Vierte Dimension

4/2023

Das Forth–Magazin

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

In dieser Ausgabe:

Bild gelöscht

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.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

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 PD–Versionen). 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

Inhaltsverzeichnis

Titelbild Wan Character Knot Symbol für die Sammlung glückverheißender Ideen und Tugenden. mk
Quelle https://www.newhanfu.com/47393.html

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 1030
48481 Neuenkirchen
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 e + Porto u. Verpackung

Liebe Leser,

Ist Forth eine moderne Knotenschrift? Knoten in Schnüren wurden von manchen Kulturen schon benutzt, um Botschaften auszutauschen und Dinge zu zählen. Die chinesische Kunst der Knoten bildet Weisheiten ab, symbolisch. Wie ihre Schrift.

Wortverknüpfungen im Forth, so mysteriös wie die *Khipu* Südamerikas, bilden lange Stränge von aufgefädelten Nebenschnüren, bunten Quasten, Troddeln und Kartuschen: colorForth. Howerd Oakford konnte damit, wie C. Moore selbst, spielend ungehen. Ob unsere Nachfahren mal da stehen werden und rätseln, was es damit auf sich hatte?

Ganz moderne chinesische Knoten werden derzeit entwickelt, da, wo der Lange Fluss ins Ostchinesische Meer mündet. *Espressif* schafft dort in rascher Folge funkende MCUs für das Internet der Dinge; 1,27–mm–Pitch–Raster, *castellated holes* am Rand, wodurch wir immer einen Adapter für die praktischen *Lochrasterbretter* brauchen, die hier im Umlauf sind, erfunden von Robert Noyce in den 1940er Jahren, im 0,1–Inch–Raster jener Zeit, was man damals für noch so gerade eben handhabbar hielt — wie wahr.

Schaut mal in die Leserbriefe. *ESP32* und Forth, das klang in unseren Heften schon an, so auch nun. Auch zu *noForth T* und überhaupt *RISC–V* gibts da was. Und zu *VIS*. Und dann gehts mit Buntem weiter im Heft.

Ohne Editoren kein Forth. Klaus Kohl–Schöpe stellt nun seinen neuen mcFEdWin.f und GforthEdWin.f zur Verfügung. Erprobt ihn. Rückmeldungen sind willkommen.

An unserem Langen Fluss, am Niederrhein, befasst sich Wolfgang Strauß damit, den Kernal der archaischen Hardware des *My4TH* ladbar zu machen, damit es auch wirklich MY–Forth sein kann, also ganz in der eigenen Hand liegt: Das EPROM muss simuliert werden.

Ans entgegengesetzte Ende, in industrielle Anwendungen, bringt uns Rafael Deliano aus dem Quellgebiet unseres anderen langen Flusses; die Isar speist die Donau: Dort ist er unermüdlich befasst mit robusten Testgeräten für Komponenten des Maschinenbaus, *Stromschleifen* inklusive.

Philip Zembrod, hauptberuflich befasst mit ganz großer Software, beweist schließlich, dass es schon zu Zeiten des *Volksforth* gut möglich war, Laufzeitoptimierungen mit Bordmitteln auszuführen — also mit dem Forth selbst, das es zu optimieren gilt.

Nun, abschließend hats diesmal einiges hoffentlich Unterhaltsames, als Zeitvertreib am warmen Ofen und unterm Weihnachtsbaum.

Besonders: *Wie kamst denn du zu Forth?* Rafael Deliano legt vor, und ihr schickt mir hoffentlich auch all eure Geschichten darüber, so nach und nach.

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-2023-04

Die Forth–Gesellschaft e. V. wird durch ihr Direktorium vertreten:
Ulrich Hoffmann Kontakt: Direktorium@Forth-ev.de
Bernd Paysan
Gerald Wodni

Howerd Oakford tödlich verunglückt

Mit großer Trauer und Bestürzung müssen wir zur Kenntnis nehmen, dass Howerd Oakford vollkommen überraschend am *16. November 2023* an den Folgen eines schweren Fahrrad–Verkehrsunfalls ums Leben gekommen ist. Mit Howerd verlieren wir einen phantastischen Menschen, der uns stets freundlich und gewitzt britisch–humorvoll begegnete. Wir vermissen Howerd sehr.

Howerd war Forth–Enthusiast erster Stunde: Er programmierte — selbst knapp 25 Jahre alt — bereits Ende der 1970er in industriellen Anwendungen Mikroprozessoren mit Forth (CDP1802 mit microForth). Seit 1981 arbeitete er in zahlreichen Projekten als selbstständiger Berater für Software– und Hardwareentwicklung und Systemdesign und gründete 1985 seine Firma Inventio Software. Howerd ist dem Forth–Gedanken der einfachsten Lösung immer treu geblieben und hat Forth–Ideen ohne Unterlass in seine Projekte eingebracht und sie zum Erfolg geführt, auch wenn sie ganz andere Werkzeuge vorsahen und einsetzten.

Howerd war eine der tragenden Säulen der EuroForth–Konferenz und hat dort regelmäßig inspirierend über seine Forth–Entwicklungen gesprochen. Seine besondere Aufmerksamkeit hat er der Weiterentwicklung von Charles Moores colorForth gewidmet, dem er in jährlichen Zyklen spannende neue Möglicheiten verliehen hat. Howerd war erst kürzlich von Wolfsburg nach Köln gezogen, um dort neuen Herausforderungen zu begegnen. Dazu wird es nun nicht mehr kommen. Howerd, mag Forth mit Dir sein.

Im Namen der Forth–Gesellschaft, Ulrich Hoffmann

Forth 200X Treffen beider EuroForth 2023

Dieses mal haben wir keine neuen Wörter und Merkmale beschlossen, wer den Standard also nur konsumiert, kann diesen Artikel überspringen.

Prozess

Wer dagegen auch etwas dazu beitragen will, für den gibt es einige Neuerungen. Das Wichtigste ist, dass wir langsam wieder soweit kommen, dass die Forth–Gemeinschaft außerhalb des Komitees nicht nur über Vorschläge (*proposals*) und Kommentare dazu den Standardisierungsprozess beeinflussen kann, sondern dass es auch wieder eine Umfrage (CfV, benannt nach den *call–for–votes* im Usenet) gibt, wie die aktuelle Praxis und die Absichten bezüglich des Vorschlags sind.

Diese Umfrage dient einerseits als Entscheidungshilfe für das Standardisierungskomitee, andererseits aber auch zur Information von Programmierern und Systemimplementierern, in welchem System das im Vorschlag beschriebene Merkmale vorhanden ist und wie weit verbreitet die Nutzung dieses Merkmals, oder zumindest der Wunsch danach ist; das ist auch bei standardisierten Merkmalen relevant: Einerseits sind die meisten davon im Standard optional, auch wenn die meisten populäreren Systeme praktisch alles implementieren. Andererseits kann die Implementierung in einem System irgendwann erfolgen: bevor es einen Vorschlag gibt, während der CfV–Stufe, wenn das Kommittee den Vorschlag angenommen hat, oder auch erst Jahre, nachdem der Vorschlag in einem Release des Standards veröffentlicht wurde.

Solche Umfragen hatte es im alten E–Mail–basierten System gegeben, aber waren beim Umstieg auf das Web 2.0 lange nur auf der ToDo–Liste. Im Moment braucht das CfV–System noch etwas Arbeit, aber für Umfragen zu vorgeschlagenen zusätzlichen Forth–Wörtern ist es schon tauglich. Allerdings gibt es derzeit nur zwei CfVs, einer zum Üben und Testen, und einen, bei dem die Antwortmöglichkeiten nicht passen. Allerdings sollen zwei weitere Vorschläge in absehbarer Zeit die CfV–Stufe erreichen. Man kann sich die aktuellen CfVs auf https://forth-standard.org/proposals?state=voting anschauen.

Sonstiges

Den Großteil der Zeit verbrachten wir damit, alle Vorschläge[1] zu sichten, die sich seit 2016 angesammelt haben, und die in irgendeiner Form abzuarbeiten. Dafür gibt es jetzt einen Zustand *considered* für Vorschläge, mit denen sich das Komitee seit der letzten Änderung auseinandergesetzt und eine Rückmeldung gegeben hat.

