementwurf. Mehr als 15 Jahre Erfahrung.



Cornu GmbH
Ingenieurdienstleistungen
Elektrotechnik

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

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

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

Titelbild Carsten Strotmann hat als Forth–Fan für das *Vintage Computer Festival Europe (VCFe)* ein neues T–Shirt haben wollen. Dazu hat er ein Logo erstellt, welches auf das Logo von Gerald Wodni auf `http://theforth.net` zurückgeht. T–Shirts anyone?

Carsten Strotmann, per email

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

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

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00€ + Porto u. Verpackung

Manuskripte und Rechte

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

Liebe Leser,

wie immer ist es ordentlich Arbeit, die nächste Ausgabe des Heftes auf die Reihe zu bekommen. Und jedesmal beeindruckt mich die Bandbreite der Themen in den Texten. Und wie immer haben viele mitgeholfen — Dankeschön!

Auf der Forthtagung 2017 in Kalkar hat Willem Overkerk provokant gefragt, ob denn Forth überhaupt noch eine Zukunft hat. *Die jungen Leute* seien an solchen Dingen nicht sehr interessiert. Das mag schon sein. Und es hat vermutlich auch damit zu tun, dass diese jungen Leute heutzutage eine unglaubliche Auswahl an Themen und Dingen zur Verfügung haben.

Etwas *machen* wird anscheinend wieder beliebter, wenn man den *Maker Faire* Machern zuhört. Es war noch nie so einfach, einen Mikrokontroller zu allerlei Kunststücken zu überreden — Arduino macht's möglich. Es muss sie also geben, die jungen Leute, die sich für so etwas interessieren. Wie erreichen wir sie? Können wir sie ansprechen oder gar begeistern? Wer die Begeisterung vorlebt, kann etwas erreichen. Daher geht mein Dank stellvertretend an die, die nach Hannover gefahren sind, um die Begeisterung weiter zu tragen.

Mal angenommen, er steht da vor Euch, der neugierige Mensch — der ist ja noch nicht einmal ein Anfänger, kann vielleicht auch gar nicht programmieren. Was macht ihr dann?

Was genau führt ihr vor, wenn ihr versucht, ihr oder ihm Forth schmackhaft zu machen? *Welches Material* habt ihr zur Verfügung? *Was antwortet ihr auf die Frage: warum Forth?* Es ist interaktiv — das hat mich jedenfalls eingefangen. Es ist syntax–arm, das ist doch vorteilhaft, oder? Es ist nur dann anders, wenn man andere Programmiersprachen schon kennt. Ist Vielfalt nicht auch erstrebenswert? Forth bietet offengelegte Innereien: als Feature, nicht als Bug — auch wenn das den Sicherheits–Experten erst mal Adrenalin–Schübe bereitet. Kann das auch ein Vorteil sein? Kann man den *verkaufen*? *Und zeigt ihr sie her, eure Kunstwerke?* Nichts überzeugt mehr, als ein Projekt / Ding / Kunstwerk, welches ihr selbst routinemäßig benutzt.

Ich würde mich sehr freuen, die Antworten der werten Leser zu hören. Diese Information würde ich für die VD aufbereiten, mit der Idee, Inspiration zu verbreiten, gute Ideen zu sammeln, Fundstellen für Material bekannt zu machen, Fehlendes zu benennen — vielleicht weiß ja jemand eine Antwort.

Viel Spaß beim Lesen!

Erich Wälde *the guy who calls himself an Anfänger*

Referenzen:

1. `vd@forth-ev.de` — Email–Adresse der Redaktion
2. `http://wiki.forth-ev.de/doku.php/events:start` — Videoarchiv
3. `http://wiki.forth-ev.de/doku.php/projects:projects` — Projekte

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-2017-03`

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

Ulrich Hoffmann Kontakt: `Direktorium@Forth-ev.de`
Bernd Paysan
Ewald Rieger

wiki.forth-ev.de: Unsichere Verbindung

Ende Juli weigerten sich die Web–Browser tatsächlich, unser Forth–ev–Wiki [1] anzuzeigen. Error code: `SEC_ERROR_EXPIRED_CERTIFICATE`. Unser Wiki ist strikt per https (*HyperText Transfer Protocol Secure*, englisch für „sicheres Hypertext–Übertragungsprotokoll") zu erreichen. Da das Zertifikat der Webseite abgelaufen war, wurde sie nicht angezeigt, die Sicherung funktionierte also! Den Fehler hat Bernd schnell behoben, und die Seite ist wieder da.

Das Forth–ev–Wiki wird neben unserem Blog immer wichtiger für den Verein, liegen doch dort inzwischen alle Projekte, historischen Dokumente und nun auch die „statischen Seiten" aus dem alten Forth–ev–geeklog inclusive der kompletten Ausgaben der „Vierten Dimension", unser Forth–Magazin.

Das Geeklog–System wollen wir ja so allmählich mal loswerden. Es ist zu angreifbar, nur schwer zu warten und auf Smartphone– und Tablet–Displays sehr schlecht darstellbar.

Let's Encrypt wird benutzt, um unsere Webseite zu identifizieren und uns, dem Betreiber, das Zertifikat auszustellen, letztlich eine Datei, anhand der ein Browser erkennen kann, dass wir 'wir' sind. Mit der Zeit ist eine Infrastruktur für die Vergabe von *public keys* (*Public Key Infrastructure*, PKI) entstanden, die es ermöglicht, solche keys automatisch zu erzeugen und zu verteilen (*Automatic Certificate Management Environment*, ACME). Dank der Linux Foundation gibt es den Certbot ACME client, der den Job erledigt.

1. `https://wiki.forth-ev.de/`
2. `https://letsencrypt.org/getting-started/`

mk

theforth.net trifft AmForth

Gerald Wodnis `theforth.net` lebt! Zumindest habe ich eine explizite Verwendung gesehen. Auf dieser Seite findet sich das Paket `lcd-hd44780` [1], welches Gerald auf der Tagung auch als Beispiel für die Benutzung von `theforth.net` vorgeführt hat [2]. Das Paket definiert Worte, die die Ansteuerung eines LCDisplays mit verbreitetem hd44780–Kontroller implementieren. Lediglich 3 Worte (`lcd-nibble`, `lcd-mode-data` und `lcd-mode-cmd` müssen vor der Benutzung des Pakets definiert werden.

Genau das hat Matthias Trute für AmForth auf dem msp430–Kontroller getan und vorbildlich für uns Leser aufgeschrieben (AmForth Cookbook). Als Belohnung bekommt er einen Swap–Drachen auf dem LCDisplay angezeigt — siehe Beweisfoto bei [3]!

1. `http://theforth.net/package/lcd-hd44780`
2. `http://wiki.forth-ev.de/doku.php/events:tagung-2017:theforth.net`
3. `http://amforth.sourceforge.net/TG/recipes/LCD-HD44780.html#lcd-hd44780`

Erich Wälde

emacs trifft Forth im Terminal

Neulich auf der Maker Faire:

Klaus: Ist das ein Terminal Programm?
Carsten: Nö, das ist `emacs`.
Klaus: `emacs`? Nie gehört...

Um diesem Missstand abzuhelfen hat Carsten Strotmann netterweise eine Seite im Forth Wiki angelegt. Dort wird beschrieben, wie man sich mit `emacs` anfreunden kann — mit Forth haben wir das ja auch geschafft. Als `emacs`–Benutzer seit 1992 kann ich das nur empfehlen (Werbung mach).

1. `http://wiki.forth-ev.de/doku.php/projects:emacsandforth:emacs_und_forth`

Erich Wälde

Neues aus dem Mikrokontroller–Verleih

Im Mikrokontroller–Verleih der Forth–Gesellschaft stehen seit Ende August drei neue Boards zur Verfügung.

Den Anfang macht das Novix 4000 EB1 Board. Dieses Board von "Forth–Systeme–Angelika–Flesch" ist eine Euro–Platine mit der Novix–4000P–CPU von Chuck Moore. Das Board wird per cmForth programmiert. Wer schon immer mal einen der ersten Forth-Prozessoren ausprobieren wollte, hat hier die Gelegenheit. `https://wiki.forth-ev.de/doku.php/mcv:forthcpu#novix_4000_eb1_board`

Das zweite Board ist das TDS–2020F mit Hitachi–H8/532 von Triangle Digital Services aus England. Das Board kommt mit Flash–Forth auf dem Board, umfangreicher Dokumentation (3 Bücher mit zusammen über 1000 Seiten) sowie einer Entwicklungsumgebung für Windows–Systeme auf CD–ROM. `https://wiki.forth-ev.de/doku.php/mcv:hitachiboards`

Das dritte Board ist ein Zilog–Super8–Entwicklungsboard von Heinz Schnitter und Klaus Kohl. Die Super8 CPU hat ein Forth im Masken ROM, die Befehle ENTER, EXIT und NEXT sind direkt in der CPU als Maschinensprache–Instruktionen implementiert. Diese CPU ist speziell für die Ausführung von Forth optimiert. Die Super8–CPU ist in der VD 3/91 beschrieben: `http://forth-ev.de/filemgmt_data/files/4d1991_3.pdf`

Als Entwicklungsumgebung gibt es das Volksforth der Forth–Gesellschaft. Weitere Informationen zu diesem Board auf den Webseiten des Mikrokontroller–Verleihs: `https://wiki.forth-ev.de/doku.php/mcv:forthsupport#zilog_super8`

Alle drei Boards wurden von Norbert Lott gespendet, vielen Dank dafür.

Carsten Strotmann

Nächste Jahrestagung: 5. – 8. April 2018

Es ist soweit, der Ort ist gebucht, die Preise sind berechnet. Wir treffen uns im kommenden April im Linuxhotel in Essen.

Das Hotel bietet mehr als nur Betten und Frühstück: Ruhe im riesigen, eingezäunten Privatpark direkt oberhalb des heute sehr sauberen Flusses „Ruhr“, eine wahre Oase (ich war schon da: stimmt! mk); konzentrierte Umgebung, in der eine homogene Gruppe mit Spezialisten lernen kann; zwangloser Erfahrungsaustausch in Park und im Kaminzimmer; Tagungsort für Gruppen, die eine sehr gute, kostenlose IT-Umgebung benötigen.

Voraussichtliche Preise:

Tagung EZ /Person:	199 €
Tagung DZ /Person:	189 €
Begleitperson:	−15 €
Anreise Donnerstag Nachmittag:	+25 €
Frühanmeldung bis 1.3.2018:	−20 €

Also schon mal vormerken! Eine online–Anmeldeseite kommt auch bald.

`http://tagung.forth-ev.de`

`https://linuxhotel.de`

cs,mk

Social Media ohne Chuck Moore

Von Carsten, von Lars, von Elizabeth:

> On 9/2/17 3:18 PM, Liang Ng wrote:
>
> > Whereabouts of Charles H. Moore?
> >
> > Along with Satoshi Nakamoto, Charles H. Moore has become the most mysterious programmer on my list. I have not been able to find recent web results regarding Moore.
> >
> > Can anyone shed some light?
>
> He has said that he is retiring from active participation in Forth–related activities. We wish his well in his retirement.
>
> Cheers,
> Elizabeth

`https://groups.google.com/forum/#!original/comp.lang.forth/LmthCiZ4Cjk/f-e03GHXAQAJ)`

Im weiteren Verlauf der Diskussion:

> 2017-09-17
> Elizabeth D. Rather
>
> As luck would have it, Greg just phoned me, because he says Greenarrays has been getting messages (presumably from some of you) about Chuck's status. Greg assures me that Chuck is not retired, he is still doing work for Greenarrays, however he has withdrawn his online presence because it was taking too much of his time. Greg says he is planning to be at Forth Day this year, and looks forward to seeing some of you there.

`https://groups.google.com/forum/message/raw?msg=comp.lang.forth/LmthCiZ4Cjk/siGtIXOSBAAJ`

Michael hat keine Mühen gescheut, Chuck selbst zu hören:

> 2017-09-23
> Chuck Moore
>
> I have retired from social media. I have not retired from Forth.

Private Mitteilung via Jürgen Pintaske.

mk, Erich Wälde

Perlen im Web

Im Wilden Weiten Webernetz finden sich ab und zu auf verschlungenen Pfaden richtige Perlen. Zwar ist die Bewertung, ob Perle oder nicht, durchaus individuell verschieden, aber ich habe den Artikel von Graydon Hoare (englisch) mit wachsender Begeisterung gelesen. Und gegen Ende findet auch Forth Erwähnung — zwar kurz, aber in einem bemerkenswerten Zusammenhang.

`http://graydon2.dreamwidth.org/253769.html`

Erich Wälde

Types of Error

Frisch gepflückt für Euch aus *Neil Bawd's Ugly Home Page* von 2004 die Liste typischer Fallen für Programfehler. Viel Vergnügen.

- A — an algorithm awry.
- B — a blunder or botch.
- C — a cleanup for consistency or clarity.
- D — a data structure debacle.
- E — an efficiency enhancement.
- F — a forgotten function.
- G — a generalization or growth of ability.
- I — an interactive improvement.
- L — a language liability.
- M — a mismatch between modules.
- P — a promotion of portability.
- Q — a quest for quality.
- R — a reinforcement for robustness.
- S — a surprising scenario.
- T — a trivial typo.

Referenzen

See Donald E. Knuth, *Literate Programming*, 1992, ISBN 0-937073-80-6, pp. 245-247.

`http://www.wilbaden.com/neil_bawd/typerr.txt`

mk

Fortsetzung auf Seite 11

Erasmus bringt Forth nach Coimbra

Jens Storjohann

Ein kleiner Reisebericht aus Europa.

Erasmus von Rotterdam verdient unser Lob und unseren Dank, weil er uns gezeigt hat, dass man mit der Wissenschaft im Gepäck in ganz Europa willkommen ist (Das Erasmus–Programm der EU ist zu seinen Ehren nach ihm benannt).

Geboren ist er vermutlich am 28. Oktober 1466, 1467 oder 1469, wahrscheinlich in Rotterdam und gestorben am 11./12. Juli 1536 in Basel.

Er studierte und forschte in vielen europäischen Ländern und schrieb wohl 150 Bücher. Sein bekanntestes Werk ist eine Satire und wird "Laus stultitiae" (Lob der Torheit) genannt. Zu seiner Zeit konnte man sich in gelehrten Kreisen auf Latein verständigen. Heute dürfen oder müssen wir Englisch sprechen und schreiben.

Weil ein elektrotechnischer Lehrstuhl meiner Uni, die von ihren Angehörigen gerne als beste von Hamburg–Jenfeld bezeichnet wird, Kooperationen im Rahmen des Erasmus–Programms mit der Universität von Coimbra (Portugal) fortsetzen wollte, wurde ich mit einem Kollegen auf die Reise nach Coimbra geschickt. Dort sollten Projekte besprochen werden.

Das Erasmus–Programm verlangt, um Besuche zu finanzieren, Wissensaustausch in Form von Vorträgen oder Vorlesungen. Ich hatte vier Themen zur Wahl angeboten: Drei „normale elektrotechnische" und einen Vortrag über Forth mit dem anreißerischen Titel *"The Programming Language Forth — The most exotic computer language in our solar system"*.

Unsere portugiesischen Freunde dachten wohl mit Recht "normal können wir selbst" oder hatten sich das oben genannte Werk von Erasmus zu Herzen genommen und wählten den Vortrag über Forth.

Ich erklärte kurz und zum Teil beispielhaft den Stapel, Arithmetik, direktes Ausführen bekannter Worte, Definition neuer Worte, Ablaufsteuerung, die in UPN ungewohnt aussieht, Definition neuer Definitionsworte.

