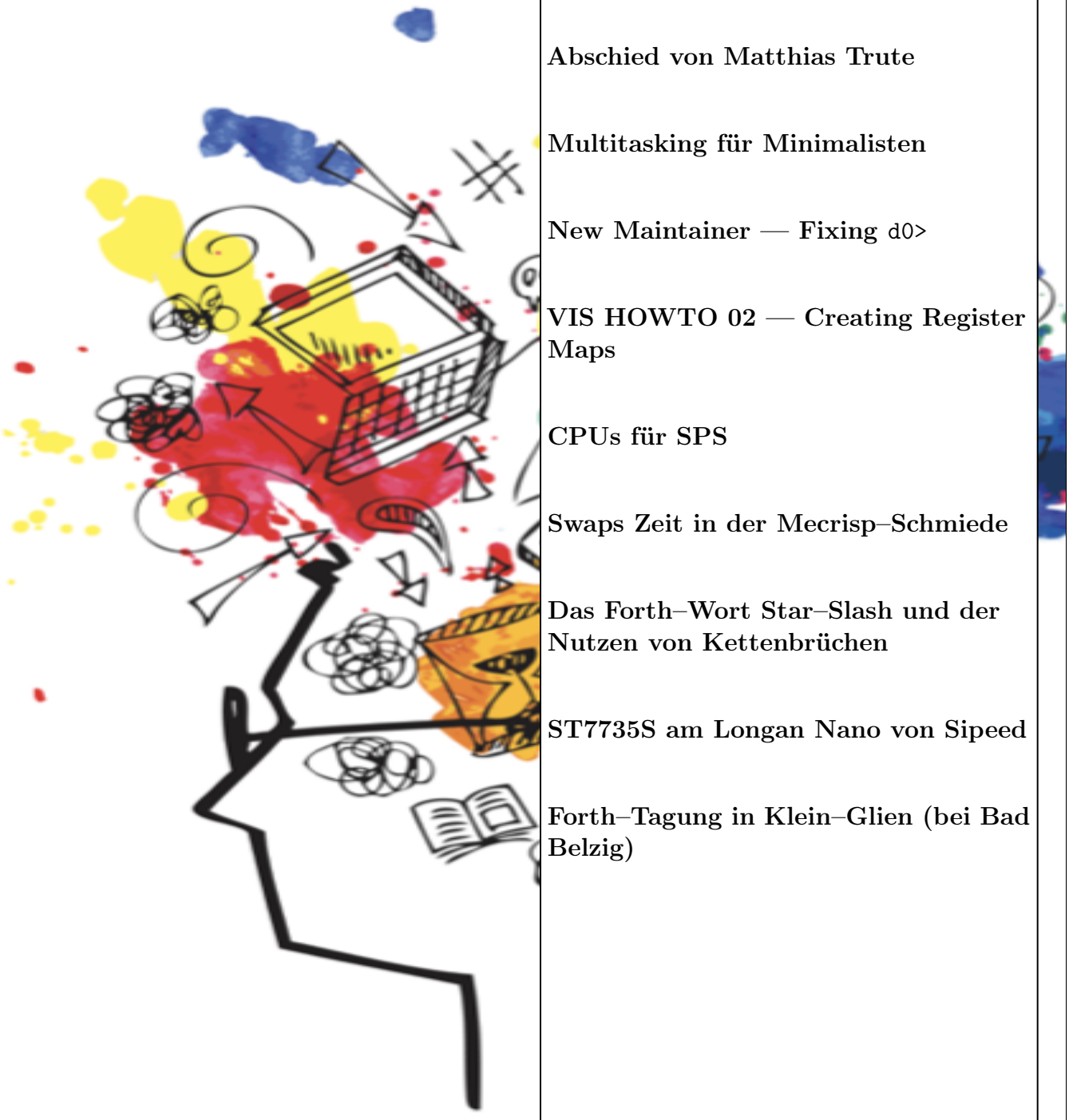
Vierte Dimension

2/2020

Das Forth–Magazin

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

In dieser Ausgabe:



Dienstleistungen und Produkte

tematik GmbH
Technische
Informatik

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

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

Forth–Schulungen

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

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

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

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

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

Cornu GmbH
Ingenieurdienstleistungen
Elektrotechnik

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

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

KIMA Echtzeitsysteme GmbH

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

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

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

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

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

Ingenieurbüro
Klaus Kohl–Schöpe

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

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

Mikrocontroller–Verleih
Forth–Gesellschaft e. V.

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

Titelbild Multitasking AUTOR mk

Quelle: Irgendwo gefischt und modifiziert.

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,

pünktlich zum Sommeranfang ist wieder ein Heft fertig geworden.

32 Seiten stark diesmal, wunderbar. Liebe Leser, liebe Forthfreunde in aller Welt, ihr seid klasse. Bleibt so! Tüchtig Forthiges sammeln und herschicken!

Und nun zum Heft. Zustandegekommen ist es wieder durch beharrliches Nachfragen bei Leuten, die sich mit Forth befassen. Als Korrespondenz, als Artikel oder als Kolumne. Ihr braucht keine besonderen Formate zu beachten für euren Text, da machen wir schon was draus. Wir, das sind derzeit Wolfgang Strauß, Bernd Paysan, Ewald Rieger und ich (Michael Kalus), aber das wusstet ihr ja schon. Wolfgang ist der Lektor, Bernd hält die Technik in Gang, kann exotische Fonts und Mathe–Formate einbauen und unsere Laune hoch halten. Ich bin der Sammler. Ewald schließlich sorgt für den sauberen Druck und Versand der Hefte.

Am Anfang steht — leider — der Abschied von Matthias Trute. Ihr alle kennt ihn und sein AmForth. Erich Wälde hatte damit und so mit ihm zu tun. Und dann kam noch die Meldung, dass auch Andrew Reid verstorben ist. Er war die Verbindung nach Australien.

Sodann im Heft stellt sich Klaus Seegebarth vor, zeigt ein einfaches Multitasking. Ja, so kann es gehen. Lieber Klaus, bleib uns gewogen. Um das *AmForth* kümmert sich nun Erich Wälde, der darin wie zu Hause ist. Und repariert sogleich den `d0>`–Bug. Manfred Mahlow setzt seine Kolumne fort und zeigt uns diesmal seinen Weg, Register–Belegungen übersichtlich darzustellen. Auch Rafael Deliano konnte ich bewegen, mal zu schauen, was er da noch so hat in seinem schier unerschöpflichen Erfahrungsschatz — die seltsamen CPUs für SPS kamen dabei ans Licht. Seinen Flug durch die Zeit bei Matthias Koch schildert uns sodann unser SWAP. Matthias war ja sein Drachenhüter im vergangenen Jahr. Eigentlich sollte SWAP nun schon einen neuen Hüter haben, aber der Verlust aller Versammlungsfreiheiten im Lande kam dazwischen. Wahrlich mehr als traurig!

Aber „die Mathematik ist eine Art Spielzeug, welches die Natur uns zuwarf zum Troste und zur Unterhaltung in der Finsternis.“ (Jean–Baptiste le Rond d'Alembert, französischer Mathematiker, Physiker und Philosoph, 1717–1783). So gibt uns Jens Storjohann Einblick in ein grundlegendes Forthwort: `*/` — doch lest selbst, was es damit auf sich hat. Ganz erfreut war ich neulich beim Treffen im Unperfekthaus in Essen darüber, dass es Martin Bitter tatsächlich geschafft hat, das winzige Display des *Longan Nano* von Forth aus zu steuern. Und noch mehr freut es mich natürlich, dass er seine Notizen davon zu einem Beitrag darüber gemacht hat.

Am liebsten hätte ich ja nun auch schon den Termin bekannt gegeben, zu dem die *Forth–Tagung* in Klein–Glien (bei Bad Belzig) nachgeholt werden wird. Doch stand bis zum Redaktionsschluss noch kein neuer Termin fest.

Bis dahin. Und euch allen eine gute Zeit. Euer Michael

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
`http://fossil.forth-ev.de/vd-2020-02`

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

Ulrich Hoffmann
Bernd Paysan
Carsten Strotmann

Kontakt: Direktorium@Forth-ev.de

Andrew Reid Obituary[1]

Rest in Peace, Andrew, my dear friend from Australia!

Ten years ago Andrew Reid, Managing Director Imaging Associates, Digital Imaging Solutions — asked at `win32forth@yahoogroups.com` "How do I contact people who use Forth in Datalogging applications. I have a project that is beyond my own time available and possibly my ability. I need to make contact with people who program in Forth — not generating a new version of the language."

So I answered "I am working with Forth since 1984 and never wrote a Forth by myself ... I started with Rockwell's RSC–Forth, used UR/Forth for a graphics project which showed the results of insole pressure measurements collected by a battery powered datalogger — I developed the datalogger by myself, using the RTX2000 Forth–Microprocessor ... The project I am currently working on uses a MSP430F2012 from Texas Instruments. I have chosen that microprocessor because the little box shall be cheap and be powered by a coin cell. What kind of datalogger programming are you looking for? Do you have a special hardware in mind or do you need a special microprocessor board? Do you have any specifications? I am sure I am not the only one who uses Forth for real applications. But if I can be of service for you, please let me know." That started an email exchange over many, many years.

His first reply was "a long time ago I used Forth in computers where it was the operating system, interpreter, compiler, and assembler on an old CP/M machine using the Z80. I built a small video digitizer and controller for a small robot back in the 80's. In those days tight code and fast execution was the order of the day. I have a renewed interest in Forth in the Data Aquisition context as you know. I am personally out of touch with Coal Face Programming in Forth. I have interests in Temperature aquisition using 1–Wire devices such as the Maxim DS18B20. I have attached information relating to a product with supporting Software (written for me by one of my past students) not Forth based to be released soon — Mid March 2010. I want to create a Forth based environment using Maxim DS18B20's intially. I haven't the time and I suspect I am too slow or not capable given the marketing objectives I have in mind. There are very few Forth programmers in Australia ... your background is of interest. I am a small business and would like to explore some projects with you."

Each of his emails had a list of links to his business offers of imagingtools and the like attached: "FREE Trial of EzyPic Photo Organizer, Discover a Special Discount on X–Flip Digital Photo Album Software, FREE Trial of Scrapbook MAX! Scrapbooking Software, FREE Trial of FlipAlbum Digital Photo Album Software, FREE chapters from our color management and image resizing ebooks, FREE monitor calibration and digital imaging software, FREE online tools for image resizing and conversion, FREE online tools for monitor calibration."

More than 800 emails he wrote. One of his last emails said "When Forth collapsed in Australia, I obtained the complete Australian Forth Library. If interest reemergences then I will pass the information — They better hurry, I am 74yrs old". A missed chance.

But he had some success with his endeavour: he got the datalogging software he was looking for, written in visualFORTH, using a TI MSP430 LaunchPad to collect temperatures with MAXIMs 1–Wire sensors, using a program written by Brad Rodriguez on his request. And he encouraged me, supported by Peter Knaggs, to attend the 2012 EuroForth conference in Oxford, GB, to speak about 4E4th.

In his last email, two years ago, he wrote "Good o hear from you. Will write more later. Kind Regards Andrew"

Andrew Simson REID, 1943–2019, Beloved Son of Betty & Simson and Father of Simon & Katrina, Passed away on June 6th in Ballarat with Loved Ones at his side.

Ballarat is a city located on the Yarrowee River in the Central Highlands of Victoria, Australia, 120 Kilometers west–northwest of Melbourne.

Links

LaunchPad Educational Environment, Andrew Reid — Seite 12–16 in:

`https://forth-ev.de/wiki/res/lib/exe/fetch.php/vd-archiv:4d2013-03.pdf`

4E4th for the Texas Instruments MSP430 LaunchPad Educational Environment. By Andrew Reid; 17 Seiten, und weitere Webseiten.

`https://issuu.com/imagingassociate/docs/monash_4e4th_paper_v0.8`

Dirk Brühl

Handbuch zum Jupiter ACE gesucht

Auf *de.comp.lang.forth* fragte Volker Pohlers am 29. Jan. 2020:

> „Hallo, besitzt jemand das deutsche Handbuch zum Jupiter ACE (Heimcomputer, 8–Bit, 1982, mit FORTH) und könnte es scannen? Die Version von `http://jupiter-ace.co.uk/usermanual.html` ist leider ziemlich schlecht bis kaum lesbar. Der Webseiten–Betreiber würde sich über eine neue Version freuen."
>
> Volker

Rafael Deliano

[Nun, Dirk Brühl, USA, hat inzwischen nachgeschaut und ist fündig geworden. Eingescannt ist das Büchlein nun schon. Derzeit läuft die Nachbearbeitung: Fleckenentfernung, Seiten ausrichten, Konturverbesserung, OCR und OCR editieren. Das ist ein mühsames Geschäft. Doch

[1] Der E–Mail–Austausch mit Andrew Reid fand in englisch statt, und so gibt es statt einem Nachruf ein Obituary. (DB)

was macht man nicht alles in der langweiligen vergnügungsfreien Coronakrisenzeit ... fertig. Nun gibts das in hübsch. Die Forth–Gemeinde fragen hilft. Fehlt nur noch ein Nachdruck. mk]

Abbildung 1: Jupiter ACE 1983

Vintage Computing — Aus dem Handbuch des Jupiter ACE

Leute, waren *das* Zeiten. Was für ein Theater, bis man eine Datei gespeichert oder geladen hatte! Zur Erinnerung hier mal das OCR der Seite 13:

```
13

Da der Computer verschiedene Programme
auf dem Band vorfinden kann, bevor er zu
dem gewünschten kommt, schreibt er alle
gefundenen Namen von Programmen auf den
Bildschirm. Die Anzeige erfolgt in der
Form:

Dict.: (Name des Programms)

Wenn Ihr Programm vollständig geladen
ist, meldet der Bildschirm OK. Sie können
jetzt das Bandgerät abschalten.

Läßt sich ein Programm nicht laden, oder
läuft das Band zu Ende, ohne daß der ACE
reagiert, sollten Sie

- nachprüfen, ob Computer und Kassetten-
recorder richtig und einwandfrei miteinander
verbunden sind

- nachprüfen, ob der Programm-Name richtig,
auch nach Groß- und Kleinbuchstaben
unterschieden, eingegeben ist

- erneut laden mit verschiedenen Einstellungen
des Lautstärkereglers

- evtl. den Magnetkopf des Recorders
reinigen. ...
```

Solche Mühen motivierten zu Verbesserungen. Da hat sich bis heute ja sehr viel getan. USB–3.0–Sticks mit 256 GByte mal eben so einstöpseln, egal ob Windows– oder Linux–PC, das ist schon eine andere Liga. :)

mk

Nextcloud, Jitsi, GitHub und Twitch

Die Forth–Gesellschaft (FG) freut sich, ihre Mitglieder mit neuen *Internet–Diensten* bei der Arbeit an Forth–Projekten unterstützen zu können.

Auf unserem *Nextcloud–Server,* `https://cloud.forth-ev.de`, könnt ihr eure Dateien speichern und mit anderen austauschen, ähnlich dem kommerziellen DropBox–Dienst. Ihr könnt Termine und Aufgaben in einem Kalender pflegen und mit dem Kalender zuhause auf dem Rechner oder Smartphone abgleichen, gemeinsam an Dokumenten arbeiten sowie Audio– und Video–Konferenzen abhalten.

Auch mit dem Dienst *Jitsi*, `https://meet.forth-ev.de`, sind Video– oder Audiokonferenzen ganz unkompliziert via Web–Browser möglich.

Beide Dienste sind auf Servern der FG in Deutschland gehostet und werden von Mitgliedern administriert. Die Konten auf diesen Servern werden von Carsten Strotmann verwaltet. Wer diese Dienste nutzen möchte, sende eine E–Mail an `cstrotm@forth-ev.de`.

Die FG ist schon seit längerer Zeit auf der Software–Projektplattform *GitHub* als eigene Organisation vertreten. Unter `https://github.com/forth-ev` findet ihr Software–Projekte der Forth–Gesellschaft, z. B. den Docker–Container für die Arbeit am Forth–Magazin oder das VolksForth. Diese *GitHub–Organisation* steht jedem Mitglied offen. Wer dort gerne seine Projekte rund um Forth im Team der Forth–Gesellschaft anbieten möchte, braucht ein Konto bei GitHub und kann sich über das Direktorium in die Organisation aufnehmen lassen.