Des Weiteren gab es Workshops über Recognizer und über Speicherzugriffe mit bestimmten Größen und bestimmter Byte–Anordnung. Aber da in diesen Fällen noch alles im Fluss ist, berichte ich an dieser Stelle nichts Näheres dazu, sondern erst, wenn diese Themen die CfV–Stufe erreicht haben. Wer sich für den aktuellen Stand interessiert, kann bei den Recognizern auf https://forth-standard.org/proposals/minimalistic-core-api-for-recognizers?hideDiff#reply-1081 nachschauen. Für die Speicherzugriffe gibt es noch keinen ausgearbeiteten Vorschlag.

Anton Ertl

NEXT für RISC–V

STC

All meine Forthe aus der Mecrisp–Familie haben *native Compiler,* sind also eng mit „subroutine threaded" verwandt. Das NEXT ist dort einfach nur ein ret oder eben ein Rücksprung über das nach Bedarf auf dem Returnstack gesicherte Linkregister.

Wie ein STC–Forth auf RISC–V funktionieren kann, mögen vier Beispiele veranschaulichen.

```
see 1+
00018FC8: 0405 c.addi x8, 1
00018FCA: 8082 c.jr x1
ok.
```

Primitiven im Kern nutzen — wenn möglich — direkt das Linkregister x1 für den Rücksprung.

[1] https://forth-standard.org/proposals

```
: gray ( u -- x ) dup 1 rshift xor ; ok.
see gray
00114932: 5713 srli x14, x8, 1
00114934: 0014
00114936: 8C39 c.xor x8, x14
00114938: 8082 c.jr x1
ok.
```

In einer einfachen Definition, die sonst nichts anspringt und auch in Forth keine Bewegungen auf dem Returnstack benötigt, kann der Rücksprung ebenfalls direkt über das Linkregister erfolgen, was wie bei Primitiven genau eine Instruktion erfordert.

```
: zeilenumbruch ( -- ) cr ; ok.
see zeilenumbruch
001149AA: 1171 c.addi x2, -4
001149AC: C006 c.swsp x1, 0 (x2)
001149AE: 17B7 lui x15, 00011000
001149B0: 0001
001149B2: 80E7 jalr x1, -3DC (x15) --> cr
001149B4: C247
001149B6: 4082 c.lwsp x1, 0 (x2)
001149B8: 0111 c.addi x2, 4
001149BA: 8082 c.jr x1
ok.
```

Bei einer Definition, die andere Definitionen anspringt, muss das Linkregister auf dem Returnstack gesichert werden, so dass insgesamt fünf Instruktionen nötig werden; wobei x1 das Linkregister und x2 der Returnstackpointer ist. Die Kombination aus lui und jalr schließlich ermöglicht es, jede beliebige Stelle im 32–Bit–Adressraum mit zwei Instruktionen zu erreichen. Möglich wäre dafür genauso die gleich lange Kombination aus auipc und jalr.

```
: absatz ( -- ) zeilenumbruch zeilenumbruch ;
ok.
see absatz
001149CC: 1171 c.addi x2, -4
001149CE: C006 c.swsp x1, 0 (x2)
001149D0: 3FE9 c.jal 001149AA --> zeilenumbruch
001149D2: 3FE1 c.jal 001149AA --> zeilenumbruch
001149D4: 4082 c.lwsp x1, 0 (x2)
001149D6: 0111 c.addi x2, 4
001149D8: 8082 c.jr x1
ok.
```

Sind die Unterdefinitionen ganz in der Nähe, können diese auch mit je nur einer einzigen Instruktion angesprungen werden.

ITC

Hier findest Du übrigens ein ITC–Forth für RISC–V, welches mit einem traditionellen NEXT ausgestattet ist:

https://github.com/aw/fiveforths

https://github.com/aw/fiveforths/blob/master/src/02-macros.s#L5-L11

```
# jump to the next subroutine (ITC),
#     appended to each primitive
.macro NEXT
lw a0, 0(s1)        # load memory addr from IP into W
addi s1, s1, CELL       # increment IP by CELL size
lw t0, 0(a0)        # load address from W into temp
jr t0               # jump to the address in temp
.endm
```

Damit hätten wir vier Instruktionen für eine ITC–Primitive im Vergleich zu einer einzigen für STC. Bei den „normalen“ Definitionen, also nicht bei den in Assembler geschriebenen Primitiven, kommt jedoch in einem ITC–Forth noch DOCOL dazu:

https://github.com/aw/fiveforths/blob/master/src/08-forth-primitives.s#L191-L194

```
docol:
PUSHRSP s1 # push IP onto the return stack
addi s1, a0, CELL # skip code field in W
                  #     by adding 1 CELL,
                  #     store it in IP
NEXT
```

Da auf RISC–V „push“ und „pop“ jeweils aus den beiden Instruktionen addi/sw oder lw/addi zusammengesetzt werden müssen, wird der Rahmen für Colon–Definition in der ITC–Implementierung damit länger als der Rahmen aus fünf Instruktionen, der bei STC nötig ist. Dafür benötigt das Anspringen einer Definition an beliebiger Stelle im 32–Bit–Adressraum bei ITC nur 4 Bytes, während die zwei Instruktionen bei STC, die nötig sind, um eine beliebige, weit entfernte Stelle zu erreichen, bis zu 8 Bytes lang sein können.

Matthias Koch

VIS RP2040 GPIO für Mecrisp Stellaris

Aktuell hat mir die VD 3/2023 [2] (Herzlichen Dank für Dein unermüdliches Engagement) den Anstoß gegeben, endlich mal ein *Raspberry Pi Pico Board* mit *Mecrisp Stellaris + VIS* in Betrieb zu nehmen. Das ist besonders interessant, da es sehr lange verfügbar sein wird.

Erstes Ergebnis, ein VIS RP2040 GPIO Modul für Mecrisp Stellaris. Die Vielzahl der GPIO Registerbezeichner (99!) des rp2040-datasheet.pdf reduziert sich damit auf ganze 6, über die alle 30 GPIOs mit Hilfe ihrer GPIO–(PAD–)Nummer (GPx) sehr einfach konfiguriert und gesteuert werden können, wie z. B. die LED (GP25) des Pi Pico:

```
25 GPIO CTRL        \ reset defaults used
25 GPIO PAD         \ reset defaults used
25 GPIO STATUS      \ not used here

25 GPIO OE    bis ( GPx -- ) \ pin enable
25 GPIO OUT bis ( GPx -- ) \ pin set, LED on
25 GPIO OUT bic ( GPx -- ) \ pin clr, LED off
25 GPIO OUT bix ( GPx -- ) \ pin xor, LED toggle
25 GPIO IN    bi@ ( GPx -- <>0|=0 ) \ LED on|off
```

[2] 4d2023–03 : https://wiki.forth-ev.de/doku.php/vd-archiv

Müssen mehrere GPIOs gleichzeitig gesetzt oder gelesen werden, ist auch das durch direkten SIO–Aufruf möglich:

```
SIO GPIO OE|OUT bis|bic|bix ( mask -- )
SIO GPIO OE|OUT|IN bi@      ( mask -- x )
```

Das alles wäre Stoff für ein VIS HOWTO. Mal sehen, wann ich dazu komme.

Bleibt noch anzumerken, dass das GPIO-Modul nicht einfach nur „syntactic sugar“ ist. Definiert man einen GPIO als Context Switching Item, z. B.

25 GPIO item constant LED

oder kürzer

25 GPIO pad: LED

kann man damit auf alle (seine) GPIO–Register zugreifen, z. B.

```
LED OE bis                  \ LED output enable
LED OUT bis|bic|bix         \ LED on|off|toggle
LED IN bi@ ( -- x <>0|=0 ) \ LED read; on|off
```

Ändert sich später die zu verwendende GPIO–Nummer, muss diese nur in der GPIO–Definition geändert werden, nicht aber im Code, der den GPIO verwendet.

Das Modul kann man im Wiki des Forth–ev finden im Archiv vis-mecrisp-stellaris.tar.gz .

Manfred Mahlow

https://forth-ev.de/wiki/ und dort nach projects:forth-namespaces:start surfen.

RP2040 — noForth T Manages the Internal Hardware

These examples will show you how to use the RP2040’s internal hardware. I’ve put them on a GitHub page to make them much easier for me and others to update. Hopefully it leads to something nice. :)

Kind regards, Willem
https://github.com/WillemOuwerkerk/noForth-T-hardware-examples-RP2040

ESP32Forth — (m)ein steiler Weg

Zwar bin ich noch nicht ganz „eingefuchst“,sondern bildlich gesprochen, auf dem Weg zur Zugspitze erst in Garmisch–Partenkirchen. Doch interessierts vielleicht den einen oder anderen.