Meine Zuhörer folgten meinen Argumenten und waren, weil sie zu einem großen Teil mit Hardware–Entwicklung befasst waren, von der Kompaktheit der Sprache und dem im Titel schon angedeuteten Einsatz durch Spezial-Prozessoren im Weltraum positiv beeindruckt.

Ich versuchte die Sprache Forth als ein System darzustellen, das auf viele Dinge, die man heute in der Informatik für selbstverständlich hält, wie z. B. strong typing, bewusst verzichtet. Andere Dinge wie z. B. die Erweiterbarkeit des Compilers und das inkrementelle Übersetzen sind der Schlüssel zum Erfolg von Forth.

Bei kurzen Vorführungen vertippte ich mich dann einmal und konnte gleich zeigen, dass dies wegen des inkrementellen Übersetzens ganz leicht und schnell repariert werden kann. In jedem Fall ist das Vorführen eines lebendigen Forth–Systems sehr nützlich, und das Vertippen lässt sich vielleicht sogar absichtlich in einen Vortrag einbauen.

Bei den Regeln, die Chuck Moore aufgestellt hat:

- keep it simple
- do not solve more general problems than you are paid for
- you do not need strong typing and syntax checks
- do it yourself (do not use subroutine packages)

mochten meine Zuhörer mir insbesondere in Bezug auf *"do not use subroutine packages"* nicht folgen. Ich selbst denke auch, dass die Regeln z. T. bedingt sind durch die Situation eines freiberuflichen Programmierers, die anders ist als die eines Mitglieds einer Forschungsgruppe, die über Jahre ein Paket von speziellen Routinen für ein Anwendungsgebiet aufbauen kann. Vielleicht hat nun Forth neue Anwender gefunden. In jedem Fall ist Forth auf Interesse gestoßen.

Die Traditionsuniversität von Coimbra ist allein wegen der schönen alten Gebäude eine Reise wert. Wer Harry–Potter–Filme gesehen hat, erkennt, dass die Universität von Coimbra als Vorbild gewirkt hat. Insbesondere die Studentinnen und Studenten, die bei traditionellen Veranstaltungen in schwarzen Samtumhängen auftreten, erinnern Besucher an Hogwarts.

Sonst haben wir die Freundlichkeit unserer Gastgeber, die Atmosphäre der Stadt und das gute Essen genossen. Auch ohne Forth ist Coimbra einen Besuch wert.

Bild: François Philipp `https://commons.wikimedia.org/wiki/File:Universidade_de_Coimbra_(10249088113).jpg`

Maker Faire Hannover

Klaus Zobawa

Die Forth–Gesellschaft e.V. ist nicht zum ersten Mal mit Leuten und Material auf der Maker Faire Hannover *aufgekreuzt.* Matthias Koch *hat keine Mühen gescheut, das Algenaquarium erneut heranzuschaffen, weil ein Blickfang gehört definitiv zum Handwerk.* Klaus Zobawa *berichtet von seinen Eindrücken.*

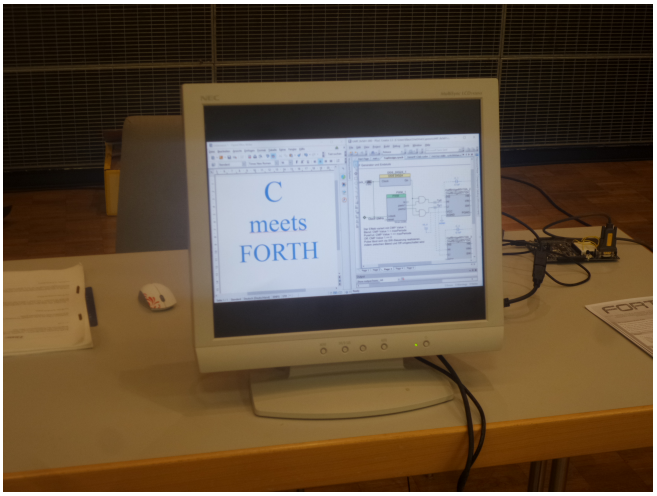
1. Tag

Als ich Samstag Morgen kurz vor zehn mit Matthias Koch, bei dem ich dankender Weise nächtigen konnte, an unserem Stand eintraf, ging es auch schon sofort los. Kurz nach dem ich den Laptop hochgefahren, das Cypress Eva–Board angeschlossen und einen Monitor für die Messebesucher aufgestellt hatte, trafen auch schon die ersten Interessenten ein — für's Algen-Aquarium.

Was ist das? Wozu braucht man das? Handelt es sich um eine alkoholische Gärung? Köchelt ihr sinneserweiternde Substanzen? Dabei die Kurve zu Forth zu kriegen war nicht immer einfach und in 99% der Fälle auch nicht weiter von Interesse. Das Algen–Aquarium zieht die Besucher zwar magisch an, aber sobald es ums Programmieren geht — Fehlanzeige. Gut, dass Matthias da kind– und/oder erwachsenengerecht sein Fachwissen vermittelte. Um mögliche Forth–Kandidaten aus dem Besucherstrom zu filtern, habe ich auf dem besagten Monitor das Wortspiel:

```
C ++
FORTH
```

zur Anzeige gebracht. (man beachte das Leerzeichen nach dem C `;)` Und tatsächlich kamen am Samstag ca. 10–15 durchweg ältere Herren an den Stand — davon gingen 2 kopfschüttelnd und lachend vorbei. Als dies zum dritten mal passierte, wollte ich doch erfahren, was die Häme bewirkt. Antwort: „Wie kann man sich das antun? Welcher Chef lässt das zu?“ Meine Frage nach Forth–Kenntnissen wurde durchaus bejaht mit dem Hinweis „ja, gerade deswegen“. Kurzerhand, ohne Umweg über das Eva–Board, wies ich auf die Einfachheit von Forth hin, die es erlaubt, den Compiler, den FORTH als Sprachelement beinhaltet, zu erweitern, um z.B. Entscheidungstabellen in graphischer Form übersetzen zu lassen und damit das V–Model mit geringerem Aufwand zu durchlaufen. Danach fing er an, etwas zugänglicher zu werden, bekehren konnte ich ihn jedoch nicht.



Angenehmer waren da schon die anderen, die quasi unisono erstaunt fragten: „Oh, Forth gibt es noch?! Ich hatte vor 30 Jahren: davon gehört, davon gelesen, damit experimentiert (auf dem Atari), beruflich damit gearbeitet und last but not least einen Forth-Interpreter geschrieben.“ Bei denjenigen, bei denen ich ein echtes Interesse erkennen konnte, erläuterte ich die Vorzüge von Forth am Beispiel des Eva–Boards. Zunächst stellte ich die Hardware des PSoC 5LP von Cypress vor und die Möglichkeit, Hardwareblöcke — nicht nur digital, sondern auch analog — als Schaltplan erstellen zu können, und damit den Chip zu konfigurieren. Als Nächstes rückte der integrierte Controller Cortex M3 in den Mittelpunkt, und die Möglichkeit, C–Applikationen zu erstellen, für die eine umfangreiche C–Bibliothek zu den einzelnen Hardwarekomponenten im PSoC–Creator mitgeliefert wird. Über

die serielle Schnittstelle zeigte ich, wie in der Main-Loop durch Abfrage der Tasten 's' und 'c', die Blinkfrequenz einer LED zwischen 30 und 3 Herz gewechselt werden kann. Bisher durchaus vertrautes Terrain für die Gäste.

Mit dem Hinweis, man könne ja in C eine Tasteneingabe realisieren, um beliebige Frequenzen zw. 0 und 1000000 Hz einstellen zu können, beendete ich durch das Betätigen der Taste `f` die Funktion `main()` und startete den mecrisp–Interpreter. Hier ließen sich am Beispiel der Frequenzvorgabe die Vorzüge des Forth–Interpreters voll ausspielen. Er erlaubt, ohne irgendeine Zeile programmieren zu müssen, die Einstellung der Frequenz durch Aufruf der C–Funktion aus Forth heraus, was durchaus in erstaunte bzw. überraschte Gesichter blicken ließ. Die Compiler–Erweiterung am Beispiel von Entscheidungstabellen war dem einen oder anderen dann doch zu schnell vorgetragen und eher suspekt. Zwei Dinge sind mir an diesem Tag klar geworden: Forth ist eine Sprache ohne Syntax (quasi) und das größte Problem beim Erlernen von Forth besteht darin, alles, was man bisher über Programmiersprachen gelernt hat, zu vergessen.

2. Tag

Ich ging davon aus, dass sich die Klientel am 2. Tag nahtlos fortsetzen würde — weit gefehlt. Der zweite Tag lief völlig anders. Es gab nach wie vor den einen oder anderen älteren Herrn der „Was? Forth gibt es noch?" zum Besten gab, aber überwiegend kamen jüngere „Semester", die Informatik oder Elektrotechnik studierten, auf der Suche nach einem Einstieg in die Programmiertechnik waren oder selbst Maker sind und händeringend nach Computerlösungen suchten. Die meisten zeigten sich positiv überrascht von den Möglichkeiten, die Forth bietet, und nahmen unseren Handzettel mit dem umfangreichen Mecrisp–Portfolio dankend entgegen und wollten das auf jeden Fall ausprobieren. Die Flyer gingen über den Tag verteilt auf diese Weise weg wie warme Semmeln.

Gegen Nachmittag flaute der Besucherstrom merklich ab, was gut war für meine Stimme, die sich mit der Zeit wieder erholen konnte. Von einer Begegnung möchte ich noch etwas ausführlicher berichten. Fast unbemerkt stand ein Herr am Tisch und studierte unseren Flyer. Mir kam des Gesicht bekannt vor und ich dachte, er war letztes Jahr schon zu Besuch. Ich sprach ihn an und fragte, ob er Forth kenne. „Ein Kollege von ihm habe in Forth entwickelt" und „sie hätten den RTX 2000 eingesetzt." Ah, dachte ich, er kennt den RTX — und hat ihn was? eingesetzt ??? — Genau! Iss klar! Ich hab auch ein engineering sample zu Hause. Das war eine Steilvorlage, um auf die 8 RTX'se im Landemodul auf den Kometen Tschuri hinzuweisen. Ab da wurde es spannend. Er „habe den Datensatz vom Gas–Spektrometer heruntergeladen." „Also aus dem Internet?", fragte ich. „Nein, vom Spektrometer, er hat die Hardware entwickelt." „WHAT? Herzlichen Glückwunsch!" Wir unterhielten uns noch über die Energieversorgung von Philae und darüber, warum Solarpanels verwendet wurden: „es war ein EU Projekt `;-)`"

Die Antwort auf meine Frage, woher ich sein Gesicht kannte, ist einfach — daher: `https://www.youtube.com/watch?v=GvDrw8AdGPM` ab Minute 14:00

Abbildung 1: Carsten (mit dem frisch gedruckten T–Shirt), Gido, Swap und die Bit–Kanone

Das RAM–ROM–Dilemma von interaktivem Forth in kleinen MCUs

Michael Kalus

Code, der von einem laufenden Forth ins RAM kompiliert wurde, ist nach einem Reset verloren. Ob das ein Problem ist, hängt davon ab, ob man den Code dann vermisst, oder ob man froh ist, dass er endlich weg ist. Code, der von einem laufenden Forth ins Flash kompiliert wurde, ist nach einem Reset noch vorhanden. Aber wenn es versäumt worden ist, den System–Zeiger auf die nächste freie Stelle des Flash VOR dem Reset ins Flash zu sichern, compiliert Forth nach dem Booten beim interaktiven Weiterarbeiten an die ihm zuletzt bekannte Stelle - und überschreibt was dort noch steht. Mit der Folge von falsch gesetzten Bits. Was dazu führt, dass Forth spätestens dann, wenn solcher Code ausgeführt wird, abstürzt - die Reset–Falle. Man muss Code, der im Flash im Weg ist, also aktiv loswerden. Der Artikel beleuchtet unterschiedliche Ansätze für 4e4th [3], noForth [4] *und* eForth [2] *für dieses Vorgehen.*

Forth wurde von Charles Moore für Computer erfunden, die ein durchgehendes Random Accsess (RAM) oder Read Only Memory (ROM) hatten. Man konnte Programme von der Diskette ins RAM laden, oder gleich vom ROM ausführen. Interaktiv erstellter Code wurde im RAM compiliert. Doch wohin damit, wenn der Strom ausgeschaltet wurde? Es musste auf irgend einen Datenträger kopiert werden. Und bei Stromunterbrechungen war alles futsch. Das war nicht immer ein lästiger Fehler, sondern auch eine Möglichkeit, den Murks den man testhalber programmiert hatte, wieder zu löschen. Flash–Memory, also dieser Speicher mit der seltsamen Eigenschaft zur Laufzeit wie ein ROM zu sein, aber dennoch beschreibbar — fast wie ein RAM zur Compilezeit — das war da noch nicht erfunden. Flash ist aber eben nur fast wie ein RAM, weil darin ja, je nach Typ des Flash, die Bytes nicht beliebig überschrieben werden können. Denn Überschreiben geht dort bekanntlich nur, indem ein vorhandenes gesetztes Bit, eine “1”, zur “0” gemacht wird, also zurückgesetzt. Umgekehrt geht das nicht. Bits wieder zu “1” zu machen geht im Flash nur sektoren– oder blockweise (*erase blocks*).

So kommt es, dass Forthsysteme für die verschiedenen Flash–Speichertypen in heutigen kleinen MCUs jeweils eine besondere Vorgehensweise für das Schreiben haben müssen. Damit es so aussieht, als laufe aller Code wie aus einem ROM, kompiliere wie im RAM, aber speichere schließlich doch wieder wie in einem ROM. Und das braucht für jede MCU einen spezifischen Treiber. Dem Anwender ist das verborgen. Fast jedenfalls. Auf den sehr kleinen MCUs kommt man nicht drumherum, sich darum Gedanken zu machen, denn der winzige Speicherplatz, 16 KB Flash oder so, darf nicht schon von zuviel Forthsystem aufgebraucht werden, sonst bleibt ja nichts übrig für eine Applikation. Das Forthsystem muss also sehr schlank gehalten werden, da ist für syntaktischen Zuckerguss kein Platz.

Tings eForth für den MSP430 ist so ein Forth ohne jeden Zuckerguss. Es läuft im ROM (Flash). Aber es kompiliert eben auch direkt ins Flash.

Im RAM interakiv zu arbeiten, war einfach: Man schützte den Forthkern (kernel) vor einem FORGET, aber alles was man ausprobiert, konnte mit FORGET sehr einfach wieder weg gemacht werden. Ein Neustart mit COLD oder der RESET leerte das RAM und man fing mit frischem Forthkern und sauberem RAM einfach nochmal an.

Auf einer MCU mit Flash–Memory geht das so NICHT! Alles, was ins Flash compiliert wird, auch das, was man nur mal so ausprobieren will, geht nicht einfach so wieder weg. Man verhunzt das Flash und kann das eben NICHT byteweise wieder bereinigen. Das geht immer nur in ganzen Segmenten.