Auf der Video–Plattform *Twitch*, `https://www.twitch.tv/4ther`, senden Mitglieder, koordiniert von Gerald Wodni, interessante Live–Videos rund um Forth und andere interessante Geschichten.

Im Youtube–Channel *4ther* befindet sich ein Archiv der schon gesendeten Videos, u. a. die Video–Aufzeichnung der virtuellen Tagung vom März 2020.
`https://www.invidio.us/channel/UC_mpkwOO_1ILd66GUTNVPQg`
oder
`https://www.youtube.com/channel/UC_mpkwOO_1ILd66GUTNVPQg`.

Abschied von Matthias Trute

Erich Wälde

Am 25. März 2020 hat Matthias Trute*, Gründer und wohlwollender Diktator des* AmForth*–Projekts, unseren Planeten für immer verlassen. Ich wünsche ihm eine außerordentlich angenehme Weiterreise.*

Es begann im Dezember 2007, als das Sturmtief *Fridjof* um die Häuser brauste. Da habe ich drei Abende lang herumprobiert, gerätselt, gelesen, editiert, assembliert, geladen, gestaunt, gerungen, gezweifelt, vielleicht auch gelötet — um dann doch das erste Mal den `ok`–Prompt von AmForth–2.3 auf dem Bildschirm zu sehen. Ein Glücksmoment!

```
amforth 2.3 ATmega32
>
> words
d2/ d>s s>d up! up@ 0 1ms >< cmove> i! i@ unloop i s
```

Und glücklicherweise kann ich das aus meinen Aufzeichnungen aus dieser Zeit rekonstruieren. AmForth war nicht mein erster Ausflug in die Welt von Forth. Ich hatte schon eine Weile mit Atmels Atmega–Mikrocontrollern herumgespielt. Die *Vierte Dimension* hatte 2007 ein AVR–Sonderheft herausgebracht, darin wurde AmForth–1.3 vorgestellt [2], vom Erfinder Matthias Trute persönlich. Die Befehlsreferenz passte da noch auf die Rückseite des Hefts!

Matthias' Bestreben, AmForth möglichst standardkonform zu gestalten, ist sicher eine Ursache, warum mir sein Forth sofort gut gefallen hat und mir die Benutzung leichtfiel. Für sehr lange Zeit habe ich kein anderes System mehr detailliert angeschaut.

Natürlich gab es Ecken und Kanten, Fehler und unvollständige Dokumentation. Ich berichtete diese Dinge fleißig an die E–Mail–Liste des Projekts [3]. Ich bekam immer Antwort. Und Matthias' Art, Dinge zu beschreiben, gefiel mir. Ich bin sicher, seine kompetenten Antworten haben nicht nur mir ermöglicht, erstaunliche Dinge mit AmForth zu realisieren. In der VD 2008–04 habe ich meinen ersten Artikel veröffentlicht, dessen Programm mit AmForth realisiert war.

Im April 2008 habe ich angefangen, den wöchentlichen IRC Chat der Forth–Gesellschaft zu besuchen. Dort habe ich Matthias wieder *getroffen*[1]. Der wöchentliche Chat mit dem harten Kern aus Bernd, Martin und Matthias wurde mir wichtig. Hier konnte ich auf dem kurzen Dienstweg allerhand Tipps bekommen, wenn ich mich im Buchstabenwald verlaufen hatte. Natürlich ging es da nicht immer nur und ausschließlich um Forth. Der Rest der Welt ist schließlich größer und voller interessanter Winkel. Interessant an diesem Chat ist, dass zwar nur wenig Leute präsent waren, aber die Download–Zahlen des Archivs [4] erstaunlich hoch sind. Wir hatten wohl eine Menge *Mitleser*.

Ungefähr im Sommer 2015 wurde der IRC Chat durch den net2o Chat [8,9] abgelöst. Der harte Kern gab sich Mühe, die Fehlerkäfer dieser neuen Erfindung von Bernd ans Tageslicht zu jagen. In meiner Erinnerung war Matthias immer dabei — naja, fast immer. Und manchmal waren wir auch nur zu zweit.

Irgendwann fragte ich ihn, wo er denn geographisch weilen würde, weil wenn's nicht allzuweit sei, könnte man sich ja vielleicht mal in Person treffen. Damals arbeitete er in Heilbronn und ich in der Nähe von Balingen/Württemberg. Das ist eine durchaus handhabbare Entfernung, und so sattelte ich eines Nachmittags mein schwarzes, benzinfressendes Ungeheuer und brauste zum ausgemachten Treffpunkt. Ich traf einen sehr freundlichen und humorvollen Menschen. Wir speisten etwas und unterhielten uns gut — ganz ohne elektronische Hilfsmittel. Es war ein sehr vergnüglicher Abend, der nur zu früh durch den Blick auf die Uhr unterbrochen wurde. Wir trafen uns noch einmal in Essen (ich weilte im Linuxhotel) und in Stuttgart (er weilte auf einer Schulung). Im Nachhinein bin ich sehr froh, nach einem Treffen gefragt zu haben. Er war nicht der Mensch, der sich im Rampenlicht wohl fühlt. Nur ein Mal besuchte er die Forth–Tagung[2].

Matthias hat *moderne* Ideen für Forth erfolgreich auf den Weg gebracht. Er hat dafür gesorgt, dass die von Anton Ertl (anno 2007) vorgeschlagenen *recognizer* implementiert und ausgestaltet, sowie beim Standardisierungskomitee eingereicht wurden. Natürlich mit der Hilfe von Bernd und anderen, aber er hat das angetrieben. Er hat dieses und mehr in typischerweise sehr knappen, aber gehaltvollen Artikeln in der VD veröffentlicht. Er hat AmForth auch auf weitere Kontroller portiert. Er hatte noch unzählige Ideen, AmForth weiter zu verbessern und für noch mehr Anwender nützlich zu halten. Die Projekt–Dokumentation ist vorbildlich. Selbstverständlich haben andere dem Projekt zugearbeitet, aber Matthias hat es auf Linie gehalten.

Die intensivste Phase unserer Zusammenarbeit war 2016/17. Ich hatte festgestellt, dass bei meinem geliebten Uhrenprogramm ein klein wenig Zeit irgendwie in Ritzen verschwand. So etwas ist natürlich nicht hinnehmbar. Mit erheblichem Aufwand und der Mithilfe der net2o Chat–Truppe ist es uns gelungen, einen Fehler in AmForth zu finden, der nur *ein Bit breit* und *nur eine Instruktion lang* war. Die Details können nachgeschaut [5] und nachgelesen [6,7] werden.

[1] Sofern man jemanden im virtuellen Raum, aka Pixelland, wirklich *treffen* kann.

[2] Leider konnte ich selbst nicht zu dieser Tagung reisen.

Matthias war ein sehr zuverlässiger Mensch. Er war immer im Donnerstags–Chat, und wenn nicht, dann hatte er sich vorher abgemeldet. Und so fiel es uns schnell auf, dass etwas nicht stimmte. Nach einiger Zeit haben Martin und ich altmodische Briefe geschrieben. Seine Gattin klärte uns über die unerwarteten Vorkommnisse auf. Matthias war krank geworden, sein Zustand zu schlecht, um auch nur an eine Tastatur zu denken.

Vor ein paar Wochen habe ich beschlossen, die Webseite und das Code–Repository von AmForth weiterzupflegen. Sein Projekt, dem er so viel Zeit spendiert hat, soll nicht der Vergessenheit anheim fallen. Die Liste der Artikel in der VD, die AmForth–relevant sind, ist beeindruckend. Dieses für sich genommen winzige Projekt hat eine erstaunliche Wirkung entfaltet.

Wolfgang hat dieser Tage eine alte E–Mail aus den Archiven gehoben. Eigentlich ging es um Sphinx [10] und die Mühen der Projekt–Dokumentation. Darin auch dieses Juwel:

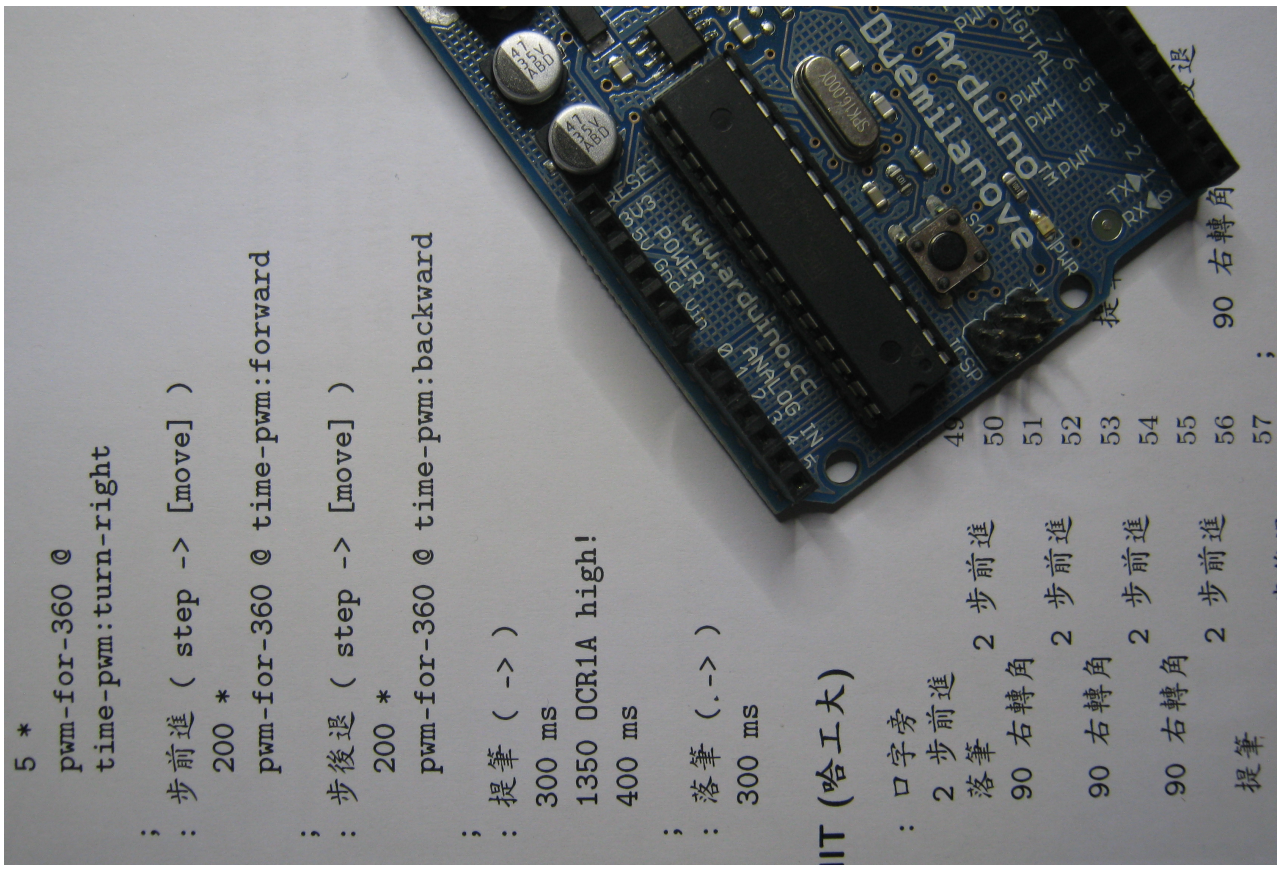
```
> > Erich.
> > Leider ein Pessimist.

Du brauchst mehr Sonne für dein Gemüt ;)

Matthias
```

Matthias ist nun abgereist von diesem Planeten — nach unserer völlig eigennützigen Einstellung natürlich viel zu früh. Wir haben ihn in den vergangenen Monaten schon schmerzlich vermisst. Es ist jetzt Gewissheit, dass er nicht zurückkommen wird. Bestimmt möchte er nicht, dass wir aufhören zu frickeln, zu fortheln, zu frotzeln. Auf geht's zu neuen, zeitfressenden Projekten und Abenteuern!

Gute Reise, Matthias! May the Forth *be with You!*

Verweise

1. https://amforth.sourceforge.net
2. VD 2007–04 *AVR Sonderheft* S.28ff — Matthias Trute, amforth
3. https://sourceforge.net/p/amforth/mailman/amforth-devel/
4. http://wiki.forth-ev.de/doku.php/infos:forthchatirc
5. https://forth-ev.de/wiki/events:tagung-2017:clock-works
6. VD 2017–03, S.12ff — Erich Wälde, Clockworks 3 Auf der Suche nach der verlorenen Zeit
7. VD 2017–03, S.23f — Erich Wälde, Clockworks 4 Des Rätsels Lösung
8. https://fossil.net2o.de/net2o/doc/trunk/wiki/net2o.md
9. http://wiki.forth-ev.de/doku.php/infos:forthchatnet2o
10. https://www.sphinx-doc.org



Auch für ungewisse Aufgaben gerüstet: AmForth *in China (VD2015/03+04).* *Bild: E. Wälde 2020*

Multitasking für Minimalisten

Klaus Seegebarth

Controller ohne viel RAM sind für das typische kooperative Multitasking, das in vielen Forth–Systemen verwendet wird, nicht sehr gut geeignet. Die einfache Kette von Zustandsautomaten in den Warteschleifen der seriellen Übertragung kann Mikrocontrollern Multitasking–Funktionen zum Preis von einer RAM–Zelle pro Task und 200 Byte zusätzlichem Codespace bieten.

Controllers without much RAM are not very well suited for the typical cooperative multitasking used in many Forth systems. The simple chain of state machines inside the waiting loops of serial transfer can bring multitasking capability to microcontrollers at the price of one RAM–cell per task and 200 Bytes additional Codespace.

Neulich beim Aufräumen fiel mir ein *TI–Launchpad* mit dem *MSP43G2553* in die Hände, das ich vor etlichen Jahren gekauft, aber nie benutzt hatte. Mit Dirk Brühls *4E4th–IDE* war schnell eine Entwicklungsumgebung dazu gefunden, allerdings waren einige Links tot. Auch die Infoseite bei `https://forth-ev.de` war gerade umgestellt worden und alle Infos tauchten erst auf, wenn man noch ein „alt.“ in das „www.forth–ev.de“ einzwängte. Wer zu spät kommt, den bestraft eben ab und zu das Leben. Dirk reagierte aber sofort auf meine E–Mail–Anfragen und so konnte ich das Board mit der 4E4th–IDE zum Leben erwecken.

Allerdings störte mich enorm, dass ich während eines Programmlaufs zwar die LEDs blinken lassen konnte, aber die serielle Verbindung zum PC dabei „tot“ war. Ein Mikrocontroller sollte doch mehr können als sinnlos Zeit mit „busy waiting“ zu verplempern.

Karl Meinzer hatte doch schon vor 40 Jahren mit seinem forth–ähnlichen *IPS* vorgemacht, wie sowas auch mit wenig RAM geht.[1] Das Basiskonstrukt ist eine Kette von Programmen, die zyklisch und permanent durchlaufen wird, wie es auch jede *SPS* (engl. PLC) und viele Mikrocontroller mit der typischen `BEGIN task1 task2 task3 ... REPEAT`–Schleife machen. Forth ist dabei die erste Task in der Kette und darf als einzige etwas auf den Stacks hinterlassen. Alle anderen müssen wie Interrupt–Service–Routinen die Stacks geputzt hinterlassen. Legt man diese Programme als Schrittschaltwerke an, so kann man auch komplizierteste Abläufe damit realisieren und hat gegenüber kooperativem Multitasking den Vorteil, dass die Schrittvariablen jederzeit anzeigen, welche Programmteile gerade abgearbeitet werden.

Mich packte der Ehrgeiz und ich wollte so etwas dem Launchpad beibringen. Dazu lud ich mir die von Dirk Brühl und Michael Kalus für die 4E4th–IDE angepasste Version des *CamelForth* herunter und auch den dafür benutzten *NakenAssembler*.