Das aktuelle Projekt ist eingebettet in eine längere Geschichte. Es handelt sich um ein Winkelmessgerät, wie es heute eigentlich in jedem besseren Handy existiert oder auch separat für circa 20e zu kaufen ist. Es soll eben nur etwas genauer funktionieren. Und so werden gleich 12 bis 16 Accelerometer–Bausteine an das Teleskop montiert, damit die Blickrichtung des Teleskops so in etwa bestimmt werden kann (Abb. 1).

Ulrich Hoffmann hat mir wertvolle Startinformationen gegeben. Insgesamt war es ein langer und harter Weg bis zum jetzigen Resultat mit dem *ESP32forth v7.0.7.5 extended*.

Die Maschine *ESP–WROOM–32* ist schon schön schnell (meldet sich beim Booten als *ESP32–D0WDQ5)*. Hier mein Vergleich für die Ausführungszeit einer leeren do ... loop über 4 Mio. Wiederholungen: Novix–Prozessor (Ch. Moore) vor ca. 40 Jahren : 1 Sekunde. ESP32–Forth ca. 8 Sekunden. Dennoch: Hier hat Doktor Ting ein fantastisches Werk hinterlassen. In diesem Zusammenhang auch meinen ganz persönlichen Dank an Brad Nelson für seine gewaltige Leistung dazu.

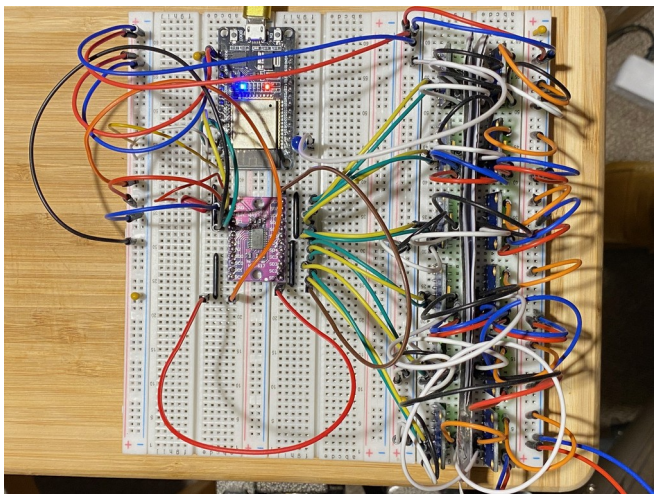


Abbildung 1: Zwölf Inertialsensoren, ein I²C–Multiplexer und der Prozessor.

Witzig: Es ist ein kleiner Satelliten–Prozessor im System eingebettet. Ich vermute, dass sogar für diesen ein eigenständiges Forth erarbeitet werden könnte. Aber das bleibt lediglich eine Vermutung.

Aktuell laufen bei mir zwölf Sensorbausteine mit je sieben Sensoren — drei Drehgeschwindigkeits–Sensoren, drei Accelerometer und noch ein Temperatur–Sensor. So viele Einzelsensoren habe ich seit meiner Studienzeit nie mehr abgefragt. Mit I²C geht es gut und schnell. Im Moment komme ich auf etwa 1500 Abfragen pro Sekunde a sieben Sensorwerte. Davon verwerte ich aber nur die Accelerometer–Werte (im Oktober).

PS: Ach ja, wichtig zu erwähnen ist in jedem Falle das Programm *CoolTerm*, welches ich als Terminal–Programm dafür verwende. Es passt sehr gut zum ESP32 und läuft sowohl auf Windows wie auf Mac–Systemen.

PPS: Vor einigen Monaten habe ich ein Forth–Happening auf meinem aktuellen Mac–Mini veranstaltet. Es liefen parallel etwa zwölf Instanzen vom *Gforth* plus 12 Instanzen von per USB angeschlossenen ESP32–Maschinen. Just for fun.

PPPS: (Inzwischen ist es November) Ich würde schon gern eher ein Artikelchen schreiben mit Bezug auf interessante oder wichtige Details des Projekts, allein es fehlt an Freizeit. Darin würde zum Beispiel ein Hinweis stehen, warum das Wort vlist und insbesondere see (kannte ich noch nicht) wichtig ist. Eventuell auch eine Auflistung der Extravokabularien, die man per words nicht als solche

erkennt, und wie man diese gegebenenfalls in der Arduino-Entwicklungsumgebung näher inspizieren kann — aber das ist nun erledigt dank Petremanns „The great book for ESP32forth“, S. 76, internals voclist.

Der für die Praxis wichtigste Tipp wäre der Hinweis auf das Wort echo (ist im Grundvokabular enthalten). Man kann damit die serielle Rückmeldung des ESP32 zum Terminal–Programm unterdrücken oder einschalten. Das ist deswegen so besonders wichtig, weil es sonst bei der Übermittlung eines Forthprogrammes zu Kompilationsfehlern kommt. Die Fehlersuche kann sich dann mühsam gestalten.Das Gleiche gilt für fehlende oder unpassende Steuerzeichen (zum Beispiel CRLF)[3], welche zu *nicht gemeldeten Kompilationsfehlern* führen[4]. Die Lernkurve war für mich an dieser Stelle etwas „ungewöhnlich“. Inzwischen hat Marc Petremann auch ein *Glossar* online, in dem man nach Funktionsgruppen von forth words filtern kann!

Ich weiß nicht, woran es liegt, dass die anfängliche Begeisterung für Forth auf dem ESP32 offensichtlich nachgelassen hat. In der jetzigen Form finde ich es zwar hochinteressant, aber eben noch nicht industrietauglich, zumindest für die meisten von uns. Dennoch erscheint es mir sehr wichtig, sich mit diesem Spottpreis–Rechner und dem zur Verfügung stehenden Forth näher zu befassen.

Es sollte jedem Soziologen zu denken geben, dass ein USB–Anschluss–Kabel für ein solches Super–Rechnerchen schon gerne teurer ist als die Elektronik selbst, und das, obwohl man beim Billig–Discounter Action[5] diese Kabel aktuell sehr günstig bekommt.

Dann wäre es schon schön, wenn man auf einem Smartphone ein Terminal–Programm hätte, mit dem man per USB direkt mit einem ESP32 kommunizieren könnte. Bis jetzt habe ich noch keins gefunden, aber vielleicht gibt es jemanden, der einen guten Hinweis liefern kann.

Viele freundliche Grüße in die Runde! Michael Hermann

https://coolterm.en.lo4d.com/windows
https://github.com/flagxor/espforth

ESP32Forth Glossaire

Marc Petremann hat ein Glossar zu den forth words online, das man nach Funktionsgruppen filtern kann. Suchst du die Worte für die serielle Kommunikation? Voilà, da sind sie. Du erhältst vorneweg eine kurze Auflistung von 15 Worten für die beiden UARTs und danach wird aber auch deren ausführlichere Verwendung näher erklärt. Das ist eine tolle Sache! So eine *handy reference card* braucht man, um sich in Forth zu orientieren, sonst ist man ziemlich verloren. Das FIG–Forth für den AIM65 hatte so etwas noch, nun ist es wieder da für das ESP32–Forth — super!

Das Glossar liegt sogar in französicher und englischer Sprache vor. mka

https://esp32.arduino-forth.com/index/glossaire/

Bemerkung zum ESP32–Treiber (win10)

Hat man, so wie ich, das Glück beschenkt zu werden mit so einem Wroom–Platinchen und ESP–Forth schon drin[6], denkt man: „Prima, dann kann ich ja gleich loslegen und damit spielen!“ In Forth, versteht sich.

Doch ganz so war das dann doch nicht. Windows 10 spielt zwar sein dudelidi, den „Neues–Gerät–erkannt–Ton“, macht aber keinen COM–Port dafür auf. Somit lässt sich *CoolTerm* (oder irgendein anderes Terminal) auch nicht mit Forth verbinden.

Der Geräte–Manager wusste da schon, dass er keinen Treiber für die *CP210x–USB–to–UART–Bridge* des Platinchens im System hatte. Und im „Getting Started Guide“ von *Espressiv* fand sich, warum das so ist: Espressif geht davon aus, dass man ihr *IoT Development Framework* (ESP–IDF) benutzt.