Wie kann man dann interaktiv immer wieder mit einem frischen Forthkernel und sauberem Flash neu anfangen? Denn das ist es ja, was man als Student seiner MCU und des Forth dauernd braucht. Da gibt es nur zwei Wege:

- Den ganzen Chip neu flashen. Das ist sehr lästig, geht beispielsweise beim MSP430 nur mit einem Flasher–Tool wie dem LaunchPad und einer Software dafür — unter Windows etwa dem Lite–FET–Pro430 von Elprotronic. Und es geht dann eben gar nicht mehr, ist der Chip erst einmal irgendwo verbaut.
- Eine Art von `FORGET` bereitstellen, also Code, der alles, was nach dem Forthkern noch im Flash steht, wieder löschen kann. Und das so, dass auch die freien Reste von schon angefangenen Segmenten hinterher wieder sauber sind und erneut beschrieben werden können. Der Code muss also auch ein angefangenes Segment irgendwohin retten, es dann löschen, und nur den gültigen Teil zurückschreiben.

Doch genau SO ETWAS kann man erst selbst machen, wenn man seine MCU und auch sein Forth schon sehr genau kennt. Man muss also etwas machen, was man als Student der MCU und seines Forth überhaupt noch nicht versteht.

Dieses Dilemma war der Grund dafür, das 4e4th für den MSP430 zu machen. Es benimmt sich wie Forth “damals” und hat `WIPE` als eine Art von `FORGET`. Etwas später kam dann das noForth und hat dieses Problem eleganter und standardkonformer gelöst durch `SHIELD` und `MARKER`.

Das Problem besteht also NICHT darin, den gelungenen fertigen Code einer Applikation zu sichern. Einmal kompiliert steht dieser Code ja schon im Flash, ist also auch nach dem Reset noch da. Und so ist Ting im eForth die

Sache auch angegangen, es wird immer alles aufgehoben. Er ist den ROM–Weg gegangen. Und sein eForth hat konsequenterweise KEINE eingebaute Möglichkeit, so eine Applikation nochmal aus dem ROM zu werfen und anders aufzusetzen. Aber damit sitzt man beim eForth eben auch in der Reset-Falle: Der Code bleibt auch ohne Strom — es hilft nur noch eins: Den ganzen Chip neu flashen. Was nicht möglich ist, wenn der unzugänglich (also meistens) verbaut wurde.

Im 4e4th dagegen sind wir den RAM–Weg gegangen. Jeder hinzuompilierte Code wird beim Booten automatisch verworfen, es sei denn, das wird ausdrücklich unterbunden! Das NoForth macht es auch so. Im 4e4th gibt es ein `SAVE` dafür und in noForth das `FREEZE` . Damit kann man so tun, als stünde alles in einem RAM. Und wenn sonst nichts mehr hilft, dreht man seiner MCU den Saft ab, bootet und die Applikation ist weg. Der Chip ist also sofort wieder einsetzbar, aller Unsinn ist getilgt, reflashen nicht nötig. Der Nachteil ist, dass diese automatischen Löschungs–Mechanismen speicherfressend sind, ein unnötiger Ballast, den Tings eForth vermeidet.

Daher hat `Manfred Mahlow` nun `SHIELD` und `MARKER` für eForth gemacht [1], sehr kompakt, gut gelöst. Damit wird eForth wieder richtig gut brauchbar.

Doch soll solcher Code dann auch wieder in den Forthkern gepackt werden? Ich finde nein! Man kann und sollte solche Forthsysteme wie eForth schön klein lassen, so klein wie überhaupt möglich. Trotzdem müssen `SHIELD` und `MARKER` unbedingt mitgeliefert werden, denn das kann ein Student noch nicht selbst! Doch Code aus einer Toolbox für die Entwicklungsphase laden, das geht. Und auch verstehen, warum man das so macht, geht. So etwas wie eForth darf also nicht ohne toolbox daher kommen, sonst tut man der Forth–Sache keinen Gefallen.

Im 4e4th haben wir, `Dirk Brühl` und ich, den Fehler gemacht, solche Tools schon in den Forthkern zu packen, fertig assembliert und nur noch beherrschbar über diese schreckliche IAR Embedded Workbench unter Windows XP. Der Forthkern wurde damit immer aufgeblähter und war schließlich nicht mehr zu handhaben. Kein Anwender konnte selbst Einfluss nehmen, und selbst wir als Entwickler konnten es nicht heile über die IAR-Updates retten und nach Windows 10 bringen. Das kann man einfach nicht leisten. Das war ein ganz großer Fehler, diese proprietäre Software für die Entwicklung zu nehmen! Und im noForth schleppt man die tools auch schon im Kernel mit herum.

Ting hat das also klar besser gemacht als wir, er hat den Forthkern wirklich bis auf das absolute Minimum reduziert, und zu einer verlässlichen Basis gemacht. Und gleichzeitig hat er damit auch alles, was man während der Entwicklungszeit zusätzlich noch so brauchen könnte, klar in nachladbaren Code verbannt. DAS ist der richtige Weg! Doch damit eForth gut brauchbar ist, MUSS so eine Toolbox mitgeliefert werden, sonst verzweifelt der Student und schmeißt Forth schnell in die Ecke.

Referenzen

1. Mahlow, eForth: `https://wiki.forth-ev.de/doku.php/projects:430eforth:start`
2. Einen Link zum offiziellen eForth gibt es derzeit leider nicht, Ting hat keine Webseite bereitgestellt. Aber ihr könnt Kontakt zu ihm aufnehmen und euch vorab im Wiki der Forth–Gesellschaft darüber Infos holen: `https://wiki.forth-ev.de/doku.php/projects:ting_s_electronic_forth_bookshelf`
3. 4e4th: `http://www.4e4th.eu/`
4. noForth: `http://home.hccnet.nl/anij/nof/noforth.html`

Fortsetzung von Seite 6

Another Tutorial for writing a Forth Interpreter in Assembly

Der Autor zeigt, wie man ein Forth schreiben kann. Das System wird in Python simuliert und in Assembler geschrieben.

`https://blog.asrpo.com/forth_tutorial_part_1`
`https://blog.asrpo.com/forth_tutorial_part_2`

Aus dem Netz gefischt von C. Strotmann

Fortsetzung auf Seite 19

Clock Works 3 — Auf der Suche nach der verlorenen Zeit . . .

Erich Wälde

Im ersten Teil dieser Reihe [1] ging es um Programm–Bausteine zur Realisierung einer Uhr. Dabei war ich schon über das Phänomen gestolpert, dass die Uhr mehrere Sekunden pro Tag in Verzug geriet. Ich stellte ein Verfahren vor, um das auszugleichen, aber es ließ mir natürlich keine Ruhe. Es passte nicht zu meiner Vorstellung von Quarz–Frequenzen und ihrer Genauigkeit. Also beschloss ich irgendwann, das Thema neu zu beleuchten.

Zeit verlieren

Im Datenblatt von Uhrenquarzen steht unter anderem, dass die Frequenzen auf 20 ppm[1] oder auch 5 ppm genau seien, vorausgesetzt man betreibt sie unter den vorgesehenen Bedingungen. Von Lastkapazität steht da auch noch was, die sei z.B. 12.5 pF. Und mit geeigneter Wahl der Last–Kapazität könne man noch deutlich genauer werden. Soweit so gut.

Jetzt sind 20 ppm nicht gerade viel. So ein Tag hat 86400 Sekunden und 20 ppm davon sind dann eben 1.7 Sekunden. Zwei Sekunden Abweichung pro Tag wären zwar immer noch eine Menge, aber schon viel weniger als die beobachteten ca. 20 Sekunden pro Tag. Irgendwas war hier ganz klar unverstanden. Ordentlich durchgeführte Messreihen mussten her.

Vorüberlegungen

Nun kann man zwar wild drauf los messen und wenig aufschreiben, aber das führt selten zum Ziel. Ein paar (auch abwegige) Ideen vorher zu sammeln, ist kein Luxus. Zumindest wollte ich ja sehen, ob Veränderungen am Programm die Abweichung signifikant verändern würden, oder nicht.

1 Zunächst wollte ich wissen, ob ich bisher einem ganz individuell angeschlagenen Kontroller aufgesessen war. Dazu eignet sich ja schon die Messung auf vier baugleichen, aber physikalisch verschiedenen Platinen. Alle benehmen sich gleich? Dann ist die Hardware eher gleich gut (oder schlecht).

2 Mit vier Platinen lässt sich auch die Frage besser klären, ob ich individuell schlechten (sprich billigen) Uhrenquarzen aufgesessen bin. Ich hatte zumindest noch ein paar davon in der Schublade, auch einen SMD–Quarz ganz anderer Bauart. Alle *Uhren* benehmen sich gleich? Dann sind die Quarze auch eher nicht die Übeltäter.

Bleiben weitere Ideen zum Thema *das Forth–Programm ist subtil fehlerhaft*. Wir erinnern uns: der Uhrenquarz treibt Timer/Counter2. Dieser produziert nach N Zyklen einen Überlauf, was einen Interrupt auslöst. Das sind die Ticks, die beispielsweise 128 mal pro Sekunde auftreten.

[1] parts per million

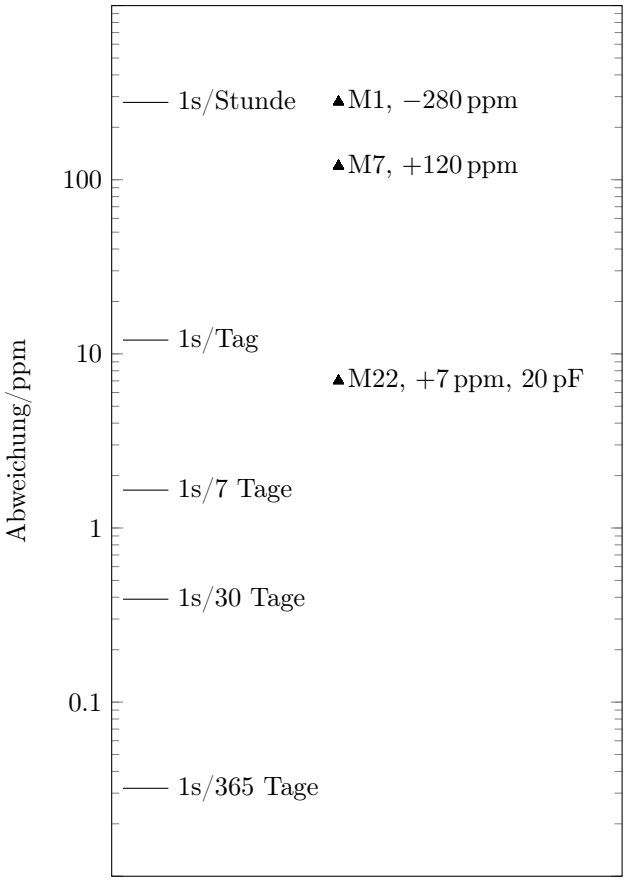


Abbildung 1: Abweichung der Uhr von der exakten Zeit

Die zugehörige Interrupt–Service–Routine macht diverse Buchhaltungsaufgaben und das Hauptprogramm reagiert entsprechend darauf.

3 Der Buchhaltung der Zähler genauer auf die Finger schauen, wäre kein Schaden. Denn die Uhren driften ja, soweit ich das beurteilen konnte, eher gleichmäßig und nicht nur manchmal oder ruckartig.

4 Eine erste Schwachstelle habe ich tatsächlich in der Buchhaltung gefunden. `i` ist die *Ebene im Räderwerk*, entspricht also Sekunden, Minuten, etc.

```
count[i] @  limit[i] @ > if
  0 count[i] !    \ Zweifelhaft!
  1 count[i+1] +!
  flag[i+1] f.set
then
```

Den Zähler `count[i]` auf 0 zurückzusetzen kann schlecht sein, wenn `count` das `limit` nicht nur erreicht, sondern schon überschritten hat. Dann gehen da Inkremente verloren. Besser ist:

```
count[i] @ limit[i] @ - count[i] !
```

Die erste Version verlässt sich drauf, dass alle Tick–Ereignisse immer komplett bedient werden, bevor der nächste Tick auftritt. Das könnte falsch sein.

5 Macht es einen Unterschied, ob ich 128, 64, 32 oder noch weniger Ticks pro Sekunde einstelle? Die Ticks dienen dazu, Dinge zu organisieren, die mehrfach pro Sekunde stattfinden müssen (z.B. die Abtastung des DCF77–Funksignals). Ohne solche Aufgaben wären die Ticks völlig überflüssig.

6 Ändert sich das Verhalten, wenn man den Uhren–Tick alternativ aus dem Hauptquarz und Timer/Counter1 gewinnt?

7 Hilft es, einen richtig guten Uhrenquarz, oder einen TCXO (*temperature compensated crystal oscillator*) als Quelle für die Ticks zu verwenden?

8 Obligatorische Ketzerfrage: C statt Forth?

9 Ist die ISR zu lang? Werden Interrupts *verschluckt* und gar nicht falsch gerechnet?

10 Hilft es, einen anderen atmega Kontroller, z.B. einen atmega-32 (das ist ein älteres Design als der 644p), zu verwenden?

11 Oder einen ganz anderen Controller, z.B. einen msp430? Ich kann mir zwar echt nicht vorstellen, dass die ganze AVR–Familie fehlerhaft sein soll, aber weiß man's.

12 Man kann den Überlauf–Takt von Timer/Counter2 auf einen zugehörigen Pin legen und mit dem logic analyzer nachsehen, ob es da Aussetzer gibt.

13 Die FUSE–Einstellungen sind auch immer mal für Überraschungen gut. Die habe ich nachkontrolliert — die erwiesen sich nicht als Ursache.

Laboraufbau

Als Ausrüstung für mein *Zeitlabor* suchte ich 4 gleiche Platinen zusammen. Von allen wurde die unbenutzte Peripherie weitgehend mit dem Lötkolben entfernt. Auf allen Platinen entfernte ich die Spannungsregler, ich wollte die Platinen direkt mit 5 V vom Labornetzteil betreiben — man weiß ja nie. Auf alle Platinen kamen baugleiche atmega–644p–Mikrokontroller, und auf alle Kontroller zuerst mal das gleiche Programm vom letzten Teil. Das Programm startet mit der willkürlich gewählten Zeit `2000-01-01 00:00:00`, die in der Funktion `init` hart einprogrammiert wurde. Als Vergleich diente ein alt–bewährter Funkwecker, der beim Vergleich mit der ntp–synchronisierten Uhr vom Rechner nie Zweifel aufkommen ließ. Eine Messreihe sollte immer mindestens einen Tag laufen, so der Plan.

1 Initiale Konfiguration

Alle Kontroller wurden im ersten Versuch mit dem gleichen Programm ausgestattet. Ich hatte in den Versuchen vom ersten Teil schon den Eindruck gewonnen, dass weniger Ticks pro Sekunde etwas vorteilhafter waren, daher habe ich 32 Ticks/s gewählt anstelle der 128 im ersten Teil.

- Der Takt des Uhrenquarzes (32768/s) treibt Timer/-Counter2
- Timer/Counter2 wird mit einem Vorteiler von 8 betrieben, zählt auf 128 und macht dort einen *clear on compare match*, d.h. der Zähler wird auf 0 gesetzt ($32768/(8 \cdot 128) = 32/s$). Das sind die Ticks.
- Die ISR addiert 1 zur Zählervariable `ct.timer2`, setzt das Tick–Flag, addiert `ct.fcontrol` zu `ct.phase` und stellt fest, ob es bei dieser Addition einen Überlauf gegeben hat. Wenn ja, wird das Sekunden–Flag gesetzt.
- Die Hauptschleife reagiert auf die gesetzten Flags, macht die Buchhaltung der restlichen Zähler (Minute . . . Jahr) und was es sonst noch zu tun gibt.