Im ersten Anlauf habe ich aufwändig die forthtypische QUIT–Schleife zerlegt, ein Schrittschaltwerk daraus gemacht und als erstes Programm in die Kette eingefügt, nur um dann festzustellen, dass Eingabe und Blinkprogramm zwar hervorragend gleichzeitig laufen, bei dem Forthwort `WORDS` das Geblinke aber anhält und erst weitermacht, wenn alles ausgegeben ist. `EMIT` macht eben auch „busy waiting“, wenn die Ausgabe über die serielle Schnittstelle erfolgt. Das weiter zu zerlegen, erschien mir denn doch zu aufwändig, soviel Ehrgeiz hatte ich nicht.

Also eine Nacht drüber geschlafen und dann radikal vereinfacht. Alle Testprogramme in die Tonne, die `QUIT`–Schleife bleibt, wie sie ist und die Kette soll nur innerhalb der Wörter `KEY` und `EMIT` anstelle der Warteschleifen aufgerufen werden. Damit müssen nur diese beiden Wörter ersetzt und ein paar Hilfsroutinen dazuerfunden werden.

`KEY?` war ohnehin schon da und ein `EMIT?` als Abfrage ohne Wartezeit war schnell aus der Warteschleife von `EMIT` extrahiert. Aus `KEY` und `EMIT` wurden die Warteschleifen entfernt und zu `KEY@` und `EMIT!` umbenannt. Damit wird der Transport der Zeichen von und zu den UART–Registern erledigt.

Nun mussten die Wartefunktionen neu aufgesetzt werden. Als Erstes wurden zwei Uservariablen `CHAIN0` und `CHAIN1` im RAM reserviert, sie sollen die Execution Tokens der auszuführenden Programme enthalten. Die Kette war mit

```
: CHAIN ( -- )
  BEGIN
  CHAIN0 @EXECUTE
  CHAIN1 @EXECUTE
  REPEAT ;
```

schnell erstellt und damit es keine uninitialisierten Pointer gibt, wurde mit

```
: NOOP ;
```

ein leeres Forthwort erzeugt und mit

```
' ( Tick) NOOP DUP CHAIN0 ! CHAIN1 !
```

die Kette erstmal zur Endlosschleife gemacht. Aufrufen sollte man sie in diesem Zustand also besser nicht.

Nun fehlten noch die Hilfsroutinen, die bei einem empfangenen Zeichen bzw. freiem Ausgaberegister die Kette wieder stoppen. Das machen:

```
: WAIT_FOR_KEYRECEIVED ( -- )
  KEY? IF ' ( Tick) EXIT CHAIN0 ! THEN ;
: WAIT_FOR_EMITPOSSIBLE
  EMIT? IF ' ( Tick) EXIT CHAIN0 ! THEN ;
```

[1] Forth Dimensions Vol.II Nr.4, Page 113

Damit werden KEY und EMIT jetzt zu:

```
: KEY ( -- c )
  ' ( Tick) WAIT_FOR_KEYRECEIVED CHAINO !
  CHAIN KEY@ ;
: EMIT ( c -- )
  ' ( Tick) WAIT_FOR_EMITPOSSIBLE CHAINO !
  CHAIN EMIT! ;
```

Das Programm CHAIN läuft damit immer nur solange, wie Forth nichts zu tun hat. Braucht man den Vorrang der Kette vor dem Benutzer am Bedienterminal, so kann man die Kette in einen Interrupt verlegen. Mir war aber gerade an einer minimalen Lösung gelegen, komplizierter kann man alles immer noch machen. Mit *Listing 1* ergibt sich etwa 1 Hz Blinkfrequenz, also dauert ein Durchlauf der Kette ca. 50 Mikrosekunden.

Damit man mit diesem einfachen Multitasking auch etwas anfangen kann, ist natürlich ein CASE–Konstrukt sehr hilfreich, da man damit Ablaufsteuerungen sehr schön übersichtlich darstellen kann. Das ist im Camelforth aber noch nicht enthalten. Meine Forth–Kenntnisse reichen leider nicht aus, um das standesgemäß in Forth zu erledigen. Ich habe es aber genau wie die UART–Routinen zu Fuß in den 8–kByte–Assemblercode des Systems untergebracht. Sollte sich jemand finden, der das in Forthcode darstellen kann, wäre das eine nette Ergänzung. Ich kann hier nur den Assemblercode dokumentieren, *Listing 2*.

Die Konstruktion ist ein wenig unelegant, da bei ENDOF ein Rücksprung an den Anfang stattfindet und erst von dort zu ENDCASE gesprungen wird. Das war für den Anfang aber viel einfacher zu programmieren und erfüllt zumindest seinen Zweck.

Happy Forthing !

Listing 1

```
1   \ Sourcecode des Blinkprogramms:
2
3   Variable BC \ für BlinkCount
4
5   HEX
6
7   \ Zähler hochzählen und
8   \ bei Grenzüberschreitung zurücksetzen
9   : B1 ( -- )
10    BC @ DUP 5000 < IF 1 + ELSE DROP 0 THEN BC ! ;
11
12  \ Eine Hälfte der Zeit rot leuchten,
13  \ die andere Hälfte grün
14  : B2 ( -- )
15    BC @ 2800 < IF 40 21 ELSE 1 21 THEN ! ;
16
17  : BLINK ( -- )   B1 B2 ;
18
19  ' ( Tick) BLINK CHAIN1 !
```

Listing 2

```
1   ; Camelforth Source Code
2   ; for Naken Assembler
3
4   IMMED(CASE,4,"CASE",DOCOLON)
5   DW lit,bran,COMMABRANCH
6   DW lit,6,COMMABRANCH   ;skip final ENDOF jump
7   DW IHERE  ;Destination for ENDOF backwards jump
8   DW lit,bran,COMMABRANCH
9   DW IHERE,SWAP          ;to be completed by ENDCASE
10  DW COMMANONE
11  DW EXIT
12
13  IMMED(ENDCASE,7,"ENDCASE",DOCOLON)
14  DW DROP     ;remove destination of backwards jump
15  ;now insert jump to endcase for all ENDOFs
16  DW IHERE,SWAP,STOREDEST
17  DW lit,DROP,ICOMMA      ;remove case variable
18  DW EXIT
19
20  IMMED(xOF,2,"OF",DOCOLON)
21  DW lit,OVER,ICOMMA
22  DW lit,EQUAL,ICOMMA
23  DW lit,qbran,COMMABRANCH
24  DW IHERE,COMMANONE,EXIT
25
26  IMMED(ENDOF,5,"ENDOF",DOCOLON)
27  DW lit,bran,COMMABRANCH
28  DW OVER,COMMADEST              ; backwards jump
29  ;now complete final forward jump
30  DW IHERE,SWAP,STOREDEST
31  DW EXIT
```

designed by freepik

New Maintainer — Fixing d0>

Erich Wälde

So here I am, the new maintainer of AmForth *[1]. I will do my very best.*

In August 2019 the mailing list received this simple error report.

```
From: Martin Nicholas via Amforth-devel
Subject: [Amforth] Missing DU<
...
Also, a bug in D0>:
Hmmm, something wrong here I feel:

> (ATmega2560)
> decimal  1553994000. d0> .   1572137999. d0> .
> -1 0  ok
```

At the time I did not have a good idea, on how to handle this, let alone how to fix any error uncovered. But times are achanging. I decided to publish my path, attempts, insights, and decisions in the hope that interested folks can see *it's not rocket science at all.* And I opted for English text in the hope to reach a larger audience. All errors in this text are mine, I'm afraid.

Reproducing the error

When preparing for the annual (german speaking) Forth Tagung 2020 [2] (see also Tagungen [3]) — which was replaced by a video conference like so many others — I started to dig in. A few things were understood quickly:

- There is an assembly version of d0>, which exhibits the bug.
- There is a pure Forth version, which works correctly.
- The sign of the lower word was apparently used to derive the answer, which seemed odd.

At this point I understood the assembly code only partly.

Adding Test Cases

One way to better understand misbehaviour comes through the addition of a useful set of test cases. In order to compare the Forth word with the assembly function, we add the Forth word under a different name:

```
\ file: lib/forth2012/double/d-greater-zero.frt
\ #require d-less-zero.frt
: d0< nip 0< ;
: d0>v1 ( d -- f )
  2dup or >r     \ not equal zero
  d0<  0= r> and \ and not less zero
  0= 0=          \ normalize to 0/-1 flag
;
```

There are simpler ways to declare d0>, however, we are not going to change two things at a time, do we? Thanks. Then we include the Hayes Tester:

```
include lib/forth2012/tester/tester-amforth.frt
```

The test cases I came up with, since we had already observed that the sign of the lower word did influence the result, are these:

```
TESTING d0>
t{         0. d0> ->  0 }t
t{         1. d0> -> -1 }t
t{     $7FFF. d0> -> -1 }t
t{     $8000. d0> -> -1 }t
t{     $8001. d0> -> -1 }t
t{    $10000. d0> -> -1 }t
t{ $00000000. d0> ->  0 }t
t{ $00000008. d0> -> -1 }t
t{ $00000080. d0> -> -1 }t
t{ $00000800. d0> -> -1 }t
t{ $00008000. d0> -> -1 }t
t{ $00080000. d0> -> -1 }t
t{ $00800000. d0> -> -1 }t
t{ $08000000. d0> -> -1 }t
t{ $80000000. d0> ->  0 }t
t{ $80000000. d0> ->  0 }t
t{ $80000008. d0> ->  0 }t
t{ $80000080. d0> ->  0 }t
t{ $80000800. d0> ->  0 }t
t{ $80008000. d0> ->  0 }t
t{ $80080000. d0> ->  0 }t
t{ $80800000. d0> ->  0 }t
t{ $88000000. d0> ->  0 }t
t{ $FFFFFFFF. d0> ->  0 }t
t{ $FFFF7FFF. d0> ->  0 }t
```

These test cases were repeated substituting d0> with d0>v1 or whatever word was going to be inspected. The result was as expected: Failed tests wherever the MostSignificantBits of both halfs of the double word argument were set.

```
> ver
amforth 6.8 ATmega644P ok
> TESTING d0>                ok
> t{         0. d0> ->  0 }t ok
> t{         1. d0> -> -1 }t ok
> t{     $7FFF. d0> -> -1 }t ok
> t{     $8000. d0> -> -1 }t
  INCORRECT RESULT: t{     $8000. d0> -> -1 }t ok
> t{     $8001. d0> -> -1 }t
  INCORRECT RESULT: t{     $8001. d0> -> -1 }t ok
> t{    $10000. d0> -> -1 }t ok
> t{ $00000000. d0> ->  0 }t ok
> t{ $00000008. d0> -> -1 }t ok
> t{ $00000080. d0> -> -1 }t ok
> t{ $00000800. d0> -> -1 }t ok
> t{ $00008000. d0> -> -1 }t
  INCORRECT RESULT: t{ $00008000. d0> -> -1 }t ok
> t{ $00080000. d0> -> -1 }t ok
> t{ $00800000. d0> -> -1 }t ok
> t{ $08000000. d0> -> -1 }t ok
> t{ $80000000. d0> ->  0 }t
```

```
  INCORRECT RESULT: t{ $80000000. d0> ->  0 }t ok
> t{ $80000000. d0> ->  0 }t
 INCORRECT RESULT: t{ $80000000. d0> ->  0 }t ok
> t{ $80000008. d0> ->  0 }t
  INCORRECT RESULT: t{ $80000008. d0> ->  0 }t ok
> t{ $80000080. d0> ->  0 }t
  INCORRECT RESULT: t{ $80000080. d0> ->  0 }t ok
> t{ $80000800. d0> ->  0 }t
  INCORRECT RESULT: t{ $80000800. d0> ->  0 }t ok
> t{ $80008000. d0> ->  0 }t ok
> t{ $80080000. d0> ->  0 }t
  INCORRECT RESULT: t{ $80080000. d0> ->  0 }t ok
> t{ $80800000. d0> ->  0 }t
  INCORRECT RESULT: t{ $80800000. d0> ->  0 }t ok
> t{ $88000000. d0> ->  0 }t
  INCORRECT RESULT: t{ $88000000. d0> ->  0 }t ok
> t{ $FFFFFFFF. d0> ->  0 }t ok
> t{ $FFFF7FFF. d0> ->  0 }t
  INCORRECT RESULT: t{ $FFFF7FFF. d0> ->  0 }t ok
time:  9.46132898331  seconds
```

Adding a new Function and the Joys of `rjmp`

So I set out to add another assembly function `d0>e0` to my AmForth system, starting with a copy of `d0>`. I created a new file `words/ew-d-greaterzero.asm` and added its name to `dict_appl.inc`. The first round of error messages:

```
... error: Duplicate label: 'VE_DGREATERZERO'
... error: Duplicate label: 'XT_DGREATERZERO'
... error: Duplicate label: 'PFA_DGREATERZERO'
```

This is ok, because these labels are now used twice. So we rename them in the additional definition. The second round of error messages is a little more subtle:

```
... error: Relative branch out of reach
... error: Relative branch out of reach
... error: Relative branch out of reach
```

Oh my! After staring at it for a bit it dawned on me, that the *tail call optimization*, i.e. `rjmp PFA_ZERO1` did not work, because the new word was included too far away for the available address range of `rjmp`; it could not reach `PFA_ZERO1` or `PFA_TRUE1`. I solved this by copying the relevant code and changing the labels. Including this function into the `nrww`–section did not work immediately, so I decided to copy the missing pieces.

```
VE_DGREATERZERO_E0:
    .dw $ff05
    .db "d0>e0",0
    .dw VE_HEAD
    .set VE_HEAD = VE_DGREATERZERO_E0
XT_DGREATERZERO_E0:
    .dw PFA_DGREATERZERO_E0
PFA_DGREATERZERO_E0:
    cp tosl, zerol
    cpc tosh, zeroh
    loadtos
    cpc tosl, zerol
    cpc tosh, zeroh
    brlt PFA_ZERO_EW1           ; test negative flag
    brbs 1, PFA_ZERO_EW1        ; test zero flag
    rjmp PFA_TRUE_EW1

;;; FALSE
PFA_ZERO_EW1:
    movw tosl, zerol
    jmp_ DO_NEXT

;;; TRUE
PFA_TRUE_EW1:
    ser tosl
    ser tosh
    jmp_ DO_NEXT
```

This code could be assembled and loaded. Test cases for `d0>e0` did produce the same errors as the original `d0>` — so we were good to go.

Unveiling the Error

Reading the AVR Instruction Set Document did not immediately reveal, why things went wrong. It occured to me that maybe loading the lower half of the argument later was somehow producing an undesired effect. So I copied the most significant word into temporary registers `temp0` and `temp1`, then called `loadtos`. Now all four bytes were available for inspection.

Then I did the comparison against `zerol,h` of all bytes, but in a different order: from least significant byte to most significant byte. This was a change from the original function!

```
VE_DGREATERZERO_E0:
    .dw $ff05
    .db "d0>e0",0
    .dw VE_HEAD
    .set VE_HEAD = VE_DGREATERZERO_E0
XT_DGREATERZERO_E0:
    .dw PFA_DGREATERZERO_E0
PFA_DGREATERZERO_E0:
    mov temp1, tosh       ; copy high word to temp
    mov temp0, tosl       ; . space
    loadtos               ; load low word
    cp  tosl,  zerol      ; compare against zero,
    cpc tosh,  zeroh      ; . start from LSByte
    cpc temp0, zerol      ; . order is significant
    cpc temp1, zeroh      ; . for "less than" (brlt)
    brlt PFA_ZERO_EW1     ; negative, done (false)
    brbs 1, PFA_ZERO_EW1 ; zero, done (false)
    rjmp PFA_TRUE_EW1     ; positive! done (true)

;;; FALSE
PFA_ZERO_EW1:
    movw tosl, zerol
    jmp_ DO_NEXT
;;; TRUE
PFA_TRUE_EW1:
    ser tosl
    ser tosh
    jmp_ DO_NEXT
```

And to my surprise and relief, this function passed all tests! But why?

Clearly some more staring was needed. The original code did inspect the four bytes in the order `word_H.l`

word_H.h word_L.l word_L.h. The last byte inspected would determine, whether the MSBit was set or not. If it was set, then the argument was negative, right? The last byte inspected originally was word_L.h — that explains the error.

Testing the zero flag does not depend on the order of inspection, but testing the less than flag does.

But can we do better?