> „ESP–IDF requires some prerequisite tools to be installed so you can build firmware for supported chips. The prerequisite tools include Python, Git, cross–compilers, CMake and Ninja build tools . . . “

> „Standard Setup of Toolchain for Windows:
> For this Getting Started we are going to use the Command Prompt, but after ESP–IDF is installed you can use Eclipse Plugin or another graphical IDE with CMake support instead . . . “

Da wir Forther all sowas natürlich total verschmähen — wir wollen ja unmittelbar in den Chip rein — stehen wir natürlich auch ohne die Treiber da; jedenfalls bei meinem Windows 10 sind die nicht dabei gewesen. Immerhin wusste ich nun, warum da nichts ging.

Im Internet fand sich dann der *Silicon Labs* CP210x–USB–to–UART–Bridge–Treiber. Das Paketchen runtergeladen und entzippt, fand der Gerätemanager in dem Ordner, was er brauchte. Nein, nicht automatisch, da muss man ihn schon hindirigieren, aber damit gings dann. CoolTerm war zufrieden mit dem neuen COM11 und ab da flutschte es auch mit dem ESP–Forth.

PS: *Linux Mint 21.1* erkannte den ESP32 ohne weiteres und CoolTerm verband sich sofort zum Gerät, sobald man die richtige ttyUSB–Verbindung ausgewählt hatte — OK und alles lief einwandfrei. Dabei läuft das auf einem betagten Laptop; Lenovo Thinkpad L412, Intel core I3

[3] In Windows both a CR and LF are required to note the end of a line, whereas in Linux/UNIX and Mac it is LF only.

[4] Die Sache mit den Steuerzeichen Carriage Return und oder Linefeed: Wenn ich in der Textbearbeitungs–Software *Pages* auf meinem Mac den Programmtext vorformuliere, um ihn dann per Copy&Paste seriell dem ESP32 zuzuführen, kommt es schon mal vor, dass das Zeichen für den Zeilenumbruch nicht vorhanden ist und somit auch nicht gesendet werden kann. Schon ist der mysteriöse Fehler da. Also mit entsprechenden Hilfsmitteln inspizieren!

[5] Niederländischer Discounter mit inzwischen über 2300 Filialen in 11 Ländern.

[6] Vielen Dank, Michael Hermann.

@2,13 GHz; aufgerüstet auf seine max 4 GB RAM und mit einer flotten neuen 400–GB–SSD.

PPS: Auf dem Mac–Mini gab es dem Vernehmen nach auch keine Probleme. Allerdings war dort zuvor auch die ESP–IDF eingerichtet worden.

C'est la vie avec Forth. mka

My4TH schrumpft

Wir erinnern uns: My4TH ist ein kleiner Forth–Rechner von Dennis Kuschel, der mit einer diskreten CPU auskommt und bastelfreundlich mit handelsüblichen Bauteilen bestückt ist, die in Durchsteckmontage auf einer Platine der Größe 10x10 cm² Platz finden (Abb. 2). Ziel war es, den Rechner mit den wenigsten Standardbauteilen zu konstruieren, der Forth ausführen kann. Nähere Informationen gibt es auf http://mynor.org/my4th

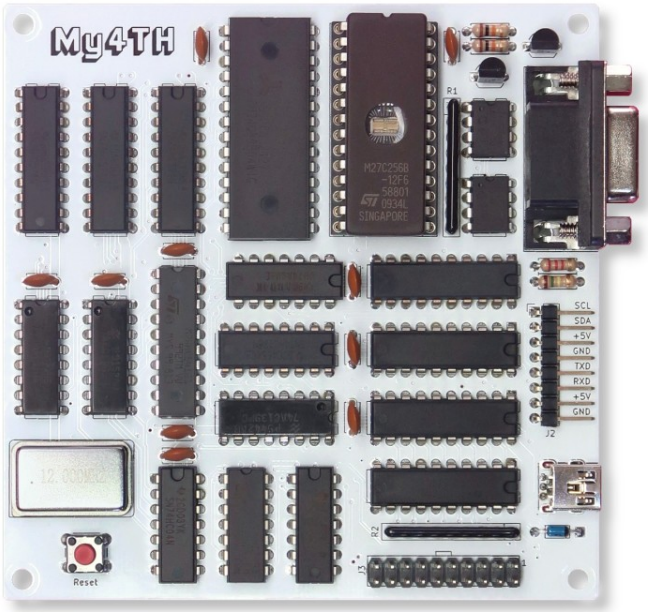


Abbildung 2: My4TH, das Original

Nachdem das Ziel erreicht war, den „minimalsten" diskreten Rechner der Welt entwickelt zu haben, ließen Dennis zwei Gedanken nicht los: Geht es noch kleiner? Und wenn ja, wie?

Und ja, er hat es tatsächlich geschafft.

Zeitgleich mit Erscheinen dieser VD wird Dennis seine neuen Entwicklungen auf seiner Website vorstellen. Es hat uns vorab mit Informationen versorgt, die wir euch nicht vorenthalten wollen.

My4TH light

Beim My4TH light (Abb. 3) konnte die Anzahl der Logikbausteine von 16 auf 13 reduziert werden. Er büßt dafür seine freien IO–Pins ein, UART und I²C–Bus sind aber vorhanden und können für Erweiterungen genutzt werden. My4TH light ist daher kein Ersatz oder kompatibler Nachfolger des originalen My4TH, eher eine Nebenlinie.

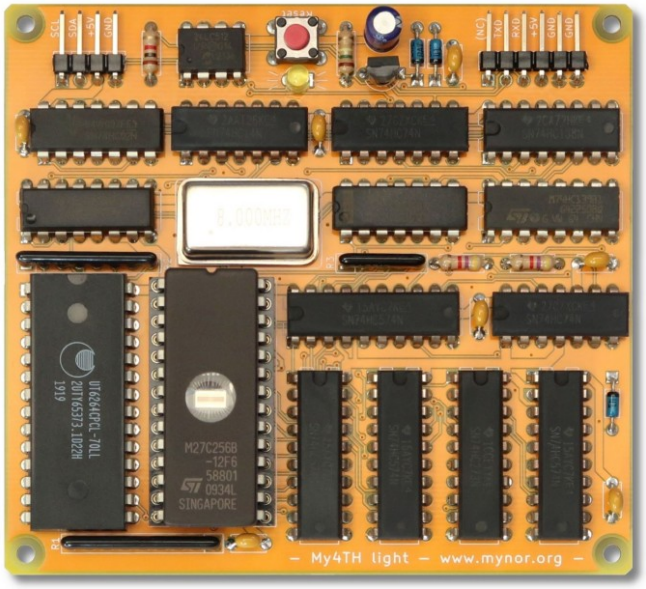


Abbildung 3: My4TH light

My4TH light Spezifikationen:

- Sehr kleine, diskrete 8–Bit–CPU aus 13 Logikbausteinen der 74HC–Serie
- 8–Bit–Computer–System mit Von–Neumann–Architektur, 8 MHz Systemtakt
- 1–Bit–NOR–Gatter–Logikeinheit (keine „ALU")
- 5200 8–Bit–Additionen pro Sekunde @ 8 MHz
- 32 KB EPROM, 64 KB EEPROM und 8 KB SRAM
- Software–UART mit 4800 Baud
- Schneller I²C–Bus mit bis zu 83 kHz @ 8 MHz
- Fließkomma–Unterstützung!
- Ins ROM eingebaute Unterstützung für einige I/O–Port–Expander, I²C–LCDs und den I²C–UART SC16IS750
- Nur 300 mW Leistungsaufnahme bei einem Systemtakt von 8 MHz
- Unterstützt den FORTH 2012 Standard. Enthalten sind die Forth Core words, Core extension words, das Double–Number word set, das Block word set, Floating Point words, und viele andere Worte mehr.

My4TH XS

Der My4TH XS (Abb. 4) stellt das (vorläufige) Ende der Entwicklungen von minimalistischen Forth–Rechnern dar. Hier hat Dennis nochmal einen Baustein „weggekürzt". Keine IOs, kein I²C für Erweiterungen. Geblieben ist der Software–UART für die Kommunikation mit dem Terminal. Dennis sieht den XS als eine Machbarkeitsstudie, um herauszufinden, was möglich ist. Ist der XS wegen seiner mangelnden Konnektivität nun nutzlos für den Forth–Enthusiasten? Ich finde, der XS hat seine Daseinsberechtigung. Auf ihm lässt sich hervorragend Forth lernen. Außerdem ist er ein Schmuckstück und ein einzigartiges Stück Forth–Hardware.