Bei einem Tick von 1/32 Sekunde und einem Quarz mit der Frequenz 11.059200 MHz stehen in einem Tick $11095200/32 = 345600$ CPU Zyklen zur Verfügung. Selbst wenn ein Durchgang durch den *inner interpreter* 50 Zyklen dauert, wären das immer noch 6912 Runden — das klingt nicht nach zeitlicher Bedrängnis.

Ergebnis: Alle Uhren gehen nach, alle ungefähr gleich viel, und zwar um ca. 24 Sekunden pro Tag (280 ppm). *Nach*–gehen wussten wir schon. *Alle etwa gleich*, das ist schon eher erstaunlich. Wären die Uhrenquarze lausig, dann sollte doch wenigstens eine der vier Uhren vorgehen. Wo immer die Abweichung her kommt, sie sieht systematisch aus.

2 SMD–Quarz

Um die naheliegende Hypothese *Alle (billigen) Uhrenquarze sind lausig* zu testen, hab ich einer Platine einen SMD–Uhrenquarz spendiert. Der lag noch in einer MSP430–Launchpad–Schachtel. Da ist jetzt mal zumindest die Bauform und vielleicht auch der Hersteller verschieden.

Ergebnis: Der SMD–Uhrenquarz macht keinen signifikanten Unterschied. Die Uhrenquarze sind damit erst mal unauffällig.

3 Tick Frequenz

Als Nächstes veränderte ich bei einer Platine die Tick–Frequenz von 32 auf 128 — so war das ja im ersten Artikel auch vorgestellt.

- Timer/Counter2 wird ohne Vorteiler betrieben, zählt auf 256 und macht dort einen *overflow*, d.h. der Zähler wird automatisch 0 ($32768/(256) = 128/s$).

Der Rest ist unverändert.

Ergebnis: Mit der höheren Tick–Frequenz (128 Ticks/s) wird die Abweichung schlechter: 325 ppm statt vorher 280.

Was ich schon ahnte, war jetzt zumindest über einen Tag vergleichend gemessen.

Wieviele Ticks Abweichung pro Tag sind das? Und ist das zufällig ein Vielfaches von Stunde (24), Minute (1440) oder Sekunde (86400)? Wäre es möglich, durch einen Fehler in der Buchführung genau so eine Abweichung zu erreichen?

280 ppm sind 24 Sekunden pro Tag oder $24 \cdot 32 = 768$ Ticks/Tag. Das sieht irgendwie nicht nach einem glatten Vielfachen aus. 325 ppm sind 28 Sekunden bzw. 3584 Ticks pro Tag. Das wird irgendwie auch nicht besser. Diese Frage erscheint also nicht hilfreich.

Interrupts in AmForth

Vielleicht hängt es ja doch irgendwie mit den Interrupts oder der Interrupt–Service–Routine zusammen.

Wenn man in der Dokumentation von AmForth über dieses Thema nachliest [2], dann lernt man, dass ein Interrupt in zwei Schritten bedient wird.

Wenn ein Interrupt ausgelöst wird, dann wird eine Assembler–Funktion aufgerufen (drivers/generic-isr.asm), und zwar die gleiche für alle Interrupts (*top half*). Diese berechnet, welcher Interrupt gefragt ist[2], kopiert die zugehörige Adresse aus der Sprungtabelle in eine Variable und setzt das T–Bit im Prozessor–Status–Register. An dieser Stelle wird *kein return from interrupt* ausgeführt und die Interrupts bleiben gesperrt! Das unterbrochene Wort läuft anschließend weiter.

Bei der nächsten Runde durch den *inner interpreter* (amforth-interpreter.asm) wird geprüft, ob das T–Bit gesetzt ist. Wenn ja, dann wird das T–Bit gelöscht und nach `isr-exec` (*bottom half*) verzweigt. Dieses Wort führt für den angeforderten Interrupt die Forth–ISR aus und ruft danach `isr-end` auf. Erst jetzt wird ein *return from interrupt* ausgeführt und die Interrupts werden wieder erlaubt. Diese zweistufige Behandlung erlaubt, dass eine Interrupt–Service–Routine als normales Forth–Wort definiert werden kann (es darf allerdings die Stacks nicht verändern). Das T–Bit dient der *Signalisierung*. Es wird in der AVR–Dokumentation als Ort beschrieben, in dem der Wert eines einzelnen Bits zwischengelagert werden kann.

Taucht in der Bearbeitungszeit für einen Interrupt ein anderer Interrupt auf, dann wird der vom Kontroller nachgeliefert, sobald die Interrupts wieder erlaubt werden. Er geht nicht verloren.

Taucht in der Bearbeitungszeit für einen Interrupt der gleiche Interrupt erneut auf, dann geht der möglicherweise verloren, beispielsweise wenn man am Ende der laufenden ISR das zugehörige Bit explizit löscht. Aber es ist mir an dieser Stelle nicht alles so glasklar.

Die Behandlung des Interrupts (*bottom half*) würde ebenfalls ausfallen, wenn zwischen der ISR (*top half*) und dem Testen des T–Bits im *inner interpreter* das T–Bit schon vorzeitig gelöscht wurde. Ob und wie auch immer das passieren kann.

Die ISR für den Tick

Was genau macht im vorliegenden Fall die Interrupt–Service–Routine?

```
: tick_isr
  1 ct.timer2 +!
  \ call to this isr means: tick.over
  ct.flags @ $0001 or ct.flags !

  \ advance ct.phase and check for overflow!
  \ compare high bytes only
  \ phase_{i+1} phase_{i} u< ?
  ct.phase    2@      dup >r
  ct.fcontrol 2@ d+   dup >r
  ct.phase 2!
  r> r> u< if
    \ overflow means: second.over
    ct.flags @ $0002 or ct.flags !
  then
;
```

Wie man leicht sehen kann, macht diese Funktion einiges an Buchhaltung.

- die Zählvariable `ct.timer2` wird um 1 erhöht
- der Tick–Flag wird gesetzt
- `ct.phase` wird um `ct.fcontrol` erhöht
- wenn das einen Überlauf produziert, wird der Sekunden–Flag gesetzt

Vielleicht ist das ja alles doch ein bisschen viel. Das muss doch auch kürzer gehen, wenn man die zugehörige Arbeit in das Hauptprogramm verlagert.

4 ISR, stark gekürzt

Theoretisch sollten zwar keine Interrupts verloren gehen, aber Versuch macht kluch.

Die neue Version von `tick_isr` beträgt nur noch eine Zeile:

```
: tick_isr   1 ct.ticks +! ;
```

Die restliche Arbeit wandert in das Hauptprogramm:

```
: msBit?  ( x -- t/f ) $8000 and 0= 0= ;
: tick.over? ( -- t/f )
  ct.ticks.follow @  ct.ticks @  - 0<
;

variable last.msBit
variable ct.ticks.follow
: run-job
  begin
    ledsensor toggle

    tick.over? if
      1 ct.ticks.follow +!
```

[2] Durch die Verwendung von `rcall` anstelle von `rjmp` wird die Adresse der auf `rcall` folgenden Instruktion auf den Stapel des Kontrollers gelegt (das ist auch der Forth–return–Stapel!). Daraus kann man auf die aufgerufene Adresse und damit auf die Interrupt–Nummer schließen. Trickreich!

```
      \ advance phase:
      ct.phase 2@  ct.fcontrol 2@
      d+ ct.phase 2!

      job.tick
    then
```

Zunächst wird geprüft, ob `ct.ticks` von der ISR erhöht wurde. Wenn ja, dann ist mindestens ein Tick vergangen und wir erhöhen unsere Kopie `ct.ticks.follow` um 1. Wenn schon mehrere Ticks vergangen sind, dann ist das zwar von der zeitlichen Abfolge verspätet, aber die Erhöhung findet entsprechend oft statt. Die alte Version des Programms hatte an dieser Stelle ein Leck! Außerdem wird `ct.phase` weitergeführt und `job.tick` aufgerufen.

```
    \ sec.over?
    \ check msBit for "falling edge"
    ct.phase 2@ swap ( LOW ) drop
    ( HIGH ) msBit?
    dup last.msBit @   =  if
      drop
    else
      \ half second over
      ( msbit ) dup 0= if
        \ second over
        timeup
        1 bv tu.flags fset
        1 jobCount !
      then
      ( msBit ) last.msBit  !
    then
```

Anschließend wird geprüft, ob das höchste Bit von `ct.phase` von 1 nach 0 wechselt. Wenn ja, dann ist eine Sekunde vergangen und die entsprechende Buchhaltung zu tun. Jetzt wird auch die Bearbeitung der regulären Jobs (Sekunde ... Jahr) angestoßen.

```
    \ these are run one job per loop,
    \ not all in one go.
    jobCount @
    bv tu.flags fset?
    if
      jobCount @ dup
      Jobs + @i execute
      bv tu.flags fclr
    then
    jobCount++

    pause
  again
;
```

Die neue Version von ISR und Hauptschleife gefällt mir so viel besser, dass ich mich frage, warum ich das nicht schon immer so gemacht habe. Die Variable `ct.ticks` wird nur von der ISR geschrieben, und vom Hauptprogramm nur gelesen. Klare Verhältnisse!

Ergebnis: Die obige Verkürzung der Forth–ISR `tick_isr` ergibt leider keine Veränderung für die Drift der Uhr. So schad!

5 Ticks aus dem Hauptquarz

Könnt's vielleicht sein, dass der asynchrone Takt des Uhrenquarzes manchmal dafür sorgt, dass die Timer/-Counter2 hardware zwar einen Überlauf produziert, das Timing der Flanken aber so ungeschickt ist, dass der zugehörige Interrupt gar nicht ausgelöst wird? Kurz: ist es ein *asynchron*–Thema? Wenn das Problem darin begründet ist, dann müsste es verschwinden, wenn man den Tick aus dem Hauptquarz bezieht.

- Der Takt des Hauptquarzes (11059200/s) treibt Timer/Counter**1**
- Timer/Counter1 wird mit einem Vorteiler von 256 betrieben, zählt auf 1350 und macht dort einen *clear on compare match*, d.h. der Zähler wird auf 0 gesetzt $(11059200/(256 \cdot 1350) = 32/s)$
- Die ISR erhöht die Zählervariable `ct.ticks` um 1
- Die Hauptschleife registriert die Veränderung von `ct.ticks` und reagiert entsprechend

Ergebnis: Leider keine Veränderung.

6 anderer Kontroller: atmega–32

Wie wäre es, wenn man einen *anderen* Kontroller ausprobiert? Die einfachste Variante war für mich, einen atmega–32 zu nehmen. Das ist ein deutlich älterer Kontroller und der würde möglicherweise anders reagieren. Die Änderungen am Code beschränkten sich darauf, ein paar Register– und Bit–Namen anzupassen.

Ergebnis: Die Drift wird größer (440 ppm), und leider nicht signifikant weniger!

7 ISR in Assembler

Die Verkürzung der Forth–ISR hat zwar nicht die gewünschte Veränderung gebracht, aber vielleicht ist das eben immer noch irgendwie *zu lang*, auch wenn es völlig abwegig erscheint? Was passiert, wenn ich die ISR in eine reine Assembler–Funktion umwandle? Leider erfordert das eine Änderung an AmForth selbst. Aber egal und frisch ans Werk:

- eine Variable im RAM anlegen
- die ISR erhöht den Inhalt der Variablen um 1
- die Variable steht auch als Forth–Variable zur Verfügung (`ct.ticks`)
- die ISR wird anstelle der o.g. Standard–ISR direkt an die Adresse `OC2Aaddr` für den *compare match* registriert

```
; 2017-03-27 words/clock_tick_v3.0.asm
; see AvrAssembler2/Appnotes2/m644Pdef.inc
; OC2Aaddr OC2Baddr OVF2addr

; --- RAM space for ct.ticks
.dseg
ct_ticks:        .byte 2
.cseg

; --- register isr for timer2 A compare match
.set pc_ = pc
.org OC2Aaddr
```

```
        rjmp    timer2_overflow_isr
.org pc_

; --- isr
timer2_overflow_isr:
        ;; save state
        push    xl
        in      xl,SREG
        push    xl
        push    xh
        push    zl
        push    zh
        push    temp0
        push    temp1
        push    temp2

        ;; ct_ticks++
        clc     ; clear carry, jibc
        lds     temp0, (ct_ticks+0)
        lds     temp1, (ct_ticks+1)
        ldi     temp2, $01
        add     temp0, temp2
        adc     temp1, zerol
        sts     (ct_ticks+0),temp0
        sts     (ct_ticks+1),temp1

        ;; restore state
        pop     temp2
        pop     temp1
        pop     temp0
        pop     zh
        pop     zl
        pop     xh
        pop     xl
        out     SREG,xl
        pop     xl
        reti

; --- forth variable
VE_CT_TICKS:
        .dw $ff08
        .db "ct.ticks"
        .dw VE_HEAD
        .set VE_HEAD = VE_CT_TICKS
XT_CT_TICKS:
        .dw PFA_DOVARIABLE
PFA_CT_TICKS:
        .dw ct_ticks
```

Diese Datei ist via dict_appl.inc einzubinden und AmForth anschließend neu zu assemblieren. Die unbenutzte Forth–ISR ist in `+ticks` *nicht* mehr zu registrieren. Dann sollte alles so gehen wie vorher.

Zwar musste ich entlang des Wegs erst mal lernen, dass ich `add` und `adc` direkt hintereinander aufrufen muss, weil sonst mein Carry–Bit möglicherweise verschwindet. Außerdem musste ich lernen, dass `inc` das Carry–Bit überhaupt nicht setzt! Um 1 zu addieren muss ich die 1 jetzt erst mal in ein Register schreiben. Alternativ kann man auch −1 abziehen mit `subi`. Das ist aber eine Indirektion, die u.U. Kopfschmerzen bereitet.

Ergebnis: Zum ersten Mal zeigt sich jetzt eine signifikante Änderung! Die Uhr geht jetzt *vor* und nicht mehr nach! Und zwar um ca. 100 ppm oder 9 Sekunden/Tag.

Dieses Ergebnis besagt, dass das Phänomen irgendwo in der Software liegt. Dass die Uhr jetzt zu schnell geht, passt in mein Weltbild, denn ich habe den Uhrenquarzen keine zusätzlichen Last–Kapazitäten spendiert. Damit kann man die Frequenz noch verringern — prinzipiell aber nicht erhöhen!.

Dieses Ergebnis besagt auch, dass das Thema Interrupts und deren Behandlung irgendwie beteiligt ist, auch wenn immer noch nicht klar ist, wie genau.

Streng genommen muss man jetzt noch beim Lesen der 2–Byte Variablen `ct.ticks` in der Hauptschleife die Interrupts sperren, damit man in jedem Fall zwei zusammengehörige Bytes gelesen hat. Sonst kann zwischen dem Lesen vom ersten und vom zweiten Byte die ISR erneut laufen und zwei inkonsistente Bytes produzieren. Das kann man durch Verwendung der Worte `critical[` und `]critical` erreichen. Alternativ mit einer weiteren Funktion `@tick`, die ebenfalls in Assembler geschrieben wird und die beim Kopieren der Werte die Interrupts ausknippst.

Ich habe das aber auf andere Weise geprüft: die Variable `ct.ticks` habe ich künstlich auf 1 Byte beschränkt. Das zieht übrigens eine Byte–Variante der Funktion `0<` nach sich, `c0<`. Einen Unterschied zwischen der 1– und 2–Byte Variante konnte ich nicht feststellen.