Now we could commit this function and be done. However: copying the high word seems like a waste of cycles somehow, doesn't it? Yes it does. If we just inspect word_H.h and see if that is negative, we are done already, right? Yes. So can't we exit prematurely then? Of course, we can.

```
...
PFA_DGREATERZERO_E1:
        cp  tosh, zeroh
        brlt PFA_ZERO_EW1
```

Well — the test cases produced funny results, of course. That is why they are repeatable with almost no effort! While we can certainly decide on the MSBit, we should clean up the stack before exiting.

```
...
PFA_DGREATERZERO_E1:
        cp  tosh, zeroh
        brlt PFA_DGREATERZERO_E2
...
PFA_DGREATERZERO_E2:
        loadtos
        rjmp PFA_ZERO_EW1
```

This works, the new branch corresponds to drop 0.

But then Bernd came along and said: *Why don't you use* zero nip *instead?* Well, yes I could indeed. In the end, I counted the instructions and decided for that.

```
...
PFA_DGREATERZERO_E2:
        movew tosl, zerol
        rjmp PFA_NIP_EW1
;;; NIP
PFA_NIP_EW1:
    adiw yl, 2
    jmp_ DO_NEXT
```

Final Version

So, what does d0> really need to do?

1. **If** the highest bit of the double word argument on the stack is set, this number is negative and we are done with the result false. Well almost — we either need to drop zero or to zero nip to get the stack right.
2. **Else If** all (four) bytes of the double word argument are zero, then the argument was zero, the answer is false and we are done.
3. **Else** we have a positive argument and the result is true.

So the changed version looks like this now:

```
; ( d -- flag )
; Compare
; compares if a double double cell number is greater 0
VE_DGREATERZERO:
    .dw $ff03
    .db "d0>",0
    .dw VE_HEAD
    .set VE_HEAD = VE_DGREATERZERO
XT_DGREATERZERO:
    .dw PFA_DGREATERZERO
PFA_DGREATERZERO:
    cp tosh, zeroh
    brlt PFA_DGREATERZERO_FALSE ; negative, done (false).
    cpc tosl, zerol
    loadtos
    cpc tosl, zerol
    cpc tosh, zeroh
    brbs 1, PFA_ZERO1           ; zero, done (false).
    rjmp PFA_TRUE1              ; positive! done (true)
PFA_DGREATERZERO_FALSE:
    movw tosl, zerol            ; ZERO
    rjmp PFA_NIP                ; NIP
```

This roughly corresponds to a Forth version like this

```
: d0> ( d -- f )
    dup $8000 and if
        drop false nip      \ d is negative
    else
        0= swap 0= and if
            false           \ d is zero
        else
            true            \ d is positive
        then
    then
;
```

Epilogue

As usual: *Afterwards, everything is obvious!*

I would like to thank Martin Nicholas for reporting this, Tristan for adding a few observations, Bernd Paysan and Anton Ertl for helpful comments. This code is going to be the first commit on the AmForth repository as the new maintainer.

Verweise

1. https://amforth.sourceforge.net
2. http://wiki.forth-ev.de/doku.php
3. http://wiki.forth-ev.de/doku.php/events:start

VIS HOWTO 02 — Creating Register Maps

Manfred Mahlow

Modern MCUs have lots of peripheral modules with lots of registers. In Forth, register access is usually done via constants, defined on a register–by–register basis. That's okay, when there is only one module of a specific type, but it's getting tedious when several module instances share the same register map. Then it's more efficient to implement a register map by combining a module specific base address with register specific address offsets.

Example: MSP432 GPIO Register Map

Let's take the TI MSP432 MCU as an example. The TI MSP432 line combines a 32–bit ARM Cortex–M4F CPU with perpherals similiar to those of the TI MSP430 MCU line.

The GPIO modules of the MSP432 MCU can be used in 8– or 16–bit mode. Here we'll implement the Register Map for the 8–bit mode. VOCs and ITEMs [1, 2] allow to do it in a simple straightforward way.

The code examples are for Mecrisp–Stellaris (msp432p401r–ra) with the VIS extension `vis-0.8.2-mecrisp-stellaris.txt` [4].

Creating the Register Map

The GPIO modules (the I/O ports) of the MSP432 MCU are named `P1` to `P10` and the related registers are denominated with a suffix, e.g. `P1IN` , `P1OUT` , etc.

For the implementation, the register names are split into the prefix and the suffix, e.g. `P1 IN` , `P1 OUT` , etc.

Listing 1: MSP432 GPIO Register Map

```
1   \ GPIO -> vis-msp432-gpio.fs
2   \ ----------------------------------------
3   \ Mecrisp-St : MSP432 GPIO Register Map
4   \ ----------------------------------------
5
6   ONLY FORTH DEFINITIONS  DECIMAL
7
8   : +: ( "name" n -- ) <BUILDS , DOES> @ + ;
9
10      \ n +: name ( a -- a+n )
11
12
13  VOC GPIO  GPIO DEFINITIONS
14
15  $00 +: IN ( a1 -- a2 )
16  $02 +: OUT
17  $04 +: DIR
18  $06 +: REN
19  $08 +: DS
20  $0A +: SEL0
21  $0C +: SEL1
22
23  $16 +: SELC
24
25  $18 +: IES
26  $1A +: IE
27  $1C +: IFG
28
29  : PORT: ( "name" a -- ) ITEM CONSTANT ;
30
31  FORTH DEFINITIONS
32
33  $40004C00 GPIO PORT: P1 ( -- a1 )
34  $40004C01 GPIO PORT: P2
35  $40006C20 GPIO PORT: P3
36  $40007C21 GPIO PORT: P4
37  $40024C40 GPIO PORT: P5
38  $40025C41 GPIO PORT: P6
39  \ ...
40
41  \ ----------------------------------------
42  \ Last Revision: MM-200407
43  \
44  \ Usage: <PORT> <REG> ( -- c-addr )
45  \
46  \  i.e.:  P1 IN ( -- c-addr )
47
```

The suffixes are defined inside a vocabulary that is created with a vocabulary prefix (Listing 1, lines 13, 15 to 28) and the register prefixes are defined as context switching items that return a modules base address on the stack and activate the suffix context (Listing 1, line 29, 31 ff.).

Using the Register Map

Using a register map could not be made simpler. Define it once and load it on demand (e.g. load directive for e4thcom terminal in Listing 2, line 30).

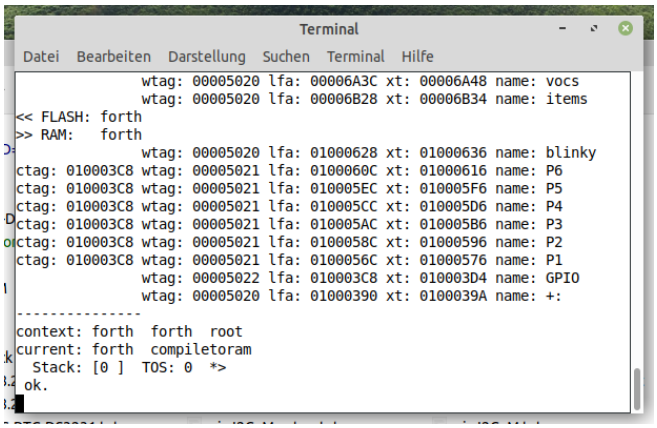
Listing 2:

```
1   \ vis-msp432-blinky.txt            MM-200407
2   \ ----------------------------------------
3   \ derived from  mecrisp-stellaris-2.5.3\
4   \               msp432p401r/blinky.txt
5
6   \ Colourful blinking
7
8   \ P1.0: Red LED
9   \ P1.1: Switch 1
10  \ P1.2: Terminal RXD
11  \ P1.3: Terminal TXD
12  \ P1.4: Switch 2
13
14  \ P2.0: RGB LED Red
15  \ P2.1: RGB LED Green
16  \ P2.2: RGB LED Blue
17
18  \ Now replaced with the GPIO Register Map
19  \ ----------------------------------------
20  \ $40004C00 constant P1IN
21  \ $40004C01 constant P2IN
22  \ $40004C02 constant P1OUT
23  \ $40004C03 constant P2OUT
24  \ $40004C04 constant P1DIR
25  \ $40004C05 constant P2DIR
26  \ $40004C06 constant P1REN
27  \ $40004C07 constant P2REN
28  \ ----------------------------------------
29
30  #require GPIO  \ the GPIO Register Map
31
32  forth definitions
```

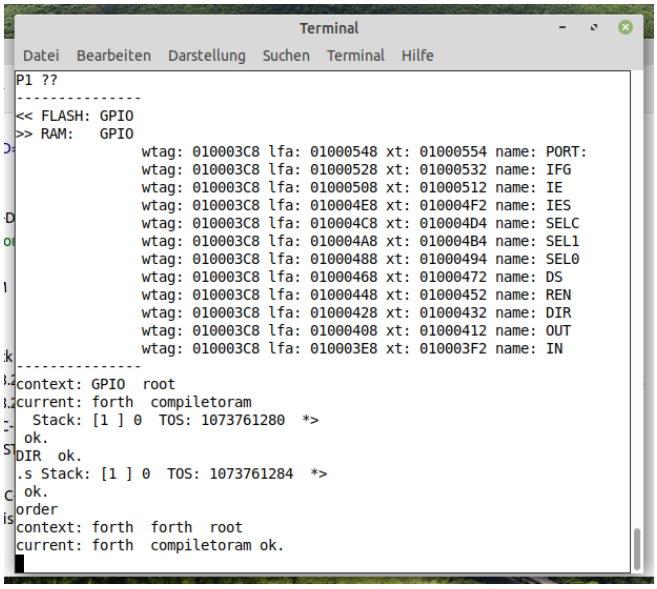
```
33
34  : blinky ( -- )
35    7 P2 DIR c! \ All three pins should be outputs
36    begin
37      1 P2 OUT c!
38      500000 0 do loop
39      2 P2 OUT c!
40      500000 0 do loop
41      4 P2 OUT c!
42      500000 0 do loop
43    key? until
44    0 P2 OUT c!
45  ;
46
47  \ ----------------------------------------
48  \ Last Revision: MM-200407
49
```

Browsing the Dictionary

Now let's use the VIS dictionary browser `??` to look at the Mecrisp–Stellaris dictionary on a MSP432 Launchpad after loading `vis-blinky.txt`.



The GPIO modules are visible as `P1` to `P6` in the Forth context. Entering a port name `Px` puts the GPIO modules base address on the stack and activates the `GPIO` search order.



The next entered register name then adds the registers address offset to the ports base address on the stack and returns from the `GPIO` search order back to the `FORTH` search order.

It's important to realize, that context switching is done in interpret or compile mode only, not in execute mode. Context switching is immediate and not compiled.

Register maps can be nice for testing and debugging. All of a modules register names are at your hands and the Forth dictionary may become more what it is called to be, a dictionary.

Unsure about a register name? Ask the dictionary with `<module> words` , e.g. `GPIO words` .

Referenzen

[1] Manfred Mahlow, *VOCs ITEMs und STICKY Words*, Forth–Tagung 2019

https://wiki.forth-ev.de/lib/exe/fetch.php/events:ft2019:vocs_items_und_sticky_words.pdf

[2] Manfred Mahlow, *VOC statt VOCABULARY*, Forth–Tagung 2019

https://wiki.forth-ev.de/lib/exe/fetch.php/events:ft2019:voc_statt_vocabulary.pdf

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

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

CPUs für SPS

Rafael Deliano

Der Erfinder der SPS (Abb. 1) legte 1969 Wert darauf, dass man seine speicherprogrammierbare Modicon 084 nicht als Computer bezeichnete. Ein Begriff, der dem Kunden Komplexität und Unzuverlässigkeit suggeriert hätte. Auf Dauer konnte man den LSI–ICs aber nicht widerstehen.

1977 war das Jahr, in dem der *Mikroprozessor* als Begriff über die Medien in der breiten Öffentlichkeit auftauchte. Die 3. Generation konventioneller 8–Bit–CPUs fand bald danach ihren Massenmarkt in den ersten Mikrocomputern. Die Industrie hatte ab 1975 die geschrumpfte Alternative zum Minicomputer im Prinzip akzeptiert. Mikroprozessoren waren ähnlich wie diese aber immer noch Bauteilgräber aus SSI– und LSI–ICs, die mit Bussystemen in Racks untergebracht waren. Die Stückzahlen stiegen zwar kontinuierlich an, aber eine Revolution hatte (noch) nicht stattgefunden.

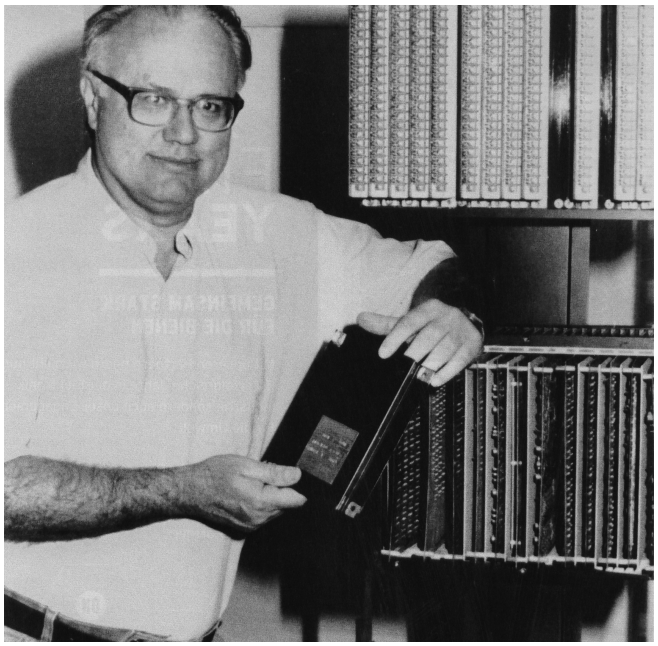


Abbildung 1: Richard Morley

Für Steuerungen war nicht Rechenleistung, sondern eine kompaktere, billigere Alternative nötig. Die logische Entwicklungslinie der Halbleiterhersteller war der NMOS–Einchip–Mikrocontroller mit abgespecktem Befehlssatz und Masken–ROM. Er wurde von diversen Herstellern angekündigt. Intel lieferte 1977 bereits 8–Bit–Varianten mit EPROM.

Die industriellen Anwender verwendeten ehedem als Logik–Bausteine Relais (Abb. 2). Langsam, aber störfest. Ab Ende der 60er Jahre wurden von mehreren Herstellern alternativ zur TTL–Logik störfeste bipolare 12V–Logik–ICs angeboten. Die IC–Familien waren untereinander inkompatibel, der Markt blieb klein. Kompakter als Relais, aber genauso unflexibel, weil fest verdrahtet. Siemens und Teledyne stellten die Produktion dieser ICs erst in den 80er Jahren ein. Die erfolgreiche Konkurrenz war die ab den frühen 70er Jahren von RCA verfügbare CD4000–Logik mit 15V–Versorgung. Ähnlich hohe Störfestigkeit, ideal niedriger Stromverbrauch. Die deutlich höhere Integrationsdichte von Metal–Gate–CMOS kam komplexen ICs wie Zählern zugute. Das reichte aber nicht für Mikroprozessoren. Sowohl RCA als auch Intersil fertigten ihre konventionellen 8–Bit–CMOS–Mikroprozessoren als Si–Gate.

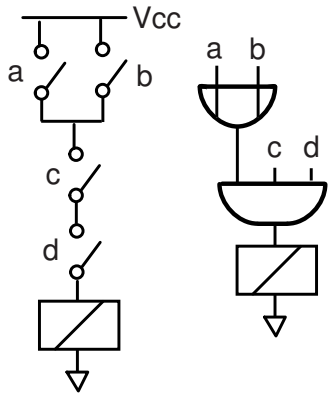


Abbildung 2: Ladder–Logic

MC14500B

Die SSI–Logik–Serie CD4000 wurde von einer Vielzahl von Firmen angeboten. Motorola wagte sich bald auch an komplexere ICs. Dort entstand von Vern Gregory die Idee, in dieser Technologie einen simplen 1–Bit–Mikroprozessor zu fertigen. Er zielte auf industrielle SPS ab. Speicherprogrammierbare Steuerungen (PLC: Programmable Logic Controller) wurden in Ladder–Logic, einem „Kontaktplan“, programmiert (Abb. 2). Diese „grafische Programmiersprache“ blieb zwar bis in die 80er in dem Bereich Standard, hat aber heute nur noch wenig Bedeutung. Motorola vermied ursprünglich den Namen „Mikroprozessor“ und bezeichnete das IC als „Industrial Control Unit“.