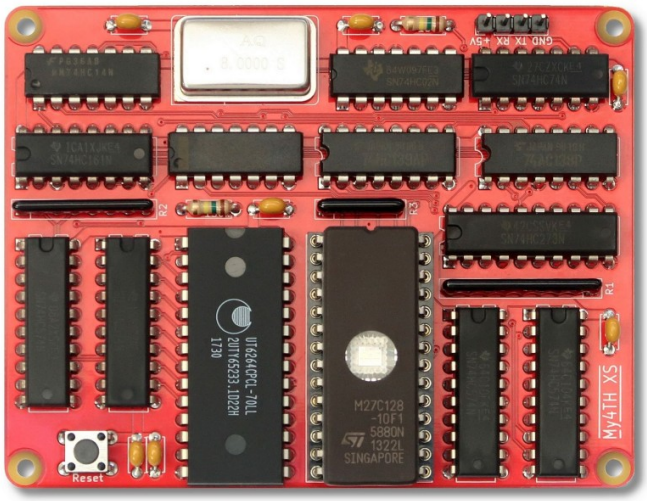


Abbildung 4: My4TH XS

My4TH XS Spezifikationen:

- Sehr kleine, diskrete 8–Bit–CPU aus 12 Logikbausteinen der 74HC–Serie
- Kleine Platine mit nur 70x90 mm² (das sind nur 39% einer Eurokarte)
- 8–Bit–Computer–System mit Von–Neumann–Architektur
- 8 MHz Systemtakt
- 1–Bit–NOR–Gatter–Logikeinheit (keine „ALU“)
- 5200 8–Bit–Additionen pro Sekunde @ 8 MHz
- 16 KB EPROM und 8 KB SRAM
- Software–UART mit 4800 Baud
- Leistungsaufnahme kleiner 300 mW bei einem Systemtakt von 8 MHz
- Unterstützt den FORTH 2012 Standard. Enthalten sind die meisten Forth Core words,Core extension words und das Double–Number word set

Wolfgang Strauß

Neues My4TH ROM

Neben den faszinierenden Hardware–Entwicklungen in Sachen My4TH gibt es auch Neuigkeiten in Sachen Software zu berichten. Dennis Kuschel hat standardmäßig Unterstützung für Peripherie–Einheiten wie z. B. LCDs oder seinen „Forth Deck“ Handheld–Computer (Abb. 5) in die EPROM–Forth–Firmware eingebaut. Damit sind die 32 KB des EPROMs gut gefüllt.

Abbildung 5: Das Forth Deck. Tipp: Video anschauen auf YouTube: https://youtu.be/Kn0MxHlima0

Aber nicht jeder baut sich so einen schickes Forth Deck, und so ist für diese Benutzergruppe wertvoller EPROM–Speicher belegt mit nicht relevanten Funktionen. Was könnte man mit dem freigewordenen Speicher alles anfangen? Nun, Dennis musste nicht lange überlegen und hat folgende Updates/Verbesserungen implementiert. Er nennt das alternative ROM zur Unterscheidung zum „normalen“ ROM die „nfd“ (no forth deck) Version.

- Erweiterte Zahlenverarbeitung. Prefixe $, %, # für Hex–, Binär– und Dezimalzahlen. Der Unterstrich „_“ ist zur besseren Lesbarkeit von Gruppen in Zahlen erlaubt. Diese neue Erweiterung ist aber auch in allen anderen neuen ROMs enthalten.
- Verbesserter Screen–Editor. Shift–Left, Shift–Right, Insert, Delete und „in Place Editing“ im Screen sind jetzt möglich.
- Mindestens 30% höhere Ausführungsgeschwindigkeit
- Schnellere Compilierung
- Schnelleres I²C

Wolfgang Strauß

VolksForth Release V3.9.5 ist da

Nachdem im letzten Jahr fast 100% meiner verfügbaren Hobbyprogrammierzeit in den CP/M–Zweig des VolksForth geflossen ist (da geht’s jetzt wahrscheinlich ans Eingemachte des Targetcompilers), habe ich es in den letzten zwei Wochen doch geschafft, der Commander–X16–Version mal wieder ein Stück der dringend nötigen Pflege angedeihen zu lassen, und es an die neueste R46–Version der X16–Firmware anzupassen, einschließlich Einbindung des jetzt standardmäßig im ROM vorhandenen X16Edit. Bei der Gelegenheit habe ich auch das erste GitHub–Release für VolksForth erzeugt. Der Name des Release bei GitHub zeigt an, daß es nur die CBM–artigen Maschinen betrifft.

Philip

https://github.com/forth-ev/VolksForth/releases/tag/v3.9.5-6502-C64

https://cx16forum.com/forum/viewtopic.php?t=6979

Screen– und sequenzieller Editor in Forth

Klaus Kohl–Schöpe

Schon vor über 30 Jahren hatte ich einen Screen–Editor für mein KKForth realisiert. Mein neueres mcForth *kann aber auch mit sequenziellen Files umgehen und deshalb wollte ich einen Editor, der mir auch da bei Fehlern die richtige Stelle zeigt. Als kleiner Bonus wurde dann auch noch das Markdown–Feature eingebaut, was mir die wichtigsten Forth–Worte farbig anzeigt.*

Ziele

Die Vorgabe war, dass alle Features des Screen–Editors vorhanden und auf sequenzielle Files ausgedehnt werden:

- Einstellbare Fenstergröße
- Einstellbare Screengröße
- Zeichen– und Zeilenpuffer
- Start bei bestimmter Zeile/Position
- Suchfunktion
- Markdown–Support für Forth

Da ich meinen ersten Editor auch bei Messsystemen mit 8*40–Zeichen–Display verwendet habe, war ein Scrollen der Anzeige wichtig. Außerdem sollten nicht nur die üblichen 16*64–Zeichen–Screens, sondern auch die C64–Version mit 25*40+24 Zeichen anzeigbar sein. Ein getrennter Puffer für Zeichen und Zeilen erlaubt das Kopieren bzw. Verschieben von Textteilen. Natürlich gehört zu einem Editor auch das Suchen und Ersetzen von Texten.

Nachdem der Editor funktionierte, überlegte ich mir, auch gleich noch das Markdown einzubauen, um Zahlen und Strukturen besser zu sehen.

Noch nicht vollständig realisiert, aber schon vorbereitet, ist das Handling eines zweiten Files, was bei Nutzung zu einer Aufspaltung des verfügbaren Speichers (freier Bereich über heap) führt.

- Getrenntes Handling von Flash / RAM (c@p statt c@ . . .)
- Andere Filebefehle und Tastencodes
- Farben über ESC–Sequenz

Gerade wegen der unterschiedlichen Tastencodes sind einige Funktionen nicht genutzt, können aber leicht wieder aktiviert werden. Leider sind auch die Farben und Zeichen für die Rahmen anders. Aber mein Ziel war hier nicht eine vollständige Anpassung, sondern ein funktionierendes Grundgerüst auf ANS–Forth–Basis.

```
d:\_Projekte\WSForth\mcForth32_1.1_Dev\Apps\mcfedwin.f
y=   146  x=     1  (L:  0/C:  0) ( )

\ ------------------------------------------------------------
\ Line and character buffer
\ ------------------------------------------------------------
\ Editor can copy or move lines or character to this buffer
\ It will be handled as a stack: last in - first out
\ For a line also the size of the line will be saved
\ Line buffer:
  Variable lb_b                 \ Start of line buffer
  Variable lb_n                 \ Saved lines
\ character buffer:
  Variable cb_b                 \ Begin of  character buffer
  Variable cb_n                 \ Saved characters

  : lbcbinit    ( -- ) \ Initialize buffer for line and chars
    here  unused +  $100 -   dup cb_b !  lb_b !
    0 cb_n !   0 lb_n ! ;

\ Check available free spaces
  : lbcb_free?  ( -- n ) \ memory for line or character ?
    cb_b @                      \ Begin of character buffer
    fpar @ IF fpar ELSE spar THEN cell+ @  -  ; \ - End of file

  : ed_free?    ( -- n ) \ memory for line or character ?
    fpar @ 0=   par @  fpar  =  or
    IF   lbcb_free?             \ only one file or fpar used
```

Abbildung 1: Screenshot (Im Original bunt!)

Verwendung

Die Verwendung ist einfach, da der Editor in mcForth das aktuell offene File verwendet und mir bei **v** die aktuelle Fehlerposition und bei **l** (line --) die angegebene Zeile zeigt.