DS3231 TCXO

Über einen Artikel in der Zeitschrift Funkamateur wurde ich auf den Uhrenbaustein DS3231 von Maxim aufmerksam. Dieser ist zwar nicht billig, aber dafür ist ein TCXO, ein *temperature compensated crystal oscillator*, drin, mit einer Abweichung von anfänglich unter 5 ppm. Netterweise hat der einen Pin, auf dem ein 32768 Hz Rechtecksignal ausgegeben wird — der kann also als alternative Clock–Quelle benutzt werden.

Das Clock–Signal des DS3231 wird an den Pin T0 (PortB0) angeschlossen und als Clock–Quelle für den Timer/Counter0 benutzt. Der zählt brav auf 256 und löst dann einen Überlauf–Interrupt aus. Die zugehörige ISR (in Assembler) erhöht einen als `ct.ticks` in Forth verfügbaren Zähler. Das Hauptprogramm ändert sich nicht.

Damit verändert man das ganze Experiment, weil sich der Oszillator für den Uhrentakt jetzt auf einem anderen Chip befindet. Vorher, mit dem zusätzlichen Uhrenquarz, befand sich der Oszillator auf dem atmega–644p–Kontroller. Timer/Counter2 kann leider nicht über einen externen Takt angesprochen werden, und leider kann man mit diesem Aufbau den Kontroller nicht schlafen legen, denn nur Timer/Counter**2** kann den Kontroller aufwecken.

Ergebnis: Die Uhr läuft jetzt *mit der Zeit* — nach mehreren Tagen war kein erkennbarer Unterschied zur Funkuhr zu sehen!

ISR Vergleich: Assembler / Forth

An dieser Stelle wurde vorgeschlagen, das externe clock–Signal auch an Pin T1 und damit an Timer/Counter1 zu

verfüttern. Diese Taktquelle würde dann mit einer Forth–ISR bedient, wie schon gehabt. Zwei unterschiedliche Sekundentakte und ebenso zwei `uptime`–Zähler würden zeigen, ob die beiden Verfahren unterschiedliche Ergebnisse liefern. Geschickterweise lassen sich die beiden Timer/-Counter jeweils auf die fallende bzw. steigende Flanke registrieren, die Aufrufe der ISRs sollten sich so nicht in die Quere kommen. Wir können jetzt davon ausgehen, dass beide Verfahren mit dem exakt gleichen Takt versorgt werden. Eigentlich dürfte sich zwischen den beiden Zählern kein Unterschied einstellen.

Ergebnis: Schon nach zwei Stunden betrug die Abweichung zwischen den beiden Zeiten etwa eine Sekunde. Was einerseits beruhigend ist (es bestätigt die oben schon gemachten Aussagen) und andererseits beunruhigend — wir wissen jetzt leider immer noch nicht, wo die Abweichung herkommt.

Interrupts zählen

Ein Kollege von mir vermutete, dass wir von *spurious interrupts* geplagt werden. Um das zu untersuchen, spendierte Matthias eine Erweiterung der *generic* ISR. Sie erhöht einen Zähler (8 bit) in einer RAM–Tabelle, dessen Index die Nummer des Interrupts ist. Mit der folgenden Definition kann man sich die (auf 255 begrenzten) Zählerstände ansehen:

```
: irqdump #31 0 do
    i irq[]#    #4 .r space  \ addr
    i 1+        #2 .r space  \ int number
    i irq[]# c@    u. cr     \ count mod $FF
  loop
;
```

Man muss dabei beachten, dass die Interrupt–Nummern bei 1 anfangen. Die Anzahl der Interrupts ist hier hart kodiert und variiert zwischen den Kontroller–Modellen. Diese Details sind verbesserungsfähig. Hiermit bekomme ich einen neuen Einblick in die Innereien des laufenden Programms geschenkt:

```
~62> irqdump
 275  1 0
 276  2 0
 277  3 0
 ...
 288 14 218
 289 15 0
 290 16 0
 291 17 0
 292 18 0
 293 19 224
 294 20 0
 295 21 50
 ...
 305 31 0
 ok
~62>
```

Das Datenblatt bestätigt: Nummer 14 ist der *timer1 compare match A*–, Nummer 19 ist der *timer0 overflow*– und Nummer 21 ist der *usart0 receive complete*–Interrupt. Das sind die Stellen, die ich sehen wollte. Alle anderen Zähler waren null.

Ergebnis: Auch nach einem Tag hatten sich die Zähler der unbenutzten Interrupts nicht verändert, obwohl zwischen den beiden Methoden die üblichen ca. 20 Sekunden Differenz auftauchten. Unerwünschte Interrupts kommen auch nicht als Ursache in Betracht.

Nur noch eine Frage . . .

. . . was würde passieren, wenn man statt der Akrobatik mit dem T–Bit eine ganz normale Variable (1 Byte) im RAM verwenden würde? Das ist eine moderate Änderung an AmForth, erzeugt allerdings ein paar mehr Instruktionen in dem am meisten durchlaufenen Code–Pfad (*inner interpreter loop*).

Ergebnis: Erstaunlicherweise ist damit die Abweichung zwischen den beiden von Timer/Counter0 und Timer/Counter1 getriebenen Zählern verschwunden.

Hypothese: Das T–Bit wird gelegentlich zur Unzeit gelöscht, auch wenn weiterhin unklar ist, wie das passieren soll. Von den zuständigen Instruktionen tauchen im Assembler–Listing von AmForth nur die erwarteten Befehle an den erwarteten Stellen auf.

Damit war die entscheidende Stelle gefunden, immerhin ein ganzes Bit! Matthias hat quasi sofort reagiert und die Geschichte mit dem T–Bit ersetzt durch ein (bis dahin wenig genutztes) Register (R11). Nachfolgende Tests haben keine Abweichungen vom üblichen Verhalten meiner AmForth–Programme ergeben. Diese Änderungen werden in der Version 6.5 von AmForth enthalten sein, genauer seit Revision 2223.

Last–Kapazitäten

Nur der Vollständigkeit halber wollte ich wissen, ob Lastkapazitäten den Takt von Timer/Counter2 in die Nähe der Realität bringen würden. Bei einer Abweichung kleiner 1 Sekunde/Tag oder /Woche wäre ich echt zufrieden, zumal der TCXO auch Platz beansprucht.

Ergebnis: Ja, man kann den Takt verlangsamen. Und ja, das muss man eben durch Messungen über mehrere Tage individuell anpassen. Im Moment habe ich zwei Kondensatoren mit jeweils 10 pF an den Uhrenquarz (gegen Masse) gelötet. Derzeit bin ich bei ca. 7 ppm Abweichung, und noch ist die Uhr zu schnell.

Schluss

Und die Moral von der Geschicht? So eine Test– oder Messreihe ist recht aufwendig, bringt aber eine Menge Erkenntnisse.

Das Ergebnis war für mich und alle Mitstreiter ziemlich überraschend. Ich hätte nicht erwartet, dass sich an dieser Stelle eine Veränderung einstellt. Es bestärkt mich aber in der Ansicht, dass zum Fehler suchen auch scheinbar irrelevante und abwegige Ideen und Möglichkeiten erlaubt bzw. erwünscht sind.

Mit der neusten Version von AmForth ist mein gewünschtes Weltbild wieder in Ordnung. Ich kann die Uhr jetzt mit einer in Forth geschriebenen ISR realisieren — und das ist schön! Denn ich habe in der letzten Version schon so nette Sachen geschenkt bekommen, wie die *deferred prompts*, die Funktion `>rx-buf` und mehr. Ich kann die Geschichte mit dem RS485–Bus inzwischen ganz ohne Assembler–Zusätze realisieren. Ich kann die Uhr auch an den RS485–Bus anschließen, fällt mir gerade auf `:-)`

Mein Dank geht an alle, die ihre Zeit spendiert haben, um meine Fragen zu erörtern und ihre Erfahrung zu teilen. Das sind insbesondere die regelmäßigen Teilnehmer am donnerstäglichen Forth–Chat via net2o, Bernd, Matthias und Martin, sowie mein Kollege Frank. Ohne Euch wäre ich hier nicht angekommen, vielen Dank!

Mehr denn je sind Kommentare und Erfahrungen der Leser ausdrücklich erwünscht! Man kann mich unter `erich.waelde@forth-ev.de` erreichen.

So Sachen

Ich wurde (berechtigt) gefragt, warum ich überhaupt den ganzen Aufwand mit dem extra Uhrenquarz betreibe, wenn ich doch genausogut mit dem Hauptquarz leben kann. Allerdings kann man den Kontroller weitgehend schlafen legen, Timer/Counter2 läuft aber weiter und weckt den Kontroller per Overflow–Interrupt wieder auf. Zum Strom sparen ist das hervorragend, wenn man das braucht. Das geht mit den anderen beiden Zählern nicht.

Listings

Wie schon beim ersten Artikel ist dieses Programm viel zu lang, um es sinnvoll abzudrucken.

Außerdem eilt die Zeit im Sauseschritt und inzwischen gibt es eine Seite im Forth Wiki [3]. Dort findet sich eine Zusammenstellung der bislang erschienen Artikel zu diesem Thema, sowie Programmquellen zum Herunterladen.

Verweise

1. VD 2016-04, S.15ff — E. Wälde, Clockworks 1 Die kleine Uhr
2. `http://amforth.sourceforge.net/TG/AVR8.html#interrupts`
3. `http://wiki.forth-ev.de/doku.php/projects:clockworks:clockworks`

Abbildung 2: Signale von oben nach unten: **OCR1A** wird vom Timer/Counter1 bei jeden *compare match* umgeschaltet. Dieser Vorgang findet direkt im Kontroller statt, es ist kein Programm im Spiel. **job.tick** erzeugt die Flanke aus der ISR heraus. Die fehlende Flanke deutet darauf hin, dass job.tick nicht aufgerufen wurde. **run-job** erzeugt bei jedem Durchgang durch die Hauptschleife eine Flanke. Die Schleife wird in jedem Tick mehrfach durchlaufen. **job.sec** erzeugt beim Sekundenwechsel eine Flanke, die letzte ist links zu sehen. **job.min** erzeugt beim Minutenwechsel eine Flanke, in diesem Bild ist kein Wechsel zu sehen.

Abbildung 3: Minutenwechsel: `job.min` hat länger zu tun. Die fehlenden `job.tick`–Aufrufe werden nachgearbeitet. Signale wie oben.

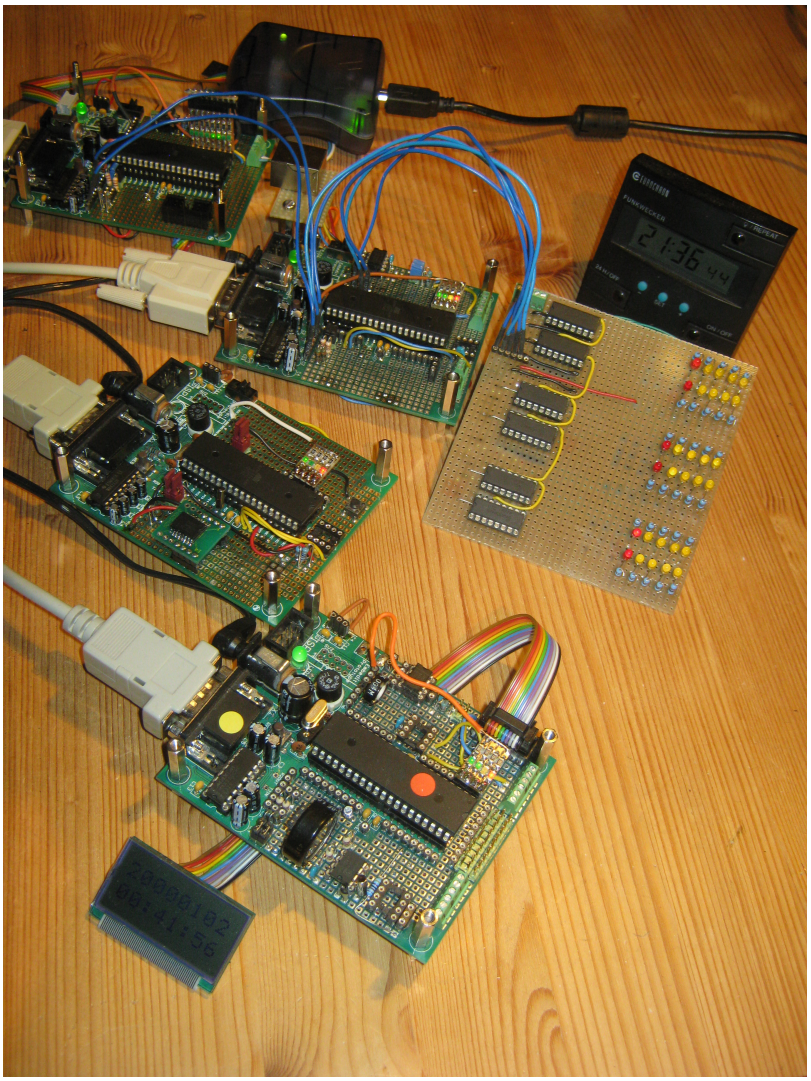


Abbildung 4: Das improvisierte Zeit–Labor

Fortsetzung von Seite 11

Chronos und Kairos — Die zwei Gesichter der Zeit

„Als Gott die Zeit erschuf, hat er von Eile nichts gesagt“, lautet ein afrikanisches Sprichwort. Uns jedoch läuft die Zeit davon. Wir leiden unter chronischem Zeitmangel. Unsere Zeit misst der Chronometer — die Uhr, während in anderen Kulturen die Zeit gewogen wurde — von Kairos, dem Gott des rechten Augenblickes.

Die verschiedenen Wissenschaften nähern sich auf verschiedene Weise dem Phänomen Zeit: Die Geschichtswissenschaft sucht Chronologien zu erstellen und Zeitalter zu definieren, die Psychologie erforscht das subjektive Zeiterleben, die Naturwissenschaft sprach seit Newton von einer absoluten Zeit, bis sich mit Einstein alles „relativierte“, und die Philosophen aller Zeiten verbanden auf die eine oder andere Weise die Zeit mit dem menschlichen Bewusstsein. Dennoch bleibt die Zeit für uns schwer zu verstehen und ebenso schwierig zu beherrschen.

Hannes Weinelt, Abenteuer Philosophie 4/2005
Aus dem Netz gefischt von M. Kalus

Weiterlesen: `http://www.abenteuer-philosophie.com/artikel/102_artikel1_zeit.pdf`

“Vocabulary Prefixing” für Forth–Quelltext–Bibliotheken

Manfred Mahlow

“Vocabulary Prefixing” räumt das Wörterbuch auf, beseitigt den Trennzeichen–Wirrwarr, erschließt zusätzlichen Speicherplatz.

Präfixe und Bibliotheken

Beim Implementieren von Software–Bibliotheken ist es gängige Praxis, den Namen der Sprachelemente einen bibliotheks–spezifischen Präfix voranzustellen. Das ist einfach und flexibel und auch in Forth gängige Praxis. Dort ist es aber leider auch mit ein paar Nachteilen verbunden:

1. Das Trennzeichen zwischen Präfix und Name ist nicht definiert.
2. Die Präfixe belegen zusätzlichen Speicherplatz.
3. Die Präfixe machen die Benutzerschnittstelle unübersichtlich.

Das muss nicht so sein. Ein Präfix repräsentiert ja einfach nur einen Kontext und kann in Forth als ein *Context Switching Word* implementiert werden. Für Forth–Systeme, die Wortlisten unterstützen, ist das mit wenig Aufwand möglich.