Das US–Patent [2] entspricht dem Produkt. Es konnte nicht verhindern, dass in Osteuropa Klone in CMOS bzw. I2L–Bipolar zumindest angekündigt wurden, die aber absehbar keine nennenswerte Verbreitung hatten.

Programmspeicher

Die CPU und die CD4000–ICs arbeiteten bei 3V... 18V. Aber Speicher gab es nur für die üblichen Mikroprozessoren mit 5V–Versorgung. In [1] wurden 512x8–Bipolar–PROMs vorgeschlagen. Zeitgleich wurden allerdings das

NMOS–2716–PROM von Intel angekündigt, das auch nur 5V benötigte. Beide Speichertypen hatten üppigen Stromverbrauch. Während die CPU nur $4,88/100 kostete, konnte man bei Windowed–CERDIP–EPROMs mit $25...$50 rechnen. Große PROMs waren absehbar nicht billiger. In SPS wurden EPROMs oft als steckbare Module (Abb. 3) ausgeführt. Aufwändig, aber eine reprogrammierbare Lösung, die Updates der Software erleichtert.

Abbildung 3: Speicherkarte Simatic S5

Weder elegant noch kompakt, war der MC14500 trotzdem kommerziell erfolgreich und bis in die 90er Jahre in Produktion. Alle Bauteile sind problemlos über `ebay.com` beschaffbar. Aber die CPU war dem allgemeinen Publikum unbekannt, weil sie kein konventioneller Mikroprozessor war. Sie hat in Deutschland trotzdem etwas Nachleben in Retro–Fanprojekten gefunden. Und das kam so:

WDR–1–Bit–Computer

Da der Mikroprozessor universell nützlich ist, interessierten sich bald auch die Funkamateure dafür („packet radio“). Burkhard John und Volker Ludwig vom „Jugend– und Ausbildungsreferat im DARC[1]“ (Abb. 7) wollten keine SPS bauen. Sie sahen die Möglichkeit, einen verständlichen Lerncomputer mit einer geringen Zahl an Bauteilen zusammenzustellen. Wenige ICs hatte auch ein KIM–1, aber LSIs erzwingen ein Denken in abstrakten Blockschaltbildern.

Sie hatten vorher an einer Volkshochschule in NRW einen Z80–Einplatinencomputer aufbauen lassen und schätzten, dass nur 10% der 50 Kursteilnehmer die Maschine verstanden hatten [3]. Der MC14500 arbeitete auf der Ebene von Flip–Flops und Gates. Er sollte und konnte von Schülern gebaut werden (Abb. 4). Diese Ausrichtung auf ein breites Publikum sagte dem Redakteur des Schulfernsehens des WDR zu. Die 6 Folgen von „Bit und Byte — Wir bauen einen Computer“ hatten einiges an Substanz, nicht typisch das, was man vom Fernsehen erwartet. Vom Löten der Leiterplatten bis zum Debugging wurde alles im Detail dargestellt. Als Anwendung steuerte der Computer allerlei Fischertechnik–Roboter (Abb. 5). Die Serie ist dem Publikum in Erinnerung geblieben. Die Folgen sind auf YouTube zu finden.

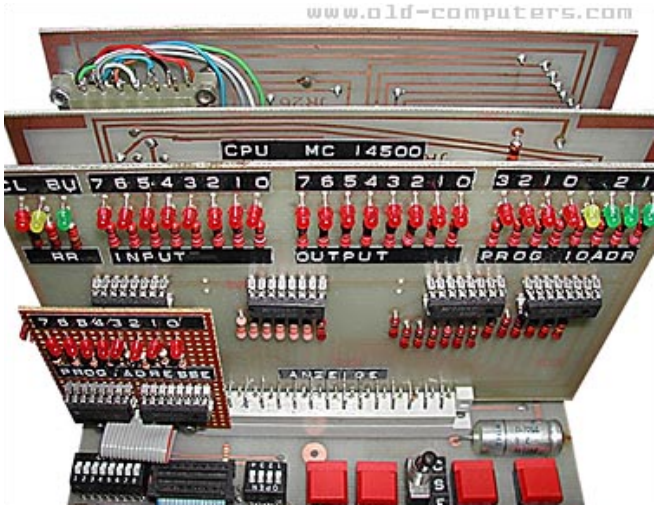


Abbildung 4: Lerncomputer

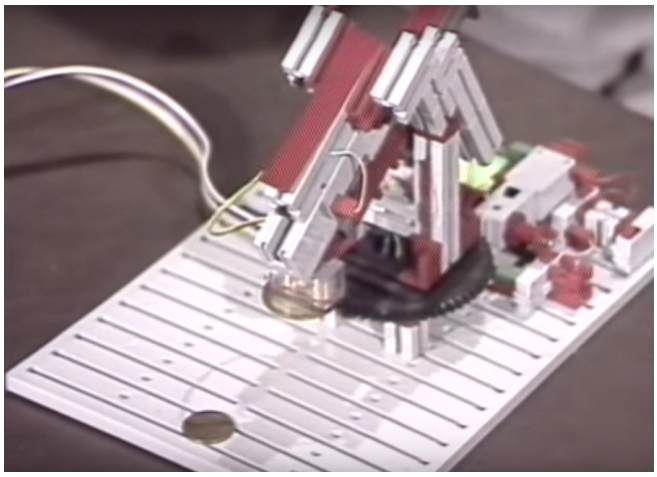


Abbildung 5: Roboter aus WDR „Bit und Byte“

General Instrument SBA

Der „Sequential Boolean Analyzer“ war ein 1–Bit–Stackprozessor, der 1977 angekündigt, aber absehbar nie produziert wurde.

Das NMOS–IC mit +12V– und +5V–Versorgung in einem 40–Pin–DIP–Gehäuse hatte 1 kByte ROM integriert. Preis: $4 ab 2500 Stück. Besonderheit: Die IO–Pegel waren 12V. Frühe NMOS–ICs mit hohen Versorgungsspannungen hatten meist Mühe, ihre Signale mit TTL–kompatiblem Pegel auszuführen. Das war hier explizit nicht erwünscht.

Für die Entwicklung war der Bond–Out–Chip SBA–1 vorgesehen (Tab. 1). Industrielle Steuerungen erreichen nicht

[1] Die *CQ DL* ist das Amateurfunkmagazin des Deutschen Amateur–Radio–Club (DARC) e.V. und erscheint zwölfmal im Jahr, 2020 im nun 91. Jahrgang.

die Stückzahlen für Masken–ROM–Fertigungen. Aber Relais werden auch in kleinen Nebenstellenanlagen (PBX) geschaltet. Es gab also einige Anwendungen mit Stückzahlen.

	RAM	ROM	I/O
SPA	120x1	1kx8	31
SPA-1	120x1	extern	31
SAP-2	120x1	2kx8	31

Tabelle 1: GI SBA

Trotz der Unterschiede im Signalpegel waren die direkte Konkurrenz konventionelle NMOS–Mikrocontroller in DIL40. Der 1976 von Intel angekündigte 8748 benötigte nur +5V, hatte aber etwas weniger I/O–Pins (Tab. 2). Für die Entwicklung gab es von Intel Windowed–CERDIP–EPROMs für satte $175 ab 100 Stück. Wie man aber feststellte, war selbst dieser Preis für Anwender kleiner Stückzahlen für den Serieneinsatz akzeptabel. Ein Einchip–Controller, bei dem die gesamte Schaltung auf eine Leiterplatte passte, war preiswerter als ein Mikroprozessorsystem aus mehreren Leiterplatten mit billigem 2716–EPROM.

	RAM	EPROM	I/O
8748	64x8	1kx8	27

Tabelle 2: Intel 8748 EPROM

Da Intel bald eine 2–kByte–Version lieferte, kündigte auch GI den SBA–2 an. Der schaffte es aber wohl nicht mal zum Datenblatt.

Architektur

Die IO war auf 30 Input– und 30 Output–Pins ausgelegt. DIL40 hat aber weniger Pins, die konkrete Zuordnung wurde in der Maske des ROMs festgelegt. Mit dem Befehl `RESTART` beginnt der Arbeitszyklus: Alle Inputpins werden in das Eingangs–Latch geladen. Gleichzeitig werden alle Outputs aktualisiert. Das Datenwort ist nur 1 Bit breit (Abb. 6). Neben dem 16–Bit–tiefen Stack gibt es 120 Bit RAM („stored state“). Da nur je 30 Bit direkt adressierbar sind, ist das RAM in 4 Bänke aufgeteilt. Der Programmspeicher ist recht konventionell: 1 k x 8 Bit. Angepasst an den Befehlssatz und etwas ungewohnt ist die oktale Notation (Tabelle 4). Im Adressbereich 00. . . 36 der jeweiligen Opcodes waren nur 01. . . 30 tatsächlich vorhanden.

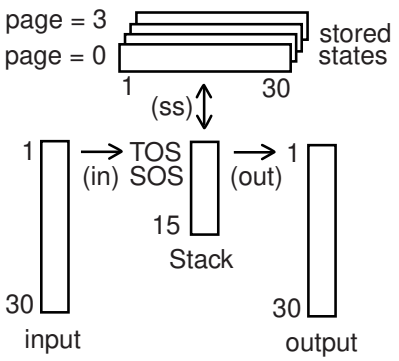


Abbildung 6: Datenspeicher

Es gab keine Arithmetik, nur die booleschen Befehle `AND OR XOR NOT` . Zähler sind nominell möglich, indem man sie im stored space praktisch auf Gate–Ebene simuliert — bezüglich des Verbrauchs an Programmspeicher nicht effizient und natürlich langsam. Es ist zwar ein *Stack–Prozessor*, aber aus den Mnemonics ergibt sich, dass die Entwickler nie mit *FORTH* in Berührung gekommen sind. Vorgesehen war ein SBA–Compiler, vermutlich in FORTRAN, der boolesche Gleichungen auflöste (Listing). Infix–Notation effizient in Postfix–Opcodes wandeln ist aber nicht unbedingt trivial. Zudem gab es einen SBA–Simulator.

```
1   ; Listing : 3 Bit Zähler
2
3   stack = A*STEP;
4   stack = B*(STEP.A);
5   stack = B*(STEP.A.B);
6   C = save;
7   B = save:
8   A = save;
```

Resümee

Die ungewöhnliche 1–Bit–Architektur war nur für die industriellen Benutzer von Ladder–Logic leidlich attraktiv. Die weniger leistungsfähige CPU von Motorola wurde zeitgleich angekündigt und war am Markt langfristig erfolgreich. Die höhere Integrationsdichte von NMOS gegenüber CMOS war kein entscheidender Vorteil. Trotz 12V–Versorgung bot NMOS nicht die gewünschte Störfestigkeit. CD4000–CMOS war Jahrzehnte später noch aktuell, als NMOS bereits obsolet war. Das Konzept von General Instrument orientierte sich zu sehr an konventionellen Mikroprozessoren, aber ohne leistungsfähig genug zu sein, um mit diesen konkurrieren zu können.

[1] Gregory, Bellande; „MC14500B Industrial Control Unit Handbook“; Motorola 1977

[2] US–Patent 761,738, 1977, Vernon Gregory, Motorola; „Industrial Control Processor“

[3] Volker Ludwig, Klaus Paschenda; „Fast alles über den WDR–1–Bit–Computer“

[4] Datenblatt „SBA Sequential Boolean Analyzer“ preliminary; ca. 1977

[5] Bursky; „microprocessor data manual“; Hayden 1978

Befehlssatz des MC14500B und des SBA im Vergleich.

			Puls	High		
0	0000	NOPP	RR <- RR	Flag	O	No Change
1	0001	LD	RR <- Data			Load Result Register
2	0010	LDC	RR <- /Data			Load Complement
3	0011	AND	RR <- Data AND RR			AND
4	0100	ANDC	RR <- /Data AND RR			AND Complement
5	0101	OR	RR <- Data OR RR			OR
6	0110	ORC	RR <- /Data OR RR			OR Complement
7	0111	XNOR	RR <- /(Data XOR RR)			XNOR
8	1000	STO	Pin <- RR	Write		Store
9	1001	STOC	Pin <- /RR	Write		Store Complement
A	1010	IEN	IEN <- Data			Input Enable
B	1011	OEN	OEN <- Data			Output Enable
C	1100	JMP		JMP	Flag	Jump
D	1101	RTN		RTN	Flag	Return and skip next instruction
E	1110	SKZ				skip next instruction if RR=0
F	1111	NOPF	RR <- RR	Flag	F	No Change

Tabelle 3: Befehlssatz MC14500B

octal	Mnemonic	Funktion
000	RESTART	TOS <- 1 ; output ; input ; page = 0 ; start
001	INVERT	SWAP
002	PAGE	page = page+1
003	HOME	page = 0
004	PUSH 0	TOS <- 0
005	PUSH 1	TOS <- 1
006	PUSH C	DUP
007	POP	DROP
010 ... 360	ANDIN	TOS <- (in) and TOS
011 ... 361	NANDI	TOS <- /((in) and TOS)
012 ... 362	ANDSS	TOS <- TOS and (ss)
013 ... 363	NANDSS	TOS <- TOS and (ss)
014 ... 364	ASPI	TOS <- TOS and SOS ; TOS <- 1
015 ... 365	STORE	(ss) <- TOS ; TOS <- 1
016 ... 366	NASPI	TOS <- /((ss) and TOS); TOS<-1
017 ... 367	OUTPUT	(out) <- TOS ; TOS <- 1
370	AND	TOS <- TOS and SOS
371	OR	TOS <- TOS or SOS
372	EXOR	TOS <- TOS xor SOS
373	COMP	TOS <- /TOS
374	PAND	TOS <- TOS and SOS ; TOS <- 1
375	POR	TOS <- TOS or SOS ; TOS <- 1
376	PEXOR	TOS <- TOS xor SOS ; TOS <- 1
377	PCOMP	TOS <- /TOS ; TOS <- 1

Tabelle 4: Befehlssatz SBA

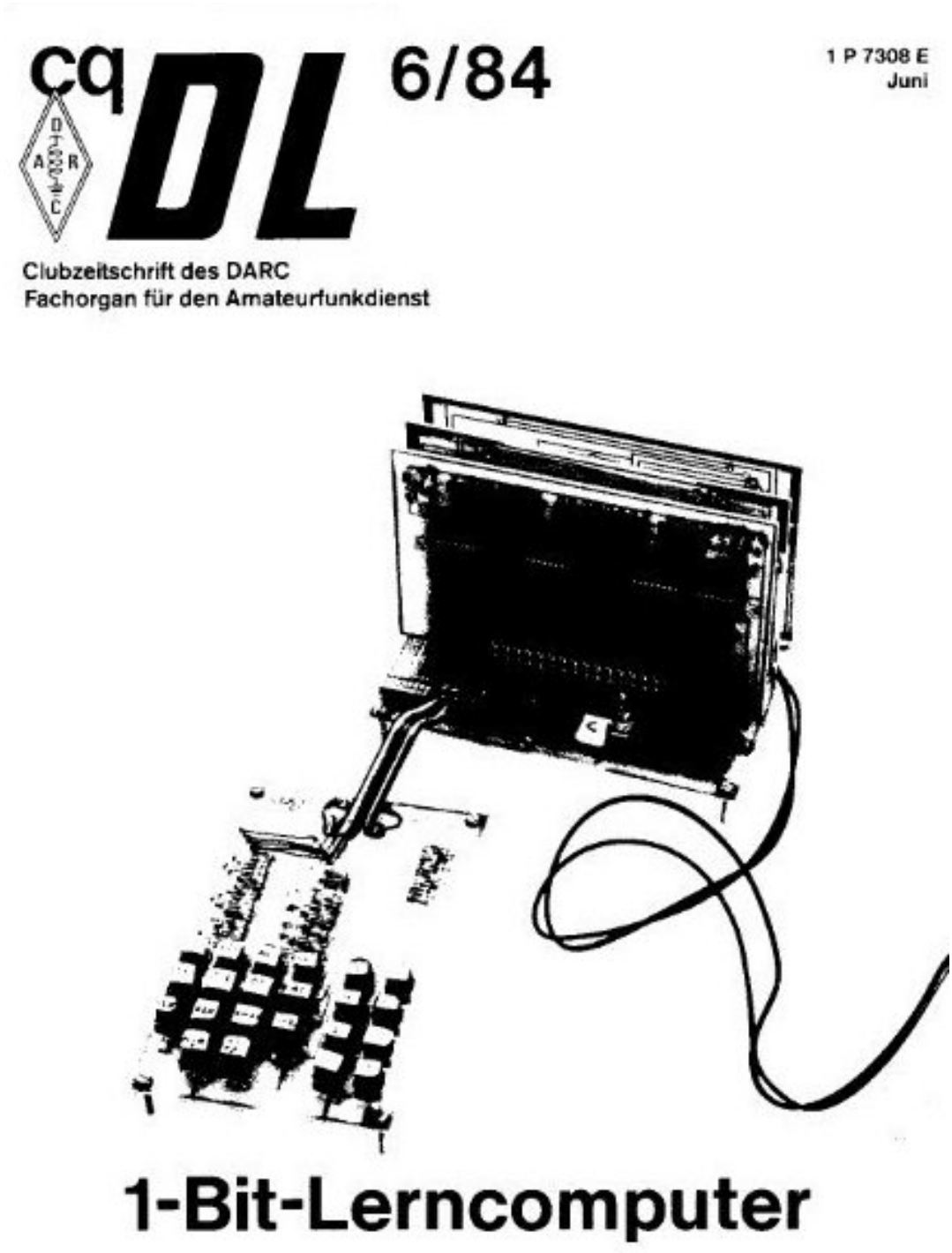


Abbildung 7: Cover cqDL

Swaps Zeit in der Mecrisp–Schmiede

Matthias Koch

Es war eine schöne Zeit mit Swap, der sich für ein Jahr sein Nest bei mir gebaut, viele Freunde gefunden (Abb. 1) und anscheinend auch zu jeder Jahreszeit etwas angeknabbert hat (Abb. 2, 3, 4 und 5). Alleine die charmanten Unarten des Drachens wären genug, um das ganze Heft zu füllen! Aber was genau ist nun eigentlich in der Zeit bei mir mit Forth passiert?