Gforth erwartet Zeile und Filename:

line s" Filename" edit

Für die Tastaturbelegung am besten das Sourcefile ansehen (oder evtl. ändern).

Anpassung an Gforth

Das mcFEdWin.f ist auf meinem *mcForth für Windows* entstanden. Deshalb hat es einige Besonderheiten, die ich für das ebenfalls verfügbare GforthEdWin.f (hier auch für Windows) „eliminieren" musste:

- Nutzung von Create / End-Create

Implementierung

Ich hoffe, dass dieser Editor mit den vielen Kommentaren gut zu verstehen und zu ändern ist, da fast alle Parameter wie Bildschirm– und Screen–Größe, verwendete Rahmenzeichen und Farben als Konstanten bzw. als Value (wenn veränderbar) definiert wurden. Da mein alter Screeneditor mit zwei Files parallel umgehen konnte, sind noch Reste davon in der Source vorhanden. Deshalb sind alle mit sv_... beginnenden Variablen für File–ID, Speicher–, Bildschirm– und Cursorposition in einem Datenfeld gespeichert, auf das par zeigt und entweder bei spar oder fpar liegen kann. Da aktuell nur ein File verwendet wird (sv_fid in fpar ist 0), wird der gesamte freie Speicher für das File sowie den Zeichen– und Zeilenpuffer verwendet. Dies wird aber nur bei der Abfrage des freien Speichers für das File oder den Puffer gebraucht.

Das File wird mit loadfile komplett in das interne RAM oberhalb von here geladen, Tab durch Space ersetzt und danach geprüft, ob es ein Screenfile (* #c/s Zeichen ohne CR bzw. CR/LF) oder ein sequenzielles File ist. Das Ergebnis wird dann in sv_seq? gespeichert, welches mit Wert 0, 1 oder 2 auch die Anzahl der Zeichen am Ende einer Zeile zeigt. Da eine Mischung von CR und CR/LF nicht erlaubt ist, wird eine Fehlermeldung ausgegeben. Am Ende eines Files wird bei Bedarf CR bzw. CR/LF ergänzt und damit schon das File geändert. Dies ist sichtbar als *() in der 2. Zeile des Editors hinter der Angabe der Cursorposition und der Anzahl an Zeichen und Zeilen im Puffer. Die Größe des Fensters für den Editor wird hier auch mit 80*25 für sequenzielle Files oder dem Screenformat 16*64 (#l/s und #c/l) festgelegt.

Der Rahmen des Editors wird mit frame aufgebaut und mit lcdrestore gefüllt, wobei einzelne Bits in disp? entscheiden, ob das ganze Fenster, der Teil ab dem Cursor oder nur die aktuelle Zeile aktualisiert wird. Deshalb wird vorher noch geprüft, ob der Cursor überhaupt noch im Fensterbereich ist und evtl. der Ausschnitt korrigiert. Danach wird der Cursor gesetzt und dessen Position in der 2. Zeile aktualisiert.

Der eigentliche Start des Editors ist edit (mcForth) bzw. (edit (GForth), welches das File lädt, die gewünschte Stelle (Zeile oder Position im File) sucht, den Bildschirm aufbaut und dann entsprechend der gedrückten Tasten reagiert. Mit ESC wird der Editor verlassen, wobei bei Änderung gefragt wird, ob das File gespeichert werden soll. Die verwendeten Tastencodes sind als Konstanten und die Funktionen bei eseq als lange Tabelle abgelegt, welche durch sequencer gesucht und ausgeführt wird.

Die Funktionen selbst, wie Cursorbewegung, Einfügen oder Löschen, Handling des Zeichen– und Zeilen–Puffers, Suchen und Ersetzen, Zusammenfügen oder Aufteilen von Zeilen oder Wechsel von Groß– zu Kleinschrift (und umgekehrt) dürfte in den Sourcen leicht zu verstehen sein. Ein Überbleibsel aus dem alten Editor ist noch das Einfügen einer ID — also Datum mit bis zu 4 Zeichen Kennung. Mit F10 kann ich diese vorgeben und mit F8 wird dann in Klammern diese ID an der Cursorpostion eingesetzt.

An vielen Stellen wird man den Einfluss von sv_seq? feststellen, da damit nicht nur die nicht druckbaren Zeichen am Zeilenende festgelegt werden, sondern es sorgt auch dafür, dass man beim Scrollen nicht aus dem aktuellen Screen herauskommt. Da der Editor immer im *Insert–Mode* ist, wird beim Einfügen in Screenfiles auch geprüft, ob noch genügend Zeichen am Zeilenende frei sind bzw. ob die letzte Zeile leer ist.

Markdown

Bei der Ausgabe einer Zeile kommt in line auch das Markdown zum Einsatz, das damit nur auf diese Zeile begrenzt ist. Selbst, wenn der Anfang der Zeile nicht sichtbar ist, muss dieser beachtet werden, um die aktuelle Farbe zu ermitteln. Dies geschieht mit einer *Statemaschine*, die sich merkt, mit welcher Farbe bis zu welcher Adresse ein Text ausgegeben wird und ob danach Leerzeichen übersprungen oder nach einem bestimmten Zeichen gesucht werden soll. Deshalb enthält die Tabelle für das Marking neben dem Befehlsstring den nächsten State, die Farbe für den gefundenen String, die Farbe für das evtl. nachfolgende Wort oder den String und das Zeichen, das diesen String beendet. Damit hat man die Möglichkeit, nicht nur das Wort selbst, sondern ganze Strings oder Kommentare wie \ , \\ oder (...) und nachfolgende Worte in unterschiedlichen Farben anzuzeigen. Eine Besonderheit ist dann noch das Erkennen von Zahlen, die ebenfalls wie Strings in blauer Farbe angezeigt werden. Dabei ist mir aufgefallen, dass in Gforth das # auch schon als Zahl behandelt wird weil das verwendete >number? eine 0 erkennt.

Wünsche

Wie immer, wenn man mit einem Projekt (fast) fertig ist, kommen weitere Wünsche auf, die man sich für das nächste Update (hoffentlich nicht wieder in 30 Jahren) merkt. Diese sind hier:

- Volle Implementierung des 2. Files
- Laden/ Speichern (andere Namen)
- Undo–Puffer
- Kopieren beliebiger Bereiche
- Kooperation mit Compiler
- Erweiterung zum Projektmanager
- Optimierung der Geschwindigkeit

Ideal ist es natürlich, wenn der Editor, ähnlich wie *Visual Studio Code*, gleich das ganze Projekt oder Verzeichnis verwaltet. Der Anfang wäre schon das Handling eines 2. Files und Laden/Speichern unter anderem Namen. Ein kleiner Schritt in diese Richtung ist schon, wenn man Teile aus dem File direkt an den Compiler schicken könnte. Man merkt deutlich den Aufbau des Bildschirms, was vermutlich größtenteils auf Markdown und der Suche nach der gewünschten Zeile geschuldet ist.

Copyright

Da ich selbst lange nach einem Editor für sequenzielle Files in Forth gesucht habe, gebe ich mcFEdWin.f und GforthEdWin.f unter MIT–Lizenz frei. Das bedeutet, dass Ihr es frei für eigene Projekte verwenden könnt, aber ich dafür keine Gewährleistung übernehme. Ich würde mich über häufige Verwendung und der Erwähnung der Quelle freuen. Die beiden Files und diese Anleitung in deutscher und bald auch englischer Sprache ist in meinem *GitHub* unter mcForth zu finden[1]:

https://github.com/KlaKoSch/mcForth/tree/main/EdWin

Viel Spaß damit.

[1] So schaut bitte dort in die Quellen. Das soll hier nicht alles abgedruckt werden.

Ein Programmiergerät für den My4TH–Einplatinenrechner

Wolfgang Strauß

In der letzten VD (vd2023–03) wurde Dennis Kuschel*, der Entwickler von My4TH, vorgestellt und es gab einen Erfahrungsbericht über das Gerät nebst kleinen Experimenten von* Matthias Koch*. Auch ich bin stolzer Besitzer eines solchen Forth–Mini–Rechners. Nach dem ersten Kennenlernen kam sehr schnell der Wunsch auf, die Firmware (den Forth–Kern) zu aktualisieren. Doch es gab da ein Problem . . .*

Das Problem

In My4TH ist als Festwertspeicher ein EPROM verbaut. Die Älteren unter uns erinnern sich: Das sind diese vielbeinigen Käfer im Keramikgehäuse mit einem Quarzglasfenster, durch das Licht auf den Chip scheinen kann.