Ein Vokabular-Präfix für Forth

Ein *Vocabulary Prefix (VOC)* ist ein *Immediate Word*, das ein Wort aus dem Eingabestrom liest und es in seiner privaten Wortliste sucht. Wird das Wort gefunden, wird das *Execution Token* in Abhängigkeit von der Variablen `STATE` ausgeführt oder kompiliert. Wird das Wort nicht gefunden, gibt es eine Fehlermeldung. Ist der Eingabestrom leer, wird eine Eingabeaufforderung angezeigt und auf eine Eingabe gewartet.

Implementierung

Die Implementierung ist sehr einfach und wurde u.a. für noForth–V [3] und AmForth [4] durchgeführt. Nur ein definierendes Wort (`VOC`) und ein paar kleine Tools werden benötigt (siehe Listing und unter [2] im Ordner templates).

```
voc ( "name" -- )
```

definiert ein neues *Vocabulary Prefix* mit dem Namen `name` und einer leeren Wortliste.

```
<voc> definitions
```

macht die Wortliste des *Vocabulary Prefix* `<voc>` zum aktuellen Compiler Kontext.

```
<voc> words
```

zeigt die Wortliste des *Vocabulary Prefix* `<voc>` an, sowie die in diesem Kontext verfügbaren Tools.

```
<voc> ’ ( "name" -- xt )
```

übergibt das *Execution Token* des Worts `name` auf dem Stack.

```
<voc> ..
```

NOOP–Wort zum Verlassen eines VOC–Kontexts.

Beispiel: eine I2C–Bus Bibliothek

```
voc i2c  i2c definitions

\ Initialize the I2C Bus interface
: init ( -- )    ... ;

\ Start an I2C Bus transmission.
: start ( -- )   ... ;

\ Stop an I2C Bus transmission.
: stop  ( -- )   ... ;

\ Send a byte to the I2C Bus.
: tx ( byte -- ) ... ;

\ Receive a byte from the I2C Bus.
: rx ( -- byte ) ... ;

\ Return a true/false flag if a NAK/ACK was received.
: nak? ( -- f )  ... ;

\ Send a NAK to the I2C Bus.
: nak ( -- )     ... ;

\ Send an ACK to the I2C Bus.
: ack ( -- )     ... ;

\ Throw an error if an NAK was received.
: ?ack ( -- )    ... ;

forth definitions
```

Auf der oberen Ebene des Wörterbuchs ist nur das Präfix I2C sichtbar,

```
WORDS
 I2C  VOC  COMPILE,  TOOLS\  DUMP  .S  NOFORTH\ ...
```

welches Zugriff auf die Wörter der I2C-Bibliothek gibt, z.B.

```
I2C WORDS
 ?ACK  ACK  NAK  RX  NAK?  TX  STOP  START  INIT (9)
 ’  ..  WORDS  DEFINITIONS (4) OK.0
```

Es gibt nur noch ein einheitliches Trennzeichen. Präfix und Wortname werden immer durch ein Leerzeichen getrennt. Das Ende aller Trennzeichen-Konfusion. ;-)

Und der Speicherbedarf? Eine weitere gute Nachricht ist, dass der Speicherbedarf für die Implementierung des *Vocabulary Prefixing* durch die dann kürzeren Wortnamen mit wachsender Codegröße überkompensiert wird.

Referenzen

1. https://wiki.forth-ev.de/lib/exe/fetch.php/events:context-prefixing.pdf
2. https://wiki.forth-ev.de/lib/exe/fetch.php/projects:e4thcom:e4thcom-0.6.1.tar.gz
3. noforth url
4. amforth.sourceforge.net

Listings

```
\ voc : A vocabulary prefix for noForth V                                MM-160205
\
\ Vocabulary prefixes help to structure the dictionary, make it more readable.
\
\ A vocabulary prefix is an immediate word. It reads the next word from the
\ input stream (waits for it), finds it in its private wordlist and then
\ executes or compiles it.
\
\ Usage: #require voc      \ loads this file
\
\        voc name          \ creates a new vocabulary prefix
\
\        name definitions  \ makes name (names wordlist) the current compilation
\                          \ context
\
\ Example:  voc i2c  i2c definitions
\
\             variable sid   \ slave id
\
\             : start ... ;  \ start an I2C Bus transmission
\
\             ...            \ more I2C definitions
\
\           forth definitions
\
\           i2c words    \ shows all i2c words and stays in the i2c context
\                        \ entering a name or .. closes the context
\           i2c start    \ executes/compiles the word start from context i2c
\
\ Requires 286 bytes without debugging tools.
\ ----------------------------------------------------------------------------

fresh

#require wordlist
#require compile,
#require search-wordlist
#require set-current

fresh inside definitions hex

wordlist constant voc-root ( -- wid ) \ the root wordlist inherited by all vocs

value voc-context ( -- wid )  \ points to the wordlist of the last used voc

: search-voc-context ( "name" -- 0 | xt 1 | xt -1 )
  \ Find name in the actual voc context.
    bl-word count ( a u )      \ bl-word waits for input
    2dup voc-context search-wordlist ( adr len wid -- 0 | xt 1 | xt -1 )
```

```
50      ?dup
51      if
52        2swap 2drop
53      else
54        voc-root search-wordlist ( adr len wid -- 0 | xt 1 | xt -1 )
55      then ;
56
57  : dovpx ( "name" -- )
58    \ Read the next word from the input stream, find it in the actual voc context
59    \ and execute (state=0 or word=immediate) or compile it (state<>0).
60      search-voc-context dup 0= ?abort state @ = if compile, else execute then ;
61
62  fresh definitions inside
63
64  {
65  : voc ( "name" -- )
66    \ Create a vocabulary prefix. Generic version, context names are not
67    \ displayed by order and see.
68      wordlist create , immediate
69      does> ( a -- )
70            \ Set the current vocabulary prefix context and execute dovpx.
71            @ to voc-context dovpx ;
72  }
73
74  \ noForth specific version
75  : voc ( "name" -- )
76    \ Create a vocabulary prefix. noForth specific version. Context names are
77    \ displayed by order and see.
78     create wordlist drop chere created lfa>N - chere 1- ROMC!  immediate
79       does> cell+ c@ to voc-context dovpx ;
80
81  voc-root set-current  \ root voc definitions
82
83  : definitions ( -- ) voc-context set-current ;
84
85  fresh definitions
86
87  \ -----------------------------------------------------------------------------
88  \ Tools, only required for debugging (216 bytes):
89  \ -----------------------------------------------------------------------------
90
91  #require show-wordlist
92
93  inside
94
95  voc-root set-current   \ root voc definitions, visible in all voc contexts
96
97  : words ( -- ) voc-context show-wordlist voc-root show-wordlist dovpx ;
98
99  : .. ; immediate  \ return from a voc context to the Forth context
100
101 : ' ( "name" -- xt )
102   \ Return the execution token of the word name in the actual voc context.
103     search-voc-context 0= ?abort ;
104
105 fresh definitions
106
107 \ -----------------------------------------------------------------------------
108 \ voc + debugging tools require 502 bytes.
109 \ -----------------------------------------------------------------------------
110 \ Last Revision: MM-170301 some names changed
111 \                MM-161015 some comments added
112 \                MM-160320 lib renamed to voc ...
113 \                MM-160215 wordlist must be aligned, otherwise order fails
```

Clock Works 4 — Des Rätsels Lösung

Erich Wälde

Im dritten Teil [3] dieser Reihe [1,2] habe ich ausführlich geforscht, warum die Zeit verloren geht. Das Ergebnis war: das T–Bit im Statusregister ist lumpig! Diesen Stand habe ich auch auf der Forth–Tagung 2017 in Kalkar vorgetragen [4]. Das hat einige Diskussionen und auch neue Ideen hervorgebracht. Am Ende hat Matthias Trute *die fragliche Stelle gesichtet: eine ganz schnöde race condition.*

Ticks verlieren

Wir erinnern uns: Es sah so aus, als würde zwar der Timer/Counter2 brav einen Überlauf produzieren (sichtbar an der Flanke in der Aufzeichnung des *logic analyzers*), aber die zugehörige Flanke aus der Funktion `job.tick` fehlte. Mit der Implementierung der zum Überlauf gehörigen Interrupt–Service–Routine (ISR) in Assembler verschwand das Phänomen. Es sah also so aus, als würde gelegentlich (ca. 4 mal in 300 Sekunden, nicht periodisch) die Forth–ISR nicht aufgerufen.

Dieses Phänomen kann man erzeugen, wenn das T–Bit im Statusregister zur Unzeit gelöscht wird, genauer: zwischen der Ausführung der *generic-isr* und der Prüfung des Bits im *inner interpreter*. Eine Idee war, dass das Bit ab und zu einfach wegkippt — aber Adolf Krüger hatte beträchtliche Zweifel an der Hypothese. Er benutzt das T–Bit ebenfalls, um eine langlebige (Tage) Statusinformation zu speichern. Bernd Paysan hat dann das benachbarte Bit im Statusregister an– und ausgeknippst, so schnell es das Programm hergab. Und er konnte über ein paar Stunden keine Aussetzer beobachten.

Allerdings fand Matthias Trute eine verdächtige Stelle in der Funktion `rp!`

```
VE_RP_STORE:
    .dw $ff03
    .db "rp!",0
    .dw VE_HEAD
    .set VE_HEAD = VE_RP_STORE
XT_RP_STORE:
    .dw PFA_RP_STORE
PFA_RP_STORE:
    in temp2, SREG    ; (1)
    cli               ; (2)
    out SPL, tosl
    out SPH, tosh
    out SREG, temp2   ; (3)
    loadtos
    jmp_ DO_NEXT
```

In dieser Funktion wird zunächst das Statusregister gesichert (1). Anschließend werden die Interrupts ausgeschaltet (2). Dann wird eine kleine Arbeit verrichtet und am Ende das Statusregister wieder hergestellt (3). Auf den ersten Blick ist das alles unverdächtig. Auf den zweiten Blick kommt eventuell die Frage auf, warum denn *zuerst* das Statusregister gesichert wird und *danach* die Interrupts gesperrt werden. Macht man das nicht normalerweise in umgekehrter Reihenfolge? Der Vorteil hier ist, dass wir nicht nachsehen müssen, ob das I–Bit überhaupt gesetzt ist. Zuerst sichern wir das Statusregister, dann schalten wir die Interrupts aus — auch wenn sie zu dem Zeitpunkt vielleicht schon aus waren. Bei (3) wird der vorherige Zustand wiederhergestellt. Klingt alles ganz plausibel.

Wenn man dann aber noch 'ne Weile draufstarrt, dann stellt man zwei weitere Eigenschaften fest. Einerseits kann die Funktion `rp!` zwischen den Zeilen (1) und (2) durch eine Interrupt–Service–Routine unterbrochen werden. Verändert diese ISR dann das Statusregister, etwa indem sie das T–Bit setzt, dann ist diese Information nach Zeile (3) leider verloren, weil in der Sicherungskopie nicht vorhanden — dieser Ablauf *löscht* das T–Bit, wenngleich als unbeabsichtigten Nebeneffekt.

Andererseits kehrt das beim Aufruf von Zeile (1) gesetzte I–Bit, also die eingeschalteten Interrupts in Zeile (3) wieder zurück — es ist also gar nicht so, wie es in der Dokumentation steht, nämlich dass die Interrupts gesperrt bleiben bis nach dem Aufruf des zugehörigen Forth–Worts. Merke: die Dokumentation ist nur die Dokumentation und möglicherweise vom richtigen Leben verschieden![1]

Diese Sorte Fehler ist natürlich ein ganz alter Bekannter und hört auf den englischen Namen *race condition*. Die fehlerhafte Stelle ist lediglich eine Instruktion lang. Wie hoch ist die Wahrscheinlichkeit, dass das passiert? Wahrscheinlich gering. Aber `rp!` ist eine der am häufigsten aufgerufenen Funktionen, wenn man den Multitasker benutzt. Und dann tritt das Problem doch so häufig auf, dass man es bemerken kann.

Abhilfe

Ich habe versuchsweise `rp!` geändert. Die gleiche, fehlerhafte Konstruktion findet sich außerdem in den Funktionen `@e`, `!e` und `(i!-nrww)`.

```
VE_RP_STORE:
    .dw $ff03
    .db "rp!",0
    .dw VE_HEAD
    .set VE_HEAD = VE_RP_STORE
XT_RP_STORE:
    .dw PFA_RP_STORE
PFA_RP_STORE:
    cli               ; (10)
    in temp2, SREG    ; (11)
    out SPL, tosl
```

[1] Kürzer: Use the Source, Luke!

```
out SPH, tosh
out SREG, temp2  ; (12)
sei              ; (13)
loadtos
jmp_ DO_NEXT
```

Zuerst werden die Interrupts gesperrt (10). Das führt dazu, dass das I–Bit im Statusregister gelöscht wird. Aber wir wissen es, weil wir es selbst tun. Dann wird das Statusregister gesichert (11), die Arbeit getan und das Statusregister wieder zurückgeschrieben (12). Dann müssen wir allerdings die Interrupts wieder einschalten (13) — wie oben schon ausgeführt, ist das hier möglicherweise falsch, denn ich prüfe nicht, ob die Interrupts vor Zeile (10) überhaupt eingeschaltet waren.

Aber mit dieser Version verringert sich die Abweichung beträchtlich — nicht ganz, vermutlich weil eben noch andere Funktionen betroffen sind.

Erkenntnisse

Es müssen einige Dinge zusammenspielen, damit die oben gefundene *race condition* überhaupt zum Zug kommt: Es muss `rp!` von einer ISR genau nach Zeile (1) unterbrochen werden. Die ISR muss das einzige geeignete Bit im Status–Register (das T–Bit) verändern, und diese Information soll (lange) nach dem Ende der ISR noch Bestand haben. Nur weil mein Programm den Multitasker benutzt, was eben dazu führt, dass `rp!` wirklich sehr oft aufgerufen wird, und nur weil ich die Tick–ISR als Forth–Wort (und nicht in Assembler) realisiert habe, fiel das Problem eben auf. Hätte ich keine Uhr gemacht, ich glaube nicht, dass das überhaupt aufgefallen wäre.

Die beobachteten Phänomene lassen sich jetzt erklären:

- Wenn das gesetzte T–Bit verschwindet, wird die zugehörige Forth–ISR gar nicht aufgerufen. Das erklärt das Aussehen der Aufzeichnungen im *logic analyzer*.
- Das T–Bit wurde aus einer ISR gesetzt. Im normalen Forth–Kontext wäre das Verschwinden nicht passsiert, weil ein normales Forth–Wort nicht zwischen den beiden fraglichen Instruktionen laufen kann.
- Die Assembler–ISR für Timer/Counter2 benutzte das T–Bit überhaupt nicht, daher war das Phänomen ab dieser Änderung verschwunden.
- Verlagert man die Information in eine Variable oder in ein separates Register, dann bleibt die Information erhalten und das Phänomen verschwindet.
- Die Instruktionen `bst`, `clt` und `set` sind völlig unschuldig. Man kann das fragliche Bit auch durch eine `in` und `out` Kombination überschreiben.
- Die Abweichung der Zeit wäre vermutlich sehr viel geringer ausgefallen, würde das Programm ohne Multitasker (quasi im Vordergrund) laufen. Das habe ich aber nicht mehr getestet.

Durch das Verlagern der Information vom T–Bit weg zu einer Variablen (oder einem anderen Register) verschwindet das Problem, weil das T–Bit jetzt gar nicht mehr benutzt wird. Kommt ein findiger Programmierer auf die Idee, das T–Bit aus irgendeiner ISR zu benutzen, um Information (länger) zu speichern, dann taucht das Problem wieder auf.