Abbildung 1: Swaps Lieblingsplatz inmitten von Gefährten aus allen möglichen Materialien.

Forth wird fliegen!

Erstmal ist etwas Verrücktes geschehen: Eigentlich bin ich ja formell Biophysiker und habe für meine Doktorarbeit die Lichtstressreaktion von Algen mit Laserspektroskopie untersucht. Als Swap im Frühling 2019 mit seinen Boten zu mir gekommen ist, war ich gerade damit fertig und mitten in den Bewerbungen auf eine neue Stelle. Am liebsten hätte ich mit *Forth* die Gewächshäuser unsicher gemacht, mit Lasern im Gepäck ungelöste Probleme der Botanik in Angriff genommen oder die Tiefen der Meere mit Tauchrobotern erkundet. Aber es kam anders.

Abbildung 2: Der Frühling ist da — und neugierig, wie Swap ist, wurden die duftenden Blüten sogleich angeknabbert.

Vierzehn vergebliche Bewerbungsgespräche sollten vergehen, in denen meine ganz besondere Kombination von Fächern immer wieder durch die Checkliste der formellen Anforderungen gerasselt ist. Da hat Swap mich eines Tages traurig angeguckt und mir einen guten Rat gegeben: „Matthias, Forth muss fliegen! Wie ich!“ So hat es mich beruflich im Oktober 2019 in die Raumfahrt verschlagen. Jetzt entwickele ich Firmware für FPGAs, die ein *Experiment mit eiskalten Gaswolken in optischen Fallen* steuern. Swap zwinkerte mir fröhlich zu, als er davon hörte. Und es war schon im Bewerbungsgespräch klar, dass *Forth* zum Einsatz kommen würde.

Abbildung 3: Auch wenn Kirschblüten gut duften, war ihm der Rosmarin im Sommer dann doch lieber.

Mecrisp–Ice wird flügge

Jetzt läuft Mecrisp–Ice in mehreren FPGAs des Experimentes, und aus dem kleinen Projekt, welches ich 2015 aus Neugier auf der Basis von Swapforth von James Bowman begann, hat sich in den letzten Monaten ein ernstzunehmendes Werkzeug entwickelt. Die Unterstützung für Double–Arithmetik wurde erweitert, Fixkommazahlen im s15.16–Format sind hinzugekommen und es gibt jetzt sogar wie in Mecrisp–Stellaris und Mecrisp–Quintus eine nachladbare Bibliothek mit mathematischen Funktionen wie Sinus, Arkustangens und Logarithmen. Außerdem habe ich die Grafikbibliothek vom MSP430 angepasst, so dass auch kleine Displays angesteuert werden können. Insgesamt steht Mecrisp–Ice mittlerweile den anderen Mitgliedern der Mecrisp–Familie in nichts mehr nach und ist dabei erstaunlich portabel geworden. Die größte Stärke von Mecrisp–Ice ist es jedoch, als Ausgangpunkt für eigene Entwicklungen in Logik zu dienen: Es ist nicht schwer, selbst Peripheriemodule in Mecrisp–Ice einzufügen.

Erst in Kombination mit eigener Logik zeigen sich die flexiblen Talente von FPGAs, die Mikrocontroller nicht bieten können!

Abbildung 4: Ob Swaps großer Appetit im Herbst für einen ganzen Kürbis reicht?

Und auch Mecrisp–Quintus für RISC–V mausert sich

In Mecrisp–Quintus ist das erste stabile Release in greifbare Nähe gerückt, der letzte noch offene Punkt ist die Behandlung von Interrupts, deren Implementierung bei RISC–V eine erstaunliche Vielfalt an Varianten und Formen erlaubt. Ansonsten ist das Forth bereits gut benutzbar, hat sich auf den frühen Entwicklerplatinen bewährt und ist seit dem Erscheinen der Portierung für den *GD32VF103* auf der Zielgeraden, wie das auf dieser Basis von Martin Bitter und Wolfgang Strauss in Angriff genommene „Projekt Feuerstein“ beweist. Auch Manfred Mahlow ist dabei und hat seine *Erweiterung für das Dictionary* von Mecrisp–Stellaris sogleich auf die freie Architektur portiert. Natürlich genießt RISC–V in diesen Tagen zu Recht ganz viel Aufmerksamkeit, aber auch MIPS wird von Mecrisp–Quintus unterstützt. Ich bin gespannt, ob es eines Tages auch eine Portierung auf die legendären Grafikrechner von Silicon Graphics geben wird: Das wäre natürlich auch genau die richtige Portierung, um der Forth–Assembly beim nächsten *Chaos Communication Congress* in Leipzig einen weiteren Hingucker zu bescheren! (Abb. 8)

Fehlersuche in Mecrisp–Stellaris

Während das Ur–Mecrisp für MSP430 größtenteils fertig zu sein scheint und schon lange keine Bugs mehr darin gefunden worden sind, gibt es in Mecrisp–Stellaris immer mal wieder Überraschungen. Im Februar erreichte mich ein Fehlerbericht aus Sofia, Bulgarien, der alle RA–Kerne betraf: Wenn `@` von einer konstanten Adresse bei einer bestimmten inneren Reihenfolge von Registern mit einem Vergleich zusammentraf, löste dies das Generieren von fehlerhaften Opcodes aus. Dieser Fehler ist seit den Anfangstagen des Registerallokators 2011 an Bord gewesen und wurde bislang einfach nicht entdeckt — jetzt ist er aber korrigiert. Ich kann nur alle ermuntern, die ein seltsames Verhalten in meinen Compilern finden: Versucht es einzugrenzen und lasst es mich wissen! Nur so kann ich die verrückten Fälle korrigieren, und es ist trotz aller Stabilität sehr wahrscheinlich, dass noch weitere Fehler darin schlummern.

Abbildung 5: Winterknabbereien aus Spekulatius … Futter oder Freund?

Drachenträger–Nachfolger

Eine ganz besondere Aufgabe eines jeden Drachenträgers ist es, Nachfolger vorzuschlagen. Für mich stand schon früh fest: Matthias Trute sollte Swap in diesem Jahr bekommen! Doch dann ist es leider ganz anders ausgegangen, als geplant.

Swaps Abreise

Als es für Swap Zeit wurde, die Reise zur diesjährigen Forth–Tagung anzutreten, hat ihn Michael Kalus abgeholt (Abb. 6). Und nun ist er dort bei ihm, da, wo seine Geschichte einmal begann. Und wo er nun ein Jahr Urlaub machen wird, weil die Tagung ja ausgefallen ist, wie ihr alle wisst.

Abbildung 6: Swaps letzter Tag bei mir, bevor er mit dem Drachenboten Michael (links) auf die Reise gegangen ist.

Vor Reiseantritt haben wir gleich mal Inventur der Schatzkiste gemacht. Ob wir verraten, was darin versteckt ist? Natürlich besteht der Drachenschatz aus Kristallen, vielen blanken Münzen und bunten Scheinen in allen möglichen aktuellen und historischen Währungen, aber es sind auch Dinge dabei, wo wir selbst nicht so genau wissen, was das ist (Abb. 7). Ein ganz besonderes Blatt, mittlerweile sehr trocken in mehrere Stücke zerbröselt, war dabei — ob Swap selbst einmal ein Gärtner gewesen ist? Wir sind neugierig herauszufinden, welcher der ehemaligen Drachenträger diese besondere Erinnerung hinzugefügt hat, und welche Geschichte sich dahinter verbirgt.

Bei mir sind zwei neue Länder zur Münzsammlung hinzugekommen — und für einen Forth–Drachen durfte natürlich auch ein Prozessor nicht fehlen! So kam ein kleiner, besonders handlicher *MSP430F2012* in den Schatz, den ich einst von Dirk Brühl geschenkt bekommen habe, und der die Entwicklung von *Mecrisp–Across* angestoßen hat. Eins haben dieser Chip und Swap gemeinsam: Forth gibt ihnen ihre Seele, aber sie enthalten nicht selbst Forth! In Swap steckt natürlich ein mutiges Drachenherz, und im Falle des kleinen Chips ist ein Crosscompiler am Werk, denn ein für den Menschen angenehmes Forth in 2 kB Flash zu quetschen — dieses Kunststück ist mir bislang nicht gelungen.

Ganz bestimmt werden Knabbereien für den Geist nie außer Mode kommen, doch wird die Zukunft für uns alle Überraschungen bereithalten. Denn eins steht unverrückbar fest — und das ist der Wandel!

Matthias (Dr.achenträger a. D.)

Abbildung 7: Inventur der Schatzkiste voller Reiseandenken des welterfahrenen Forth–Drachens

Abbildung 8: Nur der Wandel ist gewiss — wovon das Klavier beim 36C3 ein Lied spielen kann.

Das Forth–Wort Star–Slash und der Nutzen von Kettenbrüchen

Jens Storjohann

Ein Forth–Programmierer hat oft Gründe, auf den Einsatz von Gleitkommazahlen zu verzichten. Er nutzt stattdessen, wenn er eine ganzzahlige Größe mit einem echten oder unechtem Bruch multiplizieren will, das sehr nützliche Forth–Wort „star–slash“ ∗/, das eine ganzzahlige Multiplikation mit doppelt genauem Zwischenergebnis verbindet mit einer anschließenden Division, die dann ein einfach genaues Ergebnis liefert.

Einfache und komplizierte Fälle

Wenn Zähler und Nenner des Bruchs durch die Anwendung schon vorgegeben sind und beide in die Wortbreite passen, ist nicht viel mehr zu tun, als */ anzuwenden.

Wenn erst nach Kürzen des Bruchs Zähler und Nenner in die Wortbreite passen, kann man auch fortfahren.

Natürlich muss auch sichergestellt werden, dass das Endergebnis in der gegebenen Wortbreite darstellbar ist. Dies ist relevant beim Multiplizieren mit unechten Brüchen. Denn nur dort kann ein Überlauf eintreten.

Irrationale Zahlen

Wenn aber mit einer Zahl zu multiplizieren ist, die nicht exakt als Bruch dargestellt werden kann, wie z. B. die Quadratwurzel $\sqrt{2}$, muss man eine Wahl treffen, welcher Bruch dann ersatzweise verwendet wird.

Für die griechischen Mathematiker der Antike war es ein Schock, herauszufinden, dass etwas, was in der Geometrie einfach nur die Länge der Diagonale eines Quadrats mit Seitenlänge 1 war, nicht mit den algebraischen Hilfsmitteln der Zeit, nämlich als Bruch aus zwei ganzen Zahlen, ausgedrückt werden konnte.

In der Praxis — nicht in der Theorie — hat man ein ähnliches Problem, wenn Zähler und Nenner auch durch Kürzen nicht für die gegebene Wortbreite passend gemacht werden können.

Praktisches Einsetzen von Brüchen

Als erstes macht man für die Darstellung von $\sqrt{2} = 1,4142\ldots$ vielleicht eine Ersetzung wie:

$$\sqrt{2} \to \frac{14142}{10000}.$$

Aber so richtig traut man dieser Lösung, die auf dem Kappen eines Dezimalbruchs basiert, nicht zu, dass sie sich aus der Mathematik ergibt und optimal genau ist.

Klar: Das Dezimalsystem kommt daher, dass schon unsere Vorfahren mit ihren zehn Fingern zählen und rechnen konnten. Das ist aber kein Grund, sich in Computerprogrammen auf das Dezimalsystem zu stützen, denn die betrachtete Operation ist ja auch unabhängig vom verwendeten Zahlensystem(!).

Auch der Vorschlag, das Binärsystem zu wählen, weil es im Computer intern genutzt wird, ist nicht vernünftig, denn der Computer soll intern in irgendeinem System **richtig rechnen** und seine Ergebnisse in einer vom Programmierer gewünschten Weise darstellen.

Als Alternative fällt einem ein, dass man in der Schule schon gelernt hat, dass in der griechischen oder römischen Antike, deren Mathematiker **gar keine Stellenschreibweise, weder im Dezimalsystem noch einem anderen** nutzten, Näherungsbrüche für beispielsweise $\sqrt{2}$ oder π genutzt wurden:

Man verwendete zum Beispiel $\frac{22}{7}$ als Ersatz für π und hatte damit eine Näherung, die für viele praktische Zwecke ausreichte.

Man braucht Kettenbrüche

Wie gewinnt man solche sehr zweckmäßigen Brüche? Um diese Frage zu beantworten, muss man sich **mit Kettenbrüchen vertraut machen**. Das ist ein Gebiet, dass von Joseph Louis Lagrange und Leonhard Euler (nach Vorläufern [1]) systematisch bearbeitet wurde. Sie waren Zeitgenossen Katharinas der Zweiten von Russland und Friedrichs des Zweiten von Preußen.

Mit der dafür entwickelten Theorie weist man nach, dass Kettenbrüche optimale Approximationen liefern, wenn man sie vergleicht mit allen Brüchen, deren Nenner kleiner sind. Mathematische Feinheiten sind hier ausgelassen.

Kettenbrüche

Reguläre Kettenbrüche sind Brüche, deren Nenner, wie im folgenden Beispiel für eine Darstellung der **Euler–Zahl** e zu sehen ist, selbst wieder aus der Summe einer ganzzahligen Konstanten und einem Bruch mit Zähler 1 bestehen.

$$\mathrm{e} = 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{4 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{6 + \cdots}}}}}}}}$$

Kettenbrüche können unendlich sein wie obiges Beispiel, das die Euler–Zahl e darstellt, oder nach endlich vielen Gliedern abbrechen.

Das folgende Beispiel zeigt eine Weise, wie unendliche Kettenbrüche ins Spiel kommen.