Abbildung 1: EPROM

Sie werden mit UV–Licht der passenden Wellenlänge gelöscht und können durch Anlegen einer deutlich höheren Spannung als der normalen Betriebsspannung programmiert werden. Die Bausteine werden in Neuentwicklungen nicht mehr verwendet, sind aber erstaunlicherweise noch gut erhältlich, die Preise sind akzeptabel. So hat halt das EPROM seinen Platz im My4TH–Projekt gefunden.

Das EPROM in meinem Bausatz hat die Version 1.1 programmiert. Die Entwicklung bleibt aber nicht stehen, Dennis behebt Fehler, wenn sie ihm zu Ohren kommen und optimiert auch mal an der einen oder anderen Stelle. So ist momentan die Version 1.3 online und die V1.4 in den Startlöchern.

Um jetzt aber einen solchen Baustein selbst „bewirtschaften" zu können, braucht es ein passendes Programmiergerät und eine passende UV–Quelle. Hatte ich alles mal, lang ist's her, und nun stehe ich ohne da. Was tun? Es gibt nette Menschen im Umfeld des Vereins, die die passende Gerätschaft besitzen und mir sicher das das eine oder andere EPROM brennen würden.Das kann aber schnell lästig werden mit dem Hin–und–her–Schicken der Bausteine. Außerdem denke ich da nicht nur an mich, sondern an alle Benutzer von My4TH. Wie schön wäre es, wenn My4TH seine eigenen EPROMS brennen könnte?

Die Lösung?

Wie schwer kann es schon sein, so ein Programmiergerät selbst zu bauen? Ein paar IO–Bausteine, eine schaltbare Spannungsquelle und eine dieser preiswerten UV–LED–Lichtquellen, mit denen modebewusste Damen den UV–Lack auf ihren Fingernägeln härten. Fertig.

Nein, nicht wirklich. Die billigen UV–Lampen haben die falsche Wellenlänge und die Programmierung von EPROMs ist komplexer, als ich in Erinnerung hatte. Die EPROMs meiner Jugend waren mit einer konstanten Spannung von 21 V am Vpp–Pin und nach Anlegen der Adresse und Datum mit einem einfachen 50–ms–Puls zufrieden, um das Byte dauerhaft auf dem Chip abzulegen. Nicht so das auf dem My4TH verwendete EPROM des Typs 27C256. Ich möchte euch hier nicht mit zu vielen Details langweilen, nur soviel: Der Baustein hätte gerne eine gewisse Anzahl präzise getimter Pulse und mehrere Spannungen, bei denen die Daten verifiziert werden sollen. Das hätte die Schaltung doch stark verkompliziert und die Nachbausicherheit in Frage gestellt.

Ich habe dann nach anderen Bausteinen gesucht,die nicht so sperrig sind. Dennis sagte, jemand hätte ihm den W27C512 von Winbond empfohlen. Ein kurzer Blick ins Datenblatt offenbarte dann, das der Chip ein EPROM ist, das aber elektrisch löschbar ist, also kein UV–Licht braucht. Das Problem mit dem Beschaffen der Lampe entfällt damit. Aber es bleibt halt ein EPROM, also wieder nichts.

Die Lösung!

Es gibt tatsächlich einen Baustein, der aktuell noch gefertigt wird und die Anforderungen an einfache Programmierbarkeit erfüllt. Es ist das Flash SST39SF010 von Microchip, erhältlich im 32–poligen DIP–Gehäuse mit einer einfachen Versorgungsspannung von 5 V. Passt. Naja, fast. Da auf der My4TH–Platine ein 28–pol. Sockel sitzt, ist ein einfacher Adapter fällig, der aber schnellselbst hergestellt werden kann. Dazu gleich mehr.

Die Schaltung

My4TH hat 8 digitale Eingänge und 8 digitale Ausgänge. Die reichen leider nicht aus, um das parallele Flash SST39SF010 mit allen nötigen Signalen zu versorgen. Der Chip hat 32 Pins, davon sind 2 für die Spannungsversorgung. Es bleiben also noch 30 Signale,die angesteuert oder gelesen werden müssen. My4TH hat eine eingebaute I^2C–Unterstützung. Es bietet sich also an, sogenannte I^2C IO–Expander zu verwenden. Der PCF8575 ist so einer. Er hat 16 IO–Pins und ist sehr einfach anzusteuern, weil er keine internen Register hat,die aufwändig konfiguriert werden müssen. Um die 16 IO–Pins auf einen Wert zu setzen, werden immer paarweise 2 Bytes auf

den Bus geschrieben.Nach Erhalt des zweiten Bytes erscheinen die 16 Bits gleichzeitig an den Pins. Die Eingangs/Ausgangsstufen kann man sich als Open–Drain mit Pullup–Widerstand vorstellen.Schreibt man eine Null, so schalten die Transistoren und ziehen den Pin auf Low. Schreibt man eine Eins, sperrt der Transistor und der Pin wird durch den (hochohmigen) Widerstand nach oben gezogen (Pullup). In diesem Zustand kann der Pin auch von außen auf Low oder High gelegt werden und agiert so als Eingang, der gelesen werden kann. Ein Datenrichtungsregister ist deshalb nicht nötig. Für die 30 Signale braucht man also 2 Stück PCF8575 und hat noch einige Pins frei für eine Status–LED und was einem sonst so einfällt.

Ich habe mir die PCF8575 im TSSOP24 besorgt und auf ein DIP24–Breakout–Board gelötet.

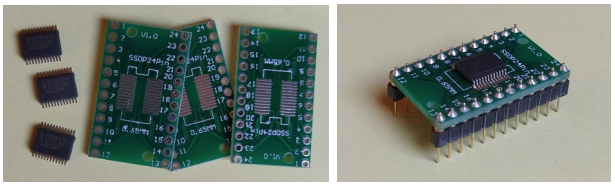


Abbildung 2: Breakout–Board für den PCF8575

Wem das zu fummelig ist, der kann sich fertige Boards von Adafruit oder auf den gängigen Handelsplattformen besorgen. Noch ein paar Widerstände und Kondensatoren dazu und man kann sich die Schaltung auf einem Breadboard zusammenstöpseln.

Die Flasher–Platine

Obwohl sich für diese einfache Schaltung das Zusammenstecken der Bauteile und Drähte auf einem Breadboard anbietet, kann ich leider keinen solchen Aufbau als Bild zeigen. Ich löte meine Prototypen lieber auf Lochrasterplatinen. Abb. 3 zeigt meinen Aufbau.

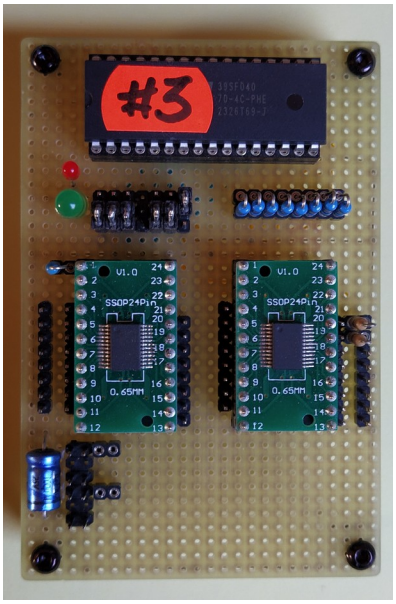


Abbildung 3: Die Flasher–Platine

Gut zu erkennen sind die zweiIO–Expander auf ihren Breakout–Platinchen sowie der darüber längs in seiner Fassung steckende Flash–Chip. Viel mehr braucht es nicht. Der dazu passende Schaltplan ist weiter unten im Artikel in Abb. 5 zu finden. Das „Programmiergerät“ wird über 4 Leitungen mit dem My4TH–Rechner verbunden. Neben der Spannungsversorgung (+5 V, GND) sind nur noch die zwei I²C–Signale SCL und SDA nötig, um den Flasher zu betreiben. Erwähnenswert sind vielleicht noch die 8 Widerstände in den Datenleitungen zwischen Flash und dem IO–Expander. Wer sich traut, kann die auch weglassen. Ich habe sie vorgesehen, um im Falle eines Programmierfehlers den Strom über die Datenleitungen zu begrenzen und so das Flash zu schonen. Der Fehlerfall tritt ein, wenn an einem IO–Expander–Pin eine Null programmiert ist (der Ausgangstransistor schaltet niederohmig nach Masse) und gleichzeitig das Flash auf Lesebetrieb gestellt ist und an dem entsprechenden Datenpin eine Eins ausgegeben werden soll. In der Folge fließt ein hoher Strom durch die Flash–Ausgangsstufe.