Das zu frühe Einschalten der Interrupts erzeugt eine weitere *race condition* — in der Zeit zwischen dem Einschalten und der nächsten Prüfung des Register– oder Variablen–Inhalts im *inner interpreter* kann eine ISR den gespeicherten Zustand mit einer anderen Interrupt–Nummer überschreiben. Der vorige Wert ist dann verloren. Wenn der vorige Wert zu einem Interrupt gehört, bei dem man das zugehörige Bit aktiv zurücksetzen muss, dann meldet der sich wieder. Wenn nicht, dann nicht. Wenn der vorige Wert erneut von der gleichen Nummer überschrieben wird, dann sieht das zwar gleich aus, aber der Interrupt wird nur einmal (und nicht zweimal) bedient, was die zugehörige Buchhaltung möglicherweise durcheinanderbringt. Dieser zweite Fall ist vielleicht schwieriger herbeizuführen.

Seit AmForth Revision 2253 wird in *generic-isr* geprüft, ob das Register den Wert Null enthält, bevor ein neuer Wert geschrieben wird. Damit kann man zumindest den Fall erkennen, dass noch eine Interrupt–Behandlung anhängig ist. Wie das weitergeführt wird, ist im Moment noch unklar.

Schluss

Ich finde es wirklich erstaunlich, dass man ein so subtiles Problem ausreichend oft triggern kann, dass es auffällt, und noch erstaunlicher, dass sich das Problem sogar auffinden lässt. Hat man das Problem wirklich verstanden, dann kann man es erfahrungsgemäß auch beheben.

Außerdem finde ich es sehr beruhigend, dass am Ende *nur* ein winziger (*out und cli in ungünstiger Reihenfolge*) und ein gröberer Fehler (*die Interrupts wurden eben nicht bis nach der Abarbeitung ausgeknippst*) in AmForth für das Phänomen zuständig war. Die Wahrscheinlichkeit, einem Fehler im Layout eines so alten Chips zu begegnen, ist zwar nicht Null, aber eben nur ϵ — und das ist in meiner kleinen Welt doch ziemlich winzig.

Mein Dank geht erneut an alle, die ihre Zeit für diese Sache eingesetzt haben. Ohne Euch hätte es auch nicht halb so viel Spaß gemacht!

Verweise

1. VD 2016-04, S.15ff — E. Wälde, Clockworks 1 Die kleine Uhr
2. VD 2017-02, S.12ff — E. Wälde, Clockworks 2 Anzeige a la Abakus
3. VD 2017-03, S.12ff — E. Wälde, Clockworks 3 Auf der Suche nach der verlorenen Zeit
4. `http://wiki.forth-ev.de/doku.php/events:tagung-2017`

Compilation on Demand — Ein einfaches mathematisches Modell

Jens Storjohann

In dem Artikel von Klaus Schleisiek [1] wird eine Lösung des Problems geschildert, dass auf einem Mikroprozessor mit kleinem Speicher, der eine oder mehrere in Forth geschriebene Applikationen ausführen soll, nur die wirklich benötigten Forth–Worte gespeichert werden.

Hier wird in meinem Beitrag die gleiche Situation auf ihren abstrakten Kern reduziert. Die Mathematik, die bekanntlich unnötiges Rechnen vermeiden hilft, kann auch mühevolles Nachdenken ersparen. Dies leisten Kalküle, also Regeln, die ein gewissermaßen automatisiertes Manipulieren von mathematischen Objekten durch den menschlichen Rechner oder den Computer erlauben. Hier nutzen wir den Matrizenkalkül.

Vereinfachte Aufgabenstellung

Wir nehmen an, dass ein Forth–System existiert und eine Applikation darauf aufbauend geschrieben ist. Diese Applikation ruft weitere Forth–Worte, die der Benutzer oder der Programmierer des Systems geschrieben haben, auf.

Details der Target–Compilation oder der Speicherung in ROM oder RAM werden nicht betrachtet. Zur Vereinfachung nehmen wir auch an, dass nur der Doppelpunkt (colon) als compilierendes Wort wirkt. Dies ist aber keine wesentliche Einschränkung. Die Möglichkeit einer rekursiven Programmierung wird ausgeschlossen, wie in einem normalen Forth–System üblich.

Programm im Detail

Die in Assembler geschriebenen Primitives

```
Pa, Pe, Pi, Po, Pu
```

sind gegeben. Sie erhalten die Nummern 5, 6, 7, 8, 9.

Darauf aufbauend werden „bottom up“ die Hilfsworte

```
Hap, Hep, Hip
```

mit den Nummern 2, 3, 4 definiert durch

```
: Hap Pa Pe ;
: Hep Pe Pi ;
: Hip Po Pu ;
```

Die Applikation `Ap` mit der Nummer 1, deren Ausführbarkeit als Einziges gefordert ist, wird compiliert durch

```
: Ap Hap Hep ;
```

Das Problem ist übersichtlich genug, um festzustellen, dass die Worte Pa, Pe Pi, Hap, Hep, Ap geladen werden müssen, die Worte Hip, Po, Pu jedoch nicht (s. Abbildung 1).

Graphen und Adjazenzmatrizen

Wir benutzen Graphen und eine spezielle Beschreibung von Graphen durch Adjazenzmatrizen.

Ein Graph besteht aus Knoten, die zum Teil durch Kanten verbunden sind.

Die Knoten symbolisieren in unserem Fall die Forth–Worte.

Wenn ein Forth–Wort mit der Nummer i in seiner Definition ein Forth–Wort j (direkt) aufruft, wird eine Kante mit *markierter Richtung* vom Knoten i zum Knoten j gezogen.

Eine Adjazenzmatrix eines solchen *gerichteten einfachen Graphen* enthält eine 1 an den Positionen (i, j), wenn das Wort mit der Nummer i zu seiner Definition das Wort mit der Nummer j gebraucht. Ein *einfacher Graph* enthält höchstens eine Kante zwischen zwei Knoten. Solch einen *einfachen* Graphen erhalten wir, denn schließlich definieren wir kein Forth–Wort mehr als einmal.

Mit obigen Programmen erhalten wir für die Adjazenzmatrix A, die die Aufruf–Verschachtelung definiert,

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Wir erinnern uns an die Definition der Adjazenzmatrix und formulieren sie suggestiver: Das Element $A_{i,j}$ gibt die Anzahl der gerichteten möglichen Verbindungswege zwischen den Knoten i und j an, die nur eine Kante durchqueren.

Damit beschreibt $A^2 = A \cdot A$, das Quadrat von der Matrix A, wie viele Verbindungswege es zwischen i und j gibt, wenn man nur solche Wege zulässt, die genau *zwei*

Kanten durchqueren.

$$A^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Entsprechend erhalten wir

$$A^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In diesem Fall ist die dritte Potenz der Adjazenzmatrix die Nullmatrix (eine Matrix mit lauter Nullen als Einträge). Das ist plausibel, denn es gibt keine Verbindungswege, die drei Kanten durchqueren.

Eine Matrix A, die ab einer bestimmten natürlichen Zahl n für A^n die Nullmatrix ergibt, nennt man *nilpotent*. Das ist nichts Schlimmes, weil Matrizen keine Gefühle haben. *Die Regeln des Forth–Programmierens sorgen dafür, dass wir immer eine nilpotente Adjazenzmatrix erhalten.* Also brauchen wir nicht unendlich oft die Adjazenzmatrix zu potenzieren, sondern nur, bis die Nullmatrix erreicht ist.

Ergebnis

Weil wir nur die für die Applikation `Ap` benötigten Forth–Worte laden wollen und wir der Applikation die 1 zugeordnet haben, sehen wir in der ersten Zeile von A alle direkt benötigten und in der ersten Zeile von A^2 alle in der Aufruf–Verschachtelung in zweiter Ordnung benötigten Elemente. Die Zahl 2 an der Stelle (1,6) von A^2 zeigt, dass das Wort `Pe` 2–mal verwendet wird.

Nutzen

An diesem Beispiel sehen wir, wie diese Methode nutzbar gemacht werden kann: Alle verwendeten Forth-Worte bekommen eine Nummer. Alle verwendeten Definitionen werden für Einträge in die Adjazenzmatrix A wie beschrieben herangezogen. Dann werden die notwendigen Potenzen von A gebildet. In den Zeilen der Matrizen $A, A^2, \ldots$, die zur Applikation gehören, finden sich jeweils Einträge an den Plätzen, deren Nummern den benötigten Forth–Worten entsprechen.

Man muss also Zugriff auf die Quellen des Forth–Systems haben und beim Übersetzungsvorgang des Systems Daten gewinnen, die die Aufrufverschachtelung beschreiben.

Selbstverständlich gibt es auch andere Methoden, Graphen zu analysieren als die Potenzen der Adjazenzmatrix. Die Einträge der Wikipedia [2] aus dem Bereich Graphentheorie bieten gute Startpunkte zur Untersuchung von Alternativen.

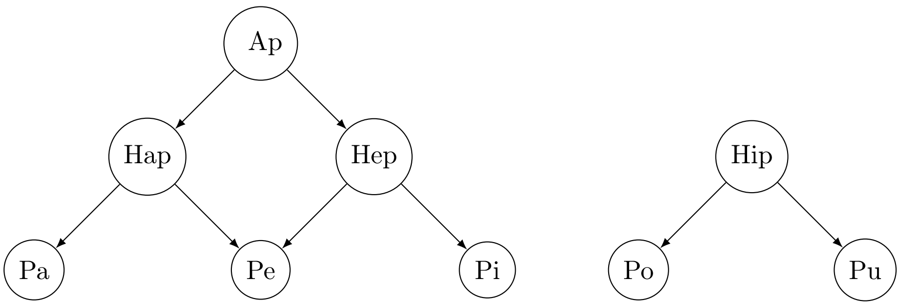


Abbildung 1: Forth-Worte und ihre Aufrufverschachtelung

Referenzen

1. Klaus Schleisiek — *A Compilation on Demand Mechanism* Vierte Dimension 2017-02 S.16ff
2. `https://de.wikipedia.org/wiki/Graphentheorie`

Erlebnisse im USER–Space

Matthias Trute

Forthiker Erich fragte neulich, wie Uservariable in AmForth funktionieren. Wie sie so sind und warum sie so sind, wie sie sind. Und ob man das nicht auch ganz... Die erste Reaktion meinerseits war, ok, das wird ein kleines Rezept für das Amforth Kochbuch [1] und eventuell ein kurzer Artikel für die VD, das ist ja alles in diversen Standards seit Jahrhunderten beschrieben. Es kam anders.

Userarea

Die Userarea. Unendliche Weiten... Sucht man auf der Forth Standard Seite [2] nach `USER`, findet man... *nichts*! Es gibt „user input devices“ und „user output devices“. Aber kein `USER` zwischen `UPDATE` und `VALUE`. Das ist zumindest unerwartet, denn die Forth Community kennt die seit mindestens FIG–Forth Zeiten (1979), wenn man FIG Forth [3] glauben darf. Starting Forth [4] widmet ihnen sogar einige Absätze im Kapitel 9. Und da heißt es so treffend wie vielsagend „User variables are not like ordinary variables.“

Was sind sie also? Allgemeiner Konsens ist, dass Uservariablen genau so funktionieren wie Variablen. Das heißt, sie haben eine Adresse, die man mit `@` oder `MOVE` benutzen kann. Ein wesentlicher Unterschied zu `VALUEs`. Darüberhinaus haben sie irgendwas mit Multitasking zu tun. Die Userarea wird als ein Speicherbereich angesehen, der neben den Stacks in jedem Task lokal zur Verfügung steht. Wenn man eine Uservariable in *einem* Task ändert, hat das keine Auswirkungen auf die gleichnamigen Variablen in *anderen* Tasks. Hier dürfte ein Grund dafür liegen, warum die Userarea nicht im Standard auftaucht: Multitasking ist nicht standardisiert. Die Forth–Gurus haben dies inzwischen als sinnvolle Erweiterung eingeschätzt. Andrew Haley hat für die Euroforth 2017 einen Vorschlag ausgearbeitet [5].

Wenn das kein standardisiertes Thema ist, aber schon seit Jahrzehnten Teil von Forth ist, ahnt man, dass es einiges an Seltsamkeiten geben wird. Das ist so, und dann doch auch wieder nicht. Zunächst die Gemeinsamkeiten. Es gibt immer ein definierendes Wort, mit dem neue Uservariablen anlegt werden. Es heißt fast immer `USER`. Der Name wird im Dictionary hinterlegt, der Aufruf des so definierten Namens liefert sodann eine Speicheradresse im *data space*, die `@` und `!` zugänglich ist. Die Adresse ist in jeder Task eine andere, man kann jedoch eine Systematik erkennen: Es gibt eine Basisadresse für die gesamte Userarea, in den Uservariablen wird nur ein Offset gespeichert. Da das Dictionary global ist, steht der Name in allen Tasks zur Verfügung. Es ist also nicht so, dass Uservariable A im Task 1 auf $\text{basis}_1 + \text{offset}_{A1}$ verweist und in Task 2 auf $\text{basis}_2 + \text{offset}_{A2}$. Es ist vielmehr $\text{basis}_1 + \text{offset}_A$ verglichen mit $\text{basis}_2 + \text{offset}_A$ im zweiten Task.

Zu den Seltsamkeiten gehören die Details, was `USER` neben dem Namen als Parameter erwartet. Das reicht von „gar nichts“ bis hin zu „Position und Größe“, zumindest bei den bekannteren Systemen. Das ist eng mit der Art und Weise, wie die Userarea verwaltet wird, verbunden. Es handelt sich in allen bekannten Systemen um einen maßvoll großen (in Relation zum Gesamtspeicher) bis eher kleinen Bereich des *data space*, der „am Stück“ vorliegt. In aller Regel werden die kleineren Offsets vom System selbst belegt, meisst für IO–Vektoren wie `KEY` und `EMIT`, aber auch Buffer werden hier gesichtet: `PAD` sowie den für den numerischen IO: `<# #s #>`.

Daran schließt sich der Bereich für die Anwendungen an. Wo genau die Grenze zwischen den beiden Bereichen liegt, ist meist nur der Dokumentation zu entnehmen. Portabilität zwischen verschiedenen Forth's oder zwischen den Versionen ist hier kein Designmerkmal.

AmForth überlässt die Verwaltung dieses Bereichs komplett dem Nutzer. Die Größe wird fest einkompiliert und ist danach auch nicht änderbar. Man kann sie über eine Umgebungsvariable `/USER` abfragen:

```
s" /user" environment? drop .
```

Da Umgebungsvariablen bei AmForth eigentlich nur Worte in einer separaten Wortliste sind, kann man sich den Aufwand sparen, die Definition von `ENVIRONMENT?` zu laden und macht es gleich selbst:

```
s" /user" environment search-wordlist drop execute
```

Bei der Definition der Uservariablen wird der Offset für das erste Byte übergeben. Die Größe der Variablen wird indirekt über die Offsets der nachfolgenden Definitionen angegeben. Wer also eine `2VARIABLE` dort platzieren will, muss den Offset um zwei Zellen erhöhen und an `2@` denken. Der Nutzer muss obendrein selbst detailliert Buch darüber führen, wo was ist.