Wir betrachten die Aufgabe, $\sqrt{2}$ näherungsweise zu berechnen und das Ergebnis als Bruch darzustellen. Aus

$$x^2 = 2 \quad \text{erhält man}$$
$$x^2 - 1 = 1$$
$$(x+1)(x-1) = 1$$
$$x = 1 + \frac{1}{1+x}$$
$$x = 1 + \frac{1}{1+1+\frac{1}{1+x}}$$
$$x = 1 + \frac{1}{2+\frac{1}{1+x}}$$

Damit erhält man durch wiederholtes Einsetzen

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2+\cdots}}}}}}}}$$

Diesen Kettenbruch können wir auf verschiedene Längen abschneiden und erhalten als Näherungswerte für $\sqrt{2} \approx 1.4142$ die Werte, die in Tabelle 1 dargestellt sind.

p	q	p/q	$(p/q - \sqrt{2})/\sqrt{2})$
1	1	1	-0.29289
3	2	1.5	0.060660
7	5	1.4	-0.010051
17	12	1.4167	0.0017346
41	29	1.4138	-0.0002.9731

Tabelle 1: Näherungsbrüche für $\sqrt{2}$ mit relativem Fehler

Die folgende Fehlerbetrachtung anhand der Tabelle 1 zeigt, dass man schon mit der fünften Näherung eine meistens ausreichende Genauigkeit erreicht.

$$\frac{(41/29 - \sqrt{2})}{\sqrt{2}} = -2.973e - 4$$

Eine für Elektrotechniker einfache Demonstration der Nützlichkeit von Kettenbrüchen geht über die Berechnung der Impedanzen von Abzweigschaltungen wie in Abb. 1 gezeigt. Diese Schaltung kann man sich auch entstanden denken als Spannungsteiler, an dessen Ausgang ein weiterer Spannungsteiler hängt, und so weiter endlich oder unendlich fortgesetzt.

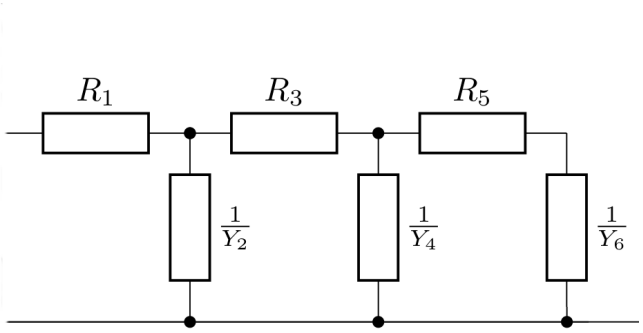


Abbildung 1: Abzweigschaltung

Kettenbrüche und der euklidische Algorithmus

Wir betrachten einen Bruch oder eine beliebige reelle Zahl größer als Null.

Das folgende Beispiel zeigt, wie eine solche Zahl mit Hilfe des euklidischen Algorithmus in einen Kettenbruch umgewandelt werden kann.

$$\frac{8}{5} = 1 + \frac{3}{5} = 1 + \frac{1}{\frac{5}{3}} = 1 + \frac{1}{1+\frac{2}{3}} =$$
$$1 + \frac{1}{1+\frac{1}{\frac{3}{2}}} =$$
$$1 + \frac{1}{1+\frac{1}{1+\frac{1}{2}}}$$

Die in Abbildung 1 gezeigte Schaltung ist direkt als Kettenbruch interpretierbar. Man erhält die folgende Liste von Näherungswerten $\hat{Z}_i$ für die Impedanz.

$$\hat{Z}_1 = Z_1$$
$$\hat{Z}_2 = Z_1 + \frac{1}{Y_2}$$
$$\hat{Z}_3 = Z_1 + \frac{1}{Y_2 + \frac{1}{Z_3}}$$
$$\hat{Z}_4 = Z_1 + \frac{1}{Y_2 + \frac{1}{Z_3 + \frac{1}{Y_4}}}$$
$$\hat{Z}_5 = Z_1 + \frac{1}{Y_2 + \frac{1}{Z_3 + \frac{1}{Y_4 + \frac{1}{Z_5}}}}$$

$$\hat{Z}_6 = Z_1 + \cfrac{1}{Y_2 + \cfrac{1}{Z_3 + \cfrac{1}{Y_4 + \cfrac{1}{Z_5 + \cfrac{1}{Y_6}}}}}$$

Man erkennt, dass die berechneten Näherungsbrüche für die Impedanz abwechselnd zu kleine und zu große Werte liefern. Außerdem schachteln die n–2–te und n–1–te Näherung die n–te Näherung ein, wenn der Kettenbruch konvergiert oder endlich ist. Konvergenz ist gegeben, wenn die Reihe der Teilnenner divergiert. Das ist insbesondere bei regulären Kettenbrüchen, deren Teilnenner natürliche Zahlen sind, garantiert.

Auswertung von Kettenbrüchen

Kettenbrüche taugen nur für sehr wenige Aufgaben. Insbesondere gibt es keine einfachen Algorithmen, zwei Kettenbrüche zu addieren oder zu multiplizieren. Dafür sind Dezimalbrüche handlicher.

Aber ihre Werte lassen sich effektiv berechnen, und die Erfahrung zeigt, dass Kettenbrüche schnell konvergieren.

Man kann Kettenbrüche von den höheren Indizes der Teilnenner beginnend „aufrippeln“ oder eine geschickte Rekursion beginnend bei den niedrigen Indizes verwenden.

Verfahren geeignet für die Gewinnung des Wertes eines Kettenbruchs als Gleitkommazahl

Wenn man zum Beispiel mit Hilfe des Taschenrechners einen Kettenbruch in eine Gleitkommazahl verwandeln will, kann man „von unten aufrippeln“.

Am Beispiel

$$Z = 2 + \cfrac{1}{3 + \cfrac{1}{5}}$$

erhält man das Ergebnis als Bruch durch ziemlich umständliche Rechnungen

$$5,\ 1/5,\ 3+1/5,\ 1/(3+1/5),\ 2+1/(3+1/5),\ 37/16$$

oder einfacher als ausgewertete Gleitkomma–Zahlen:

$$5,\ 0.2,\ 3.2,\ 0.312,\ 2.312$$

Rekursives Verfahren

Wir setzen Startwerte:

$$A_0 = c_0 \qquad A_{-1} = 1$$
$$B_0 = 1 \qquad B_{-1} = 0$$

und berechnen zwei dreigliedrige Rekursionsformeln

$$A_n = 1 * A_{n-1} + c_n * A_{n-2}$$
$$B_n = 1 * B_{n-1} + c_n * B_{n-2}$$

Dann ist die n–te Näherung für den Kettenbruch gegeben durch A_n/B_n. Man erhält also, wie für den Einsatz von `*/` gewünscht, Zähler und Nenner getrennt. Die Verschiedenheit der A_n von den B_n mit gleichem Index rührt nur von den verschiedenen Anfangswerten her. Die Rekursionsformeln sind die gleichen.

Wir haben eben den Kettenbruch

$$Z = [c_0; c_1, c_2] = [2; 3, 5]$$

behandelt. Dafür erhalten wir als Startwerte:

$$A_0 = 2 \quad A_{-1} = 1$$
$$B_0 = 1 \quad B_{-1} = 0$$

Die Rekursion liefert:

$$A_1 = 3 * 2 + 1 * 1 \qquad = 7$$
$$B_1 = 3 * 1 + 1 * 0 \qquad = 3$$
$$A_2 = 5 * 7 + 1 * 2 \qquad = 37$$
$$B_2 = 5 * 3 + 1 * 1 \qquad = 16$$

Weil wir uns einen endlichen Kettenbruch als Beispiel gewählt haben, erhalten wir im letzten Schritt mit $A_2/B_2 = 37/16$ den exakten Wert.

Welche Forth–Worte benötigen wir?

Gewinnung von Kettenbrüchen

Wir wollen beliebige Gleitkommazahlen (Floating Point Numbers) durch Kettenbrüche annähern. Zunächst müssen also Parameter (Teilnenner) bis zu einer vorgegebenen Ordnung k bestimmt werden. Dies geschieht in

```
r>cont_frac
( r,k -- c_0,c_1,...,c_k, k )
```

Auswertung von Kettenbrüchen

Dann muss der Wert dieses Kettenbruchs als Quotient zweier ganzer Zahlen A_n/B_n berechnet werden.

Die Berechnung des Wertes eines Kettenbruchs mithilfe des „Aufrippelns“ ist eher für Gleitkomma–Rechnungen geeignet und liefert dann als Ergebnis nur eine Gleitkommazahl. Das Verfahren ist zweckmäßig beim Einsatz eines Taschenrechners. Darum verzichten wir auf die Vorstellung eines dafür entwickelten Forth–Wortes.

Deshalb wählen wir ein Wort, das eine oben beschriebene **dreigliedrige Rekursion** durchführt,

```
cont_frac>A/B
( c_0, c_1,...,c_k, k --
 A_k-1 B_k-1 A_k, B_k )
```

als abschließend wichtigsten Baustein.

Schlussbemerkung

Weil ich die Forderung aufstellte, dass das Programmsystem im Prinzip (in den Grenzen der Hardware) für beliebig lange Sequenzen von Teilbrüchen geeignet ist,

die auf dem Stapel übergeben werden, brauchte ich `PICK` und `ROLL` . Sonst hätte ich die Stapel–Gymnastik mithilfe von *locals* vermeiden können. Das Jensen–Wirth–Pascal hätte das Problem mit einem Verbot gelöst: „Es gibt keine Sequenzen unbestimmter Länge als Parameter in Pascal, weil das gegen die Forderung des *strong typing* verstöße und damit ein schlechter Programmierstil wäre.“

Listing

```
: f>cont_frac
\ x,k -- c0,c1,...,ck, k
\ Real x wird näherungsweise in einen Kettenbruch
\ [c0;c1,...,ck]
\ mit 1 Absolutglied und k Teilnennern gewandelt
\ der Zähler k wird wieder auf TOS angehängt
  >r \ k sichern
  fdup floor fdup f>s f- \ c0 F:(x-floor(x))
                         \    R:k
  r@
  0 DO
   1/f fdup floor fdup f>s f- \ cn F:(y-floor(y))
                              \    R:k
  LOOP
  r> FDROP ;


\ Reihenfolge der Parameter muss umgdreht werden

: append_rev_sequ \ Sequenz drehen und angehängen
  ( n1 n2 ... nk k --
\   n1 n2 ... nk   nk ... n2 n1 k)
  DUP 1+
  0 DO \ k+1 mal iterieren
      i 2 * 1 + PICK SWAP
  LOOP ;

: cut_original_sequ \ Original Sequenz entfernen
  ( n0 n1 n2 ... nk nk ... n2 n1 n0 k --
\   nk ... n2 n1 n0 k )
  DUP 2 +
  1 DO   \ k+1-mal
     DUP  2 + ROLL DROP \ jeweils (k+2) ROLL DROP
   LOOP
   r> 1-
   ;

: reverse_sequ \ Sequenz
  ( n0 n1 n2 ... nk k -- nk ... n2 n1 n0 k )
  append_rev_sequ    cut_original_sequ ;

: step ( c A2 B2 A1 A2 -- c*A2+A1 c*B2+B1 A2 B2 )
\ Durchführung der dreigliedrigen Rekursion
  2OVER 2>r 2>r
  ROT DUP
  ROT *
  -ROT * SWAP
  2r>
  ROT +
  -ROT + SWAP
  2r>
;

: cont_frac>A/B ( c_k c_k-1 ... c_0 k
\                 -- A_k-1 B_k-1 A_k B_k )
\ Regulärer Kettenbruch wird in einen
\ einfachen Bruch gewandelt.
\ zwei Näherungen werden ausgegeben.
\ A_0=c_0  A_{-1}=1
\ B_0=1    B_{-1}=0
\ Rekursiv:
\ A_n = 1*A_{n-1} + c_n*A_{n-2}
\ B_n = 1*B_{n-1} + c_n*B_{n-2}
\
  >r \ k sichern
\ Startwerte auf den Stapel legen:
\ A0 B0 A{-1} B{-1}
 \ c0=A0 liegt schon oben
1 1 0
r> 0  \ k Schritte
DO  step  LOOP
2swap
;
```

Quellen

Eine sehr ausführliche Quelle ist die Wikipedia.

[1] `https://de.wikipedia.org/wiki/Kettenbruch`

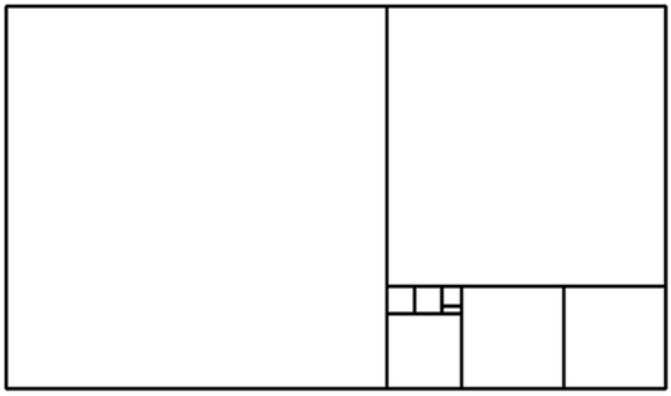


Abbildung 2: Ein Spiel, bei dem es darum geht, ein Rechteck in möglichst große Quadrate zu zerlegen, ist der Einstieg in die anschauliche Behandlung des euklidischen Algorithmus sowie die Darstellung von Zahlen als Kettenbrüche. (aus: Strick H.K., Mathematik ist schön. Springer, Berlin, Heidelberg 2019)

ST7735S am Longan Nano von Sipeed

Martin Bitter

Im Rahmen des Projektes Feuerstein *sahen wir uns den Mikrocontoller* GD32VF103CB *an. Dieser wird auf einem kleinen Board ausgeliefert, dem Longan Nano, das neben Anschlüssen für eine serielle Schnittstelle und für eine JTAG–Verbindung einen USB–Type–C–Stecker und zwei Taster (Boot, Reset) enthält. Das Augenfälligste für mich ist allerdings das nur 2,4 cm kleine 160x80–Pixel–Display. Es zeigt gestochen scharfe, leuchtkräfige, farbige Bilder. Das wollte ich unbedingt programmieren können. Glücklicherweise hat Matthias Koch sein Mecrisp schon auf RISC–V portiert. Sogar eine spezielle Version für den Longan Nano ist im Release [1].*

Also: Beispielcode in C überfliegen, Datenblätter lesen, programmieren. Letzteres, also das Programmieren, entwickelte sich zum Schreiben eines (fast) vollständigen Treibers.

Stichworte: GD32VF103CB, ST7735S, SPI, 3–Wire, Display, RISC–V, Mecrisp–Quintus

MCU GD32VF103CB

Das war die erste MCU, die wir in RISC–V–Technik bekommen konnten. Der Name GD32VF103CB entschlüsselt sich wie folgt: Hersteller ist GigaDevice (GD), er hat eine Zellgröße von 32 Bit (GD32), ist ein RISC–V–Prozessor (GD32V) und kopiert die Peripherie eines STM32F103 (GD32VF103). C und B geben Gehäusegröße und Speicherausbau an (GD32VF103CB). Wichtig hier: Das Peripheriedesign stützt sich auf die ARM–32–Bit–Architektur. Vieles (leider nicht alles), was für die Peripherie von ARM–32–Prozessoren gilt, findet beim Longan Nano ebenfalls Verwendung. Register entsprechen sich, ihre Namen im Datenblatt sind leider fast alle neu. ARM–Quelltext, z. B. aus Mecrisp–Stellaris–Quelltext, kann meist einfach genutzt werden. Man kann die ARM–Registernamen beibehalten (wir erschaffen und benennen sie ja selbst) und der Code wird funktionieren. Das kann aber zu Verwirrungen führen, wenn man im GD–Datenblatt nachliest. Ich fand es daher einfacher, die Registernamen auf die GD–Nomenklatur umzustellen [2]. Mit dem eigentlichen RISC–V–Kernel, der Bumblebee genannt wird, habe ich (noch) wenig zu tun.

Displaytreiber ST7735S

Angesteuert wird das Display durch den Treiberchip *ST7735S* zu dem es ein übersichtliches Datenblatt gibt [3]. Da das Display bzw. der Treiberchip auf dem Longan fest verdrahtet ist, bin ich an die damit verbundenen Möglichkeiten gebunden. Der ST7735S kann über parallele Protokolle (bis zu 18 Leitungen) oder seriell via SPI angesprochen werden. Für mich neu und ungewohnt ist die Möglichkeit, eine SPI–Schnittstelle mit nur drei Leitungen zu betreiben (Clock, Select und MISO/MOSI). Genau so haben es die Designer von Sipeed festgelegt. Während beim herkömmlichen SPI mit jedem gesendeten Byte (via MOSI) gleichzeitig ein Byte (via MISO) empfangen wird, müssen hier eventuelle Antwortbytes explizit abgeholt werden. Je nach Länge der erwarteten Antwort muss zwischen dem letzten Byte des an den ST7735S gesendeten Kommandos und dem Abholen der Antwort einmal mit der Clock–Leitung gewackelt werden. Zusätzlich zu den SPI–Leitungen gibt es noch einige andere, die den Kommandofluss oder besondere Funktionen (Tearingeffektbehandlung) steuern. Letztere ist ebenso wie die Displaybeleuchtung fest verdrahtet, d. h. hier: nicht beeinflussbar bzw. ungenutzt.

Gesteuert wird der ST7735S bzw. das Display, indem man Bytekommandos und darauf folgend Parameterbytes (im Extremfall bis zu 128!) über die SPI–Schnittstelle schickt. Die meisten Kommandos, 44 Stück, sind reine Schreibkommandos, d. h., der ST7735S sendet darauf keine Antwort. Die 11 Lesekommandos sind dagegen in der Unterzahl. Man kann ganz ohne Lesekommandos auskommen, wenn man eine gute Buchführung betreibt und sich merkt, wie man das Display eingestellt hat.

Ein Beispiel: Es gibt ein Kommando, mit dem man die Orientierung einstellt, also in welcher Ecke der Nullpunkt des Displays liegt. Man kann sich in einer Variablen merken, was man zum Display geschickt hat und so auf das Auslesen dieses Wertes verzichten. Hat man allerdings Lesefunktionen implementiert, kann man den aktuellen Wert auslesen. Will man das Display auslesen, also als Bild speichern, so kommt man um Lesefunktionen nicht herum. Das wollte ich haben!

Das SPI–Modul

Leitungen

Der GD32VF103CB verfügt über ein SPI/I2C–Modul, das zwei Schnittstellen versorgen kann. Der ST7735S ist an die Leitungen des SPI0–Moduls angeschlossen.

MOSI	GPIOA–P7
Clock	GPIOA–P5
Chipselect	GPIOB–P2

Der Kommandofluss (D/CX Parameter/Kommando) wird über GPIOB–P0 gesteuert. Ein RESET kann über GPIOB–P1 ausgelöst werden.

Geschwindigkeiten

Die Geschwindigkeit, mit der die Daten an den ST7735S geschickt werden, hängt von drei Gegebenheiten ab. Zum einen ist es die Taktfrequenz der MCU (ich habe hier

104 MHz verwendet), der Teilungsfaktor, mit dem das SPI–Modul gesteuert wird (/4) und die Möglichkeiten des ST7735S selbst. Der ST7735S empfängt bei diesen Einstellungen zuverlässig, d. h., alle Kommandos werden wie erwartet ausgeführt. Anders ist es, wenn vom ST7735S gelesen werden soll. Zuverlässige Daten bekomme ich hier nur, wenn ich den Teiler auf 16 erhöhe. Der größere Teiler bewirkt eine geringere Geschwindigkeit. Das liegt aber vielleicht daran, dass ich die Daten (noch) nicht schnell genug abholen kann. TODO: Assembler?

Modi

Das SPI–Modul der MCU kennt zehn verschiedene Modi. Interessant davon sind hier die beiden Modi MTB und MRB. Ausgeschrieben: „Master transmission with bidirectional connection“ und „Master reception with bidirectional connection“. Wie die Namen es sagen, gehen sowohl beim Schreiben als auch beim Lesen die Datenbytes über ein und dieselbe Leitung. Eingestellt werden Geschwindigkeit und Schreib–/Lesemodus im Register `SPI0_CTL0`.

Schreibbefehle an den ST7753S sehen in Pseudocode so aus:

```
Konfiguriere SPI0 (MTB 1/4 Takt);
setzte Pin D/CX auf 'Kommando' (high);
setze Pin CSX (chip select) auf low;
sende Kommandobyte;
setze Pin D/CX auf 'Parameter' (low);
sende Datenbytes.
```

`CSX` (chip select) kann „ewig“ auf low bleiben.

Beim Lesen:

```
Konfiguriere SPI0 (MTB 1/4 Takt);
setzte Pin D/CX auf 'Kommando' (high);
evtll.: setze Pin CSX (chip select) auf low;
sende Kommandobyte;
füge einen eventuellen Clocktakt ein;
Konfiguriere SPI0 (MRB 1/16 Takt);
lese N Daten;
setze Pin CSX (chip select) auf high;
```

Mit dem `CSX`–Signal (high) entscheidet man, dass der ST7735S mit dem Senden von Daten aufhören soll. Das ist notwendig, um eine neue Kommunikation zu starten.

Dummyclock

Um einen Dummyclocktakt zu senden, muss dem SPI–Modul kurzzeitig die Kontrolle über den Clockpin entrissen werden, eine aufwendige und vielleicht unnötige Angelegenheit. Ich kann auf das aktive Senden dieses Dummysignals verzichten, wenn ich statt der Anzahl der Antwortbytes ein Byte mehr lese, alle Bytes kombiniere und das Ergebnis entsprechend verschiebe und beschneide.

Ein Beispiel möge dies veranschaulichen.

Mit Dummyclock `x` drei Antwortbytes lesen:

```
Kommando Antwort
⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍
00111111x001101000111010001101000

( lesbarer: )
Kommando   Antwort
⎍⎍⎍⎍⎍⎍⎍⎍ ⎍ ⎍⎍⎍⎍⎍⎍⎍⎍ ⎍⎍⎍⎍⎍⎍⎍⎍ ⎍⎍⎍⎍⎍⎍⎍⎍
00111111 x 00110100 01110100 01101000
   '?'        '4'      't'      'h'

⎍=Clock,
  darunter die Bits,
  darunter das Zeichen.
  x=Dummyclock
```

Oder ohne Dummyclock vier Antwortbytes lesen:

```
Kommando Antwort
⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍⎍
00111111x001101000111010001101000xxxxxxx

( lesbarer: )
Kommando Antwort
⎍⎍⎍⎍⎍⎍⎍⎍ ⎍⎍⎍⎍⎍⎍⎍⎍ ⎍⎍⎍⎍⎍⎍⎍⎍ ⎍⎍⎍⎍⎍⎍⎍⎍ ⎍⎍⎍⎍⎍⎍⎍⎍
00111111 x0011010 00111010 00110100 0xxxxxxx
  '?'      $1A    ':' $3A  '4' $34  $40 +x

⎍=Clock,
  darunter die Bits,
  darunter das Zeichen.
  Jetzt x=0, aber ohne Bedeutung.
```

Die wahre Antwort wird so ermittelt:

```
kombiniere die vier Bytes ...
x0011010 00111010 00110100 0xxxxxxx
zu 32bit:
x001101000111010001101000xxxxxxx
dann: 1 lshift
001101000111010001101000xxxxxxx0
das 4. Byte weg:
001101000111010001101000
ergibt die gesuchten 3 Zeichen:
00110100 01110100 01101000
  '4'      't'      'h'
```

Überraschung

Mit dem Gesagten lassen sich alle Lesekommandos ausführen, die erhaltenen Werte sind, soweit überprüfbar, richtig. Eines hat mich zur Verzweiflung getrieben: Es gibt ein Kommando zum Auslesen des Displayspeichers. Zuvor übergibt man den Bereich, der ausgelesen werden soll und mit dem Kommando `RAMRD` ($2E) sollen dann alle darin liegenden Werte konsekutiv gelesen werden können. Sollen!

Mir wollte das in vielen Versuchen nicht gelingen. Ihr ahnt, das ich an allen möglichen und unmöglichen Schrauben gedreht habe. Erfolglos! Das Einzige, was mir gelang, war das jeweils erste Pixel des übergebenen Bereiches

auszulesen. Kurz bevor ich alles in die Tonne werfen wollte, kam mir die Idee, doch mal das schlaue Internet zu bemühen [4]. Und siehe da: Ich war nicht der einzige mit dem Problem, und andere hatten auch keine Lösung gefunden! Fazit: Die Doku übertreibt in diesem Punkt.

Jetzt kann ich mit ruhigem Gewissen laaaangsam den Speicher auslesen.

Quellcode

Der Quellcode ist Teil eines größeren Projektes. Er ist zu finden unter:

`https://wiki.forth-ev.de/doku.php/projects:feuerstein:ST7735S`

Quellen

[1] mecrisp–quintus–0.27a–experimental `https://sourceforge.net/projects/mecrisp/files/`

[2] GD32VF103_User_Manual_EN_V1.2.pdf `https://dl.sipeed.com/LONGAN/Nano/DOC/`

[3] driver chip ~ST7735S_V1.5_20150303.pdf~ `https://dl.sipeed.com/LONGAN/Nano/HDK`

[4] Adafruit sagt: “You can’t!” `https://forums.adafruit.com/viewtopic.php?f=47&t=24796`

[5] An anderer Stelle (Post #13) zeigt „Postman“, wie er ein Pixel mit 4–Drahtverbindung lesen kann. `https://www.avrfreaks.net/forum/st7735-graphics-chip-read-frame-buffer-memory-over-spi?page=all`

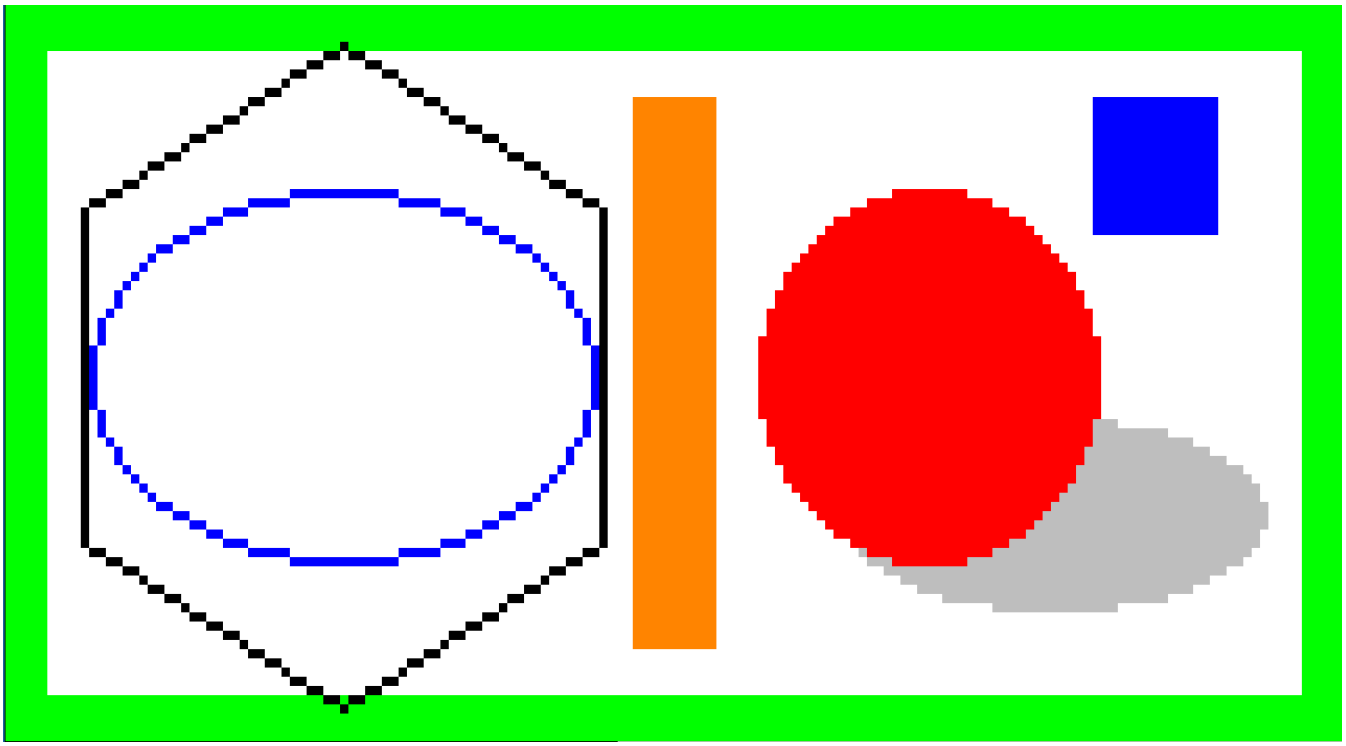


Abbildung 1: Ausgelesenes Bild

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@net2o.de`
Treffen: Jeden 4. Donnerstag im Monat um 19:00 in der Pizzeria La Capannina, Weitlstr. 142, 80995 München (Feldmochinger Anger).

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

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

Dienste der Forth–Gesellschaft

Jitsi `https://meet.forth-ev.de`

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

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

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

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

Spezielle Fachgebiete

Forth–Hardware in VHDL
microcore (uCore)
Klaus Schleisiek
Tel.: **(0 58 46) – 98 04 00 8**$_p$
kschleisiek@freenet.de

KI, Object Oriented Forth,
Sicherheitskritische Systeme
Ulrich Hoffmann
Tel.: **(0 41 03) – 80 48 41**
`uho@forth-ev.de`

Forth–Vertrieb
volksFORTH
ultraFORTH
RTX / FG / Super8
KK–FORTH
Ingenieurbüro
Klaus Kohl–Schöpe
Tel.: **(0 82 66)–36 09 862**$_p$

Termine

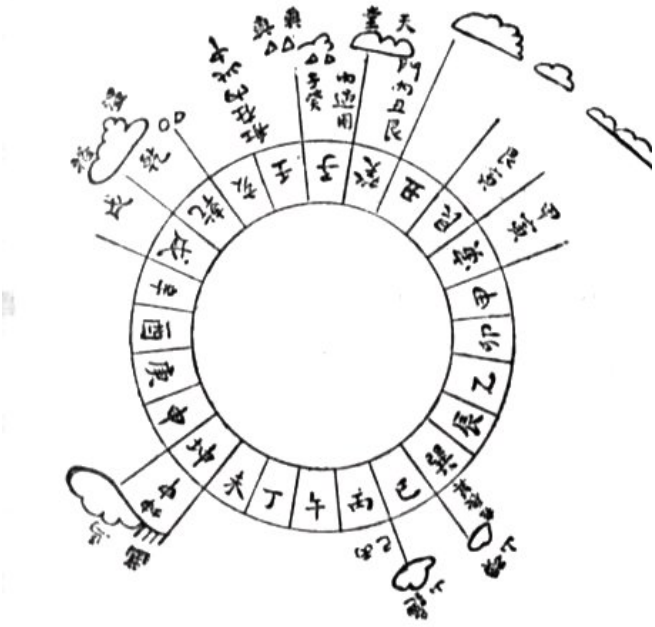
Donnerstags ab 20:00 Uhr
Forth–Chat net2o `forth@bernd` mit dem Key `keysearch kQusJ`, voller Key:
`kQusJzA;7*?t=uy@X}1GWr!+0qqp_Cn176t4(dQ*`

26.–29.03.2020
+++ verschoben +++
Forth–Tagung in Klein–Glien
`https://tagung.forth-ev.de`

12.–13.09.2020
+++ abgesagt +++
Maker Faire Hannover, HCC
`http://www.makerfairehannover.com`

27.–30.12.2020
+++ wenn wir alle geimpft+gechippt sind +++
37c3, Messe Leipzig

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.

Forth–Tagung in Klein–Glien (bei Bad Belzig)

Carsten Strotmann

Die Tagung im Jahr 2020 führt uns nun, nach der Corona–Krise, dennoch in das CocoNAT Workation Retreat *in Klein–Glien bei Bad Belzig (ca. 1 Stunde südlich von Berlin). Das CocoNAT — Klein Glien 25 — 14806 Bad Belzig — ist ein umgebauter Gutshof und liegt am internationalen Kunstwanderweg „Hoher Fläming". In der Umgebung gibt es viel Natur und Kultur zu entdecken und Berlin ist nicht weit.*

Anmeldung:

Die Anmeldung ist derzeit geschlossen.
Informationen zum neuen Termin gibt es unter:
`https://tagung.forth-ev.de`.
Dort wird, sobald der neue Termin feststeht, die Anmeldung möglich sein. Alle, die sich schon für den Termin im März angemeldet hatten, werden vom Orga–Team angeschrieben und müssen sich nicht nochmal anmelden.
Informationen über die Tagungsstätte CocoNAT:
`http://coconat-space.com/de/`