Gforth macht es komplett anders. Man legt lediglich einen Namen fest und das System kümmert sich um den Rest. Dabei wird von einem zell–großen Bereich ausgegangen. Wird mehr (oder weniger) benötigt, muss man sich mit den Details der Speicherverwaltung beschäftigen und landet bei `uallot`. Das funktioniert genau so wie `allot` und setzt den Startpunkt für die nächste Anforderung. VFX Forth arbeitet analog zu AmForth mit direkt angegebenen Offsets. Die Grenze zwischen der Systemseite und der Anwenderseite liegt bei $1000. Swift–Forth weicht ab. Zum einen nennt es das Definitionswort `+USER`. Darüber hinaus werden zwei Angaben erwartet: Einen Offset wie bei anderen Systemen und die Größe des Bereichs.

AmForth	`$0 user name`
gforth	`user name`
swift forth	`$0 1 cells +user name`
vfx	`$0 user name`

Viele Systeme haben einen Pointer auf den Anfang der Userarea, der entweder direkt `UP` heißt und als Variable implementiert ist, oder mit `UP!` bzw `UP@` zugänglich ist. Bei AmForth ist er ein CPU–Register(–paar), und damit sowohl schnell wie auch nur über die zweite Methode erreichbar.

Nutzbarmachung

Wie nutzt man nun den Userspace? Ohne Multitasking funktionieren die Uservariablen genau so wie normale Variablen. Durch die Angabe des Namens erhält man eine Adresse im RAM und kann mit den üblichen Verdächtigen damit arbeiten. Die Bereitstellung der Adresse kann geringfügig langsamer sein, das dürfte jedoch außerhalb von Benchmarks niemand merken.

Mit Multitasking wird es interessanter. Die Feinheiten lassen wir mal außen vor. Im Grundsatz hat ein Task seine zwei Stacks und die Userarea als RAM–Bereich. In jedem Task kann eine Uservariable immer gleich heißen und hat doch unterschiedliche Werte. Eine Variable hingegen ist global für alle Tasks, ändert einer den Wert, ist das unmittelbar für die anderen Tasks gültig. Nun kann ein Task ja durchaus unterschiedliche Dinge tun, da ist eine identische Userarea kaum angemessen. Hier braucht man eine Möglichkeit, die Bereiche individuell zu gestalten. Dies ist mit den Ansätzen von AmForth und VFX/Swift–Forth recht einfach umsetzbar. Hier kann man sogar identischen Offsets unterschiedliche Namen geben und so den Quellcode lesbar halten und doch RAM sparen. Bei gforth hat man keine (einfache) Eingriffsmöglichkeit in die Offsets und hat somit für unterschiedliche Bezeichner auch immer unterschiedliche Offsets. Macht das Leben des Entwicklers einfacher und vermeidet unbeabsichtigtes Überschreiben von Daten, lebt aber auch etwas von den gigantischen Ressourcen der modernen PC Architektur.

Der Zugriff auf Uservariable ist in der Regel auf den eigenen Task zu beschränken. Was aber, wenn man einen anderen Task beeinflussen will? Dann nimmt man etwas, das den anderen Task identifiziert und sagt einfach `HIS` vor dem Zugriff auf die Uservariable. Die Implementierung ist für gewöhnlich stark davon abhängig, wie das Multitasking umgesetzt wird, da aber ziemlich viele Systeme die Startadresse der Userarea (`UP@`) für den laufenden Task auch als Task–ID nehmen, ist das fast schon portabel:

```
\ access user variables of another task
: his ( task-id var-addr -- addr )
  up@ - swap @ +
;
```

Dann erfährt man über die Sequenz `other-task BASE his @` in welchem Zahlenformat der andere Task seine Zahlen konvertiert. Nur ist hier Vorsicht angebracht. Das fängt schon damit an, dass der andere Task vielleicht seine Userarea vor fremden Zugriff geschützt hat (moderne CPU können sowas) oder der Entwickler hat sich entschieden, Platz zu sparen und nimmt den Offset von `BASE` für ganz andere Zwecke. Es kann ja durchaus sein, dass nicht jeder Task eine Zahlenkonvertierung vornehmen soll. Dann liefert die obige Sequenz natürlich keine sinnvollen Angaben.

Was macht AmForth

AmForth nutzt die Userarea sehr weitreichend. Ein Grund hierfür ist, dass die meisten der Angaben ohnehin im RAM liegen müssen. Daneben ist das Thema Multitasking auch für Microcontroller wichtig genug, so dass alles, was in das Schema „gehört in den RAM und ist tasklokal“ in die Userarea gewandert ist. Neben den Daten für den Multitasker selbst sind dies auch der Exception Handler für `CATCH` und der gesamte Command IO. Das schließt sogar `REFILL` und `SOURCE` ein. Das ist dem Ziel geschuldet, dass man nicht nur über einen (meist dem seriellen) Kanal mit dem System interagieren können soll, sondern auch über so exotische Dinge wie einen Kanal auf der Grundlage des RS485 oder I2C. Dann startet man einfach[1] einen weiteren Interpreter mit passenden Angaben und kann parallel Dinge tun.

Eine Forth–Definition dieses Teils der Userarea sähe wie folgt aus:

```
#10 user handler \ exception handling
#12 user BASE    \ numeric IO
```

Die üblichen Definitionen bleiben unverändert, der Code für `THROW` stammt sogar vom Standard selbst:

```
: DECIMAL #10 BASE ! ;
: HEX     $10 BASE ! ;
: .base BASE @ DUP DECIMAL . BASE ! ;
: THROW ?DUP IF
    handler @ rp! R> handler !
    R> SWAP >R sp! DROP R>
  then ;
```

Damit ist es jedoch nicht getan. AmForth bietet darüber hinaus noch die Möglichkeit der `Udefers`, die die `DEFER`–Vektoren tasklokal machen. Sie sind bereits im Kernsystem von AmForth enthalten, lediglich die Verwaltungswerkzeuge sind als nachladbare Source dabei.

```
: Udefer ( offs "name" -- )
    (defer) \ die allgemeine DEFER--Maschine
    ,       \ Adresse des Speicherorts für den XT
    ['] Udefer@ , \ XT für Lesen des obigen Orts
    ['] Udefer! , \ XT für Beschreiben desselben
;
```

Damit kann man sich den IO–Bereich der Userarea als wie folgt entstanden vorstellen:

```
#14 Udefer EMIT   ' tx-poll IS EMIT
#16 Udefer EMIT?  ' tx?-poll is EMIT?
#18 Udefer KEY    ' rx-buf IS KEY
#20 Udefer KEY?   ' rx?-buf IS KEY?

#22 Udefer SOURCE ' source-tib IS SOURCE
#24 user   >IN
#26 Udefer REFILL ' refill-tib IS REFILL
```

[1] *Einfach* ist eher als Metapher zu sehen.

Ohne jetzt allzu tief in die IO–Architektur von AmForth einzutauchen: `tx-poll` sendet direkt an die serielle Hardware sobald sie frei ist, `tx?-poll` also true liefert. Im Zweifel wird aktiv gewartet (polling). `rx-buf` liest aus einem kleinen Ringbuffer, der von einem Interrupt gespeist wird. Dabei wird solange gewartet bis `rx?-buf` true wird, es also ein ungelesenes Zeichen gibt. Diese Worte rufen auch den `PAUSE` Defer auf, der das Herz des Multitaskers ist.

`refill-tib` hat einen Zeilenpuffer, den es von `ACCEPT` füllen lässt und der dann via `source-tib` weiter gegeben wird. Der Name `TIB` hat eine ehrwürdige Geschichte, ist aber nur noch eine historische Fussnote im Quelltext. Aus Nutzersicht gibt es keinen Terminal Input Buffer.

Wenn in einem Task *keine* IO–Befehle ausgeführt werden sollen, steht es dem Entwickler frei, den Speicherbereich für ganz andere Aufgaben zu nutzen:

```
#14 user day
#16 user month
#18 user year
#20 user hour
#22 user minute
#24 user second
```

Damit wird der Speicherbereich, der in einem Task `EMIT` heißt und ein defered XT beinhaltet, in einem anderen Task zu `day`, einer simplen Zahl die keinesfalls geeignet ist, als XT ausgeführt zu werden.

Und weil es so einfach ist, sei noch einmal auf die Value–Manufaktur verwiesen [6], die ihre Dienste natürlich auch für die Userarea anbietet: `Uvalue`. Wer einen `UBUFFER:` oder ein `2USER` benötigt, weiß sie sicher zu benutzen. Konvention ist, dass alle Worte, die mit einem großen U anfangen, irgendwas mit der Userarea zu haben könnten.

Referenzen

1. `http://amforth.sourceforge.net/TG/Cookbook.html`
2. `http://forth-standard.org`
3. `http://tinymicros.com/wiki/FIG_Forth`
4. `http://www.forth.com/starting-forth/`
5. `http://www.complang.tuwien.ac.at/anton/euroforth/ef17/papers/haley-slides.pdf`
6. Matthias Trute — Value Manufaktur in AmForth, VD 2014-01, S.11ff

Neues auf the Forth Net

Gerald Wodni

> *27 Pakete haben es schon auf* `theforth.net` *[1] geschafft und können entweder online durchstöbert werden, oder direkt mit dem Paketmanager* `f` *in Gforth heruntergeladen werden.*

Mitmachen! Hiermit sei jeder dazu aufgerufen, seine Forth–Quelltextsammlung zu durchstöbern und die Schmankerl daraus als Paket der Welt zugänglich zu machen. Wie das geht, kann man sich in den *guidelines* [2] durchlesen, oder man schaut sich einfach das minimalistische Paket `mrot` an.

2013 war diese Kolumne zum ersten mal im Heft, aber seitdem wurde die Website komplett überarbeitet und es wird Zeit, einige dieser tollen, neuen Pakete vorzustellen.

Referenzen

1. `http://theforth.net`
2. `http://theforth.net/guidelines`

keccak

Bernd Paysan

Die Basis für den Secure Hash Algorithm 3 (SHA-3) heißt Keccak. Diesen modernen, kryptologischen Algorithmus kann man dank Bernd in Forth für Prüfsummen und Verschlüsselung einsetzen.

Hash berechnen

```
> s" FORTH" hash256 .st256
6C811C04FB3EB1EF70224848AFCDAA8 ...
```

`Forth` in Textpuffer kopieren:

```
> 5 buffer: text
s" Forth" text swap cmove
```

Verschlüsseln:

```
> text 5 s" Geheim" enc256
> text 5 dump
5D FC C2 59 B8  "{]}..Y."
```

Entschlüsseln:

```
> text 5 s" Geheim" dec256
> text 5 dump
46 6F 72 74 68  "Forth"
```

stack

Matthias Trute

Wem der Daten–, Return– und Gleitkomma–Stack nicht genügt, der möge sich mit diesem Paket etwas Gutes tun und soviele Stacks wie beliebt parallel stapeln.

Anlegen:

```
> 42 STACK constant meins
```

Einkellern:

```
> 4 7 2 meins SET-STACK
> 1337 meins >STACK
```

Auskellern:

```
> meins STACK> .
1337
```

Ausführen:

```
> :noname . false ;
> meins MAP-STACK
7 4
```

mrot (-rot)

Gerald Wodni

Um jedem die Angst zu nehmen, das eigene Packerl könnte doch zu simpel sein, sei hier das trivialste vorgestellt. Es wurde während der ForthTagung in Augsburg erstellt.

```
> 1 2 3 -rot .s
3 1 2
```

stringstack

Ullrich Hoffmann

Wenn einem das `stack`–Paket von Matthias Trute Lust auf mehr Stacks gemacht hat, hier noch ein ganz besonderer Stack, welcher das Verarbeiten von Zeichenketten viel einfacher macht.

Erstellen und ausgeben:

```
> " a string" ".
a string
> " a string" "dup ". ".
a string a string
```

Zusammenfügen:

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

Ersetzen:

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

matmul — Matrixmultiplikation

Anton Ertl

Gleitkommamatrixmultiplikation in Standard–Forth. Dieses Paket ist nicht nur gut für Benchmarks geeignet, sondern macht Lust, mit Matrizen in Forth zu experimentieren.

Tipp: Matrizen werden zeilendominierend gespeichert, außerdem finden sich in `test.4th` einige Wörter, welche bei der Ein– und Ausgabe von Matrizen hilfreich sind.

Erstellen und ausgeben:

```
> 6 floats buffer: a
> a 2 3 1 init-matrix
> a 2 3 mat.
   1.   2.
   3.   4.
   5.   6.
> 6 floats buffer: b
> b 3 2 7 init-matrix
> b 3 2 mat.
   7.   8.   9.
  10.  11.  12.
```

Multiplikation:

```
> 9 floats buffer: c
> a b c 3 2 3 matmulr
> c 3 3 mat.
  27.  30.  33.
  61.  68.  75.
  95. 106. 117.
```

Forth–Gruppen regional

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

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

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

Ruhrgebiet **Carsten Strotmann**
ruhrpott-forth@strotmann.de
Treffen alle 1–2 Monate Freitags im Unperfekthaus Essen
http://unperfekthaus.de.
Termine unter: http://forth-ev.de

Mainz Rolf Lauer möchte im Raum Frankfurt, Mainz, Bad Kreuznach eine lokale Gruppe einrichten.
Mail an rolf@llar.de

Gruppengründungen, Kontakte

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

μP–Controller Verleih

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

Spezielle Fachgebiete

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

Termine

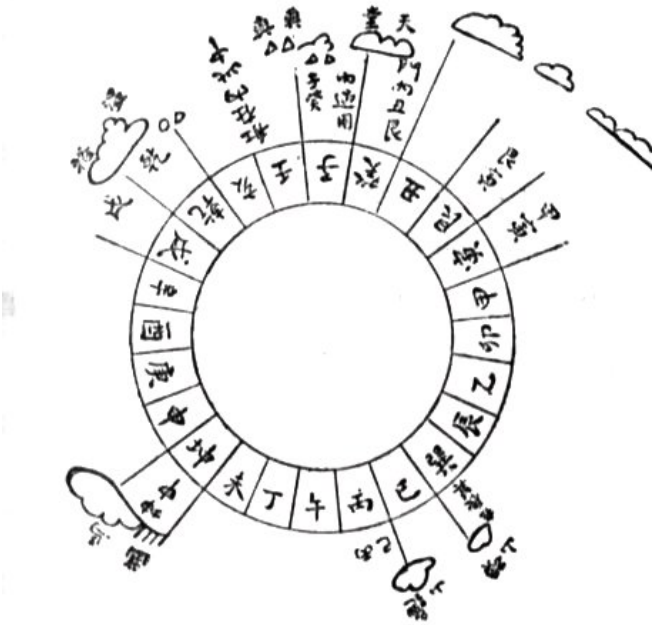
Donnerstags ab 20:00 Uhr
Forth–Chat net2o forth@bernd mit dem Key keysearch kQusJ, voller Key:
kQusJzA;7*?t=uy@X}1GWr!+0qqp_Cn176t4(dQ*

27.–30.12.2017
34c3 in Leipzig
https://events.ccc.de/

10.-11.03.2018
Maker Faire Ruhr in Dortmund
http://www.makerfaire-ruhr.com

05.–08.04.2018
Forth–Tagung in Essen
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.

Einladung zur

Forth–Tagung 2018 vom 5. bis 8. April im Linux Hotel in Essen–Horst

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

Anreise

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

Anmeldung

`http://tagung.forth-ev.de`

Programm

Donnerstag

ab 14:00	Frühankommer Workshops

Freitag

vormittags	Frühankommer Ausflug
14:00	Begin der Tagung, Vorträge und Workshops

Samstag

vormittags	Ausflug
nachmittags	Vorträge und Workshops

Sonntag

09:00	Mitgliederversammlung
12:00	Ende der Tagung