Der Adapter

Was dann noch fehlt, um die Sache rund zu machen, ist eine Möglichkeit, den Flash–Baustein anstelle des EPROMs in den My4TH einzusetzen. Wie schon erwähnt, hat das Flash 32 Pins, der Sockel auf der My4TH–Platine aber 28 Pins. Wie bekommt man den gerade frisch beschriebenen Flash nun in den Sockel? Na, mit einem Adapter (Abb. 4). Den kann man sich einfach selbst herstellen, indem man einen 32–poligen Präzisionssockel nimmt und dessen Pins 1, 2, 3 verbindet und dann noch die Pins 30, 31, 32 verbindet. Dabei darauf achten, dass die Stifte der Pins 3 und 30 nur ganz oben verlötet werden und ansonsten frei von Lötzinn bleiben, denn sie müssen ja noch in den 28–pol. Sockel auf der Platine passen. Die Signale VPP und VCC des EPROMs sind auf dem My4TH–Board mit +5 V verbunden. Somit haben die Adressleitungen A15 bis A18 einen festen High–Pegel. My4TH spricht also nur die obersten 32 KB des Flash an. Die Schaltung ist dementsprechend ausgelegt und bei der Software wird das natürlich auch berücksichtigt.

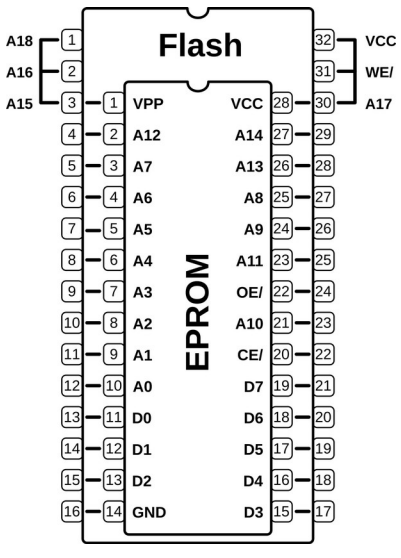


Abbildung 4: Der Sockel–Adapter für das Flash

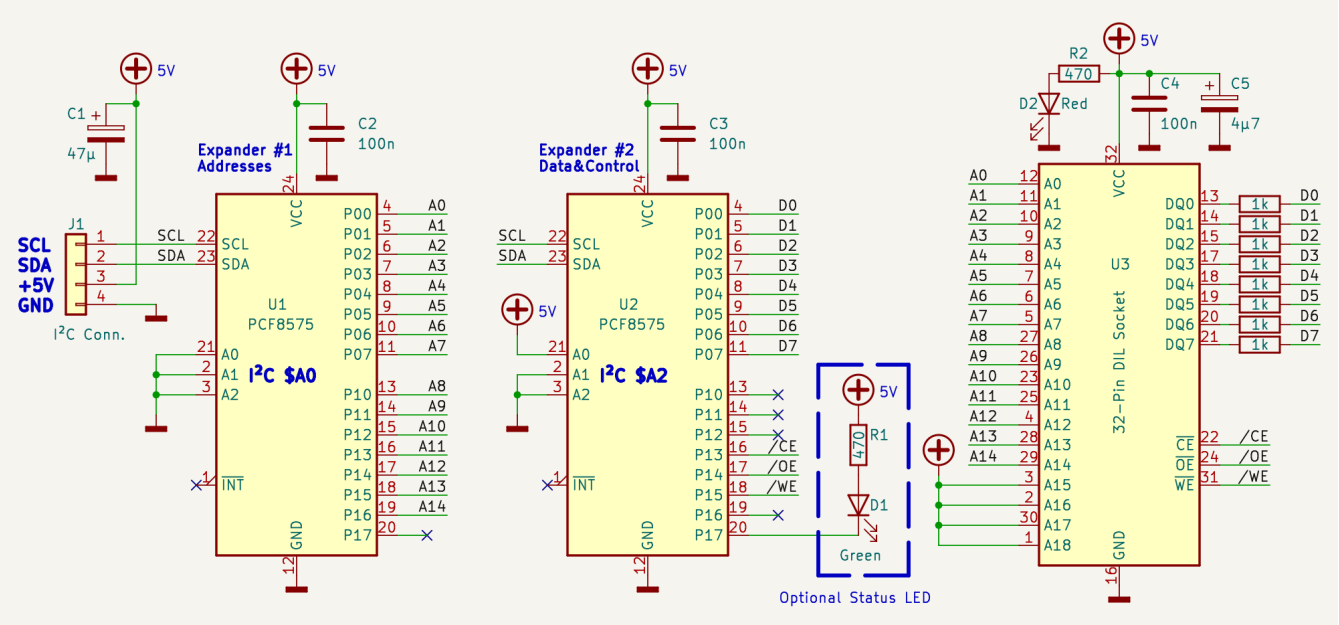


Abbildung 5: Schaltplan

Die Betriebssoftware

Schön, die Schaltung ist nun aufgebaut und alle Verbindungen sind noch einmalüberprüft. Dann kann es ja losgehen und das erste Flash programmiert werden.

Das my4th–Tool

Passend zum My4TH–Rechner liefert der Entwickler einen Werkzeugkasten in Form eines Programms für den PC mit, das sogenannte my4th–Tool. Es läuft unter Linux und Windows. Der Autor empfiehlt Linux, also benutze ich hier mein Linux Mint. Windows–User können einfach die mitgelieferte my4th.exe in einem Terminal ausführen, Linux–Freunde rufen einmalig das Shellscript build_my4th.sh auf und compilieren sich so die Programmdatei „my4th“ im aktuellen Ordner.

Das my4th–Tool ist eine „Eierlegende Wollmilchsau“. Wir benutzen hier nur die Datentransfer–Funktionen für das Übertragen von ROM–Images und Quellcode, aber das Tool kann mehr: Es sind auch ein Microcode–Generator, ein CPU–Hardwaresimulator, ein VCD–Signalschreiber für das Low–Level–Debugging sowie eine Funktion zur Erzeugung von EPROM–Abbildern enthalten. Wer will, kann also schon mit My4TH „spielen“, ohne reale Hardware zu besitzen. Faszinierend.

EPROM–Image ins Flash schreiben

Um ein EPROM–Image, z. B. von der My4TH–Website, in ein Flash zu bekommen, um dann damit den My4TH–Rechner zu betreiben, sind ein paar Schritte nötig:

1. Besorgen des Images von der Internetseite
2. Transferieren des Images in den Blockspeicher (EEPROM) von My4TH
3. Transferieren der Betriebssoftware in den Blockspeicher
4. Einloggen per Terminal in My4TH
5. Compilieren der Betriebssoftware mittels „#Startblock #Endblock thru“
6. Aufruf von disk>flash , um das Image vom Blockspeicher ins Flash zu schreiben
7. Optional ein disk<>flash-verify , um die geschriebenen Daten auf Korrektheit zu überprüfen
8. System spannungslos schalten, Flash aus dem Programmiersockel entnehmen und mittels Adapter anstelle des EPROMs in die My4TH–Platine einsetzen

Doch der Reihe nach.

Voraussetzungen

Das Programmiergerät ist an den My4TH–Rechner angeschlossen, ein Flashbaustein steckt im Sockel. Auf der My4TH–Platine steckt mindestens ein I²C–EEPROM, damit Blockspeicher verfügbar ist. In den Blocks 32 bis 63 liegt dann später das 32–KB–ROM–Image für die Übertragung in den Flashbaustein. Der My4TH–Rechner ist mittels USB–Seriell–Adapter an den PC angeschlossen und mit Spannung versorgt. Wie so ein Aufbau aussehen kann, zeigt die Abb. 6 am Ende des Artikels.

Die 8 Schritte im Detail

Zu 1): ROM–Images gibt es auf der M4TH–Website (siehe Abschnitt „Links“)

Zu 2): Das ROM–Image muss nun in den Blockspeicher von My4TH. Dazu das my4th–Tool aufrufen:
my4th write /dev/ttyUSB0 binary 32 my4th-rom.bin
Hier meint „/dev/ttyUSB0“ den seriellen Port der Verbindung, „binary“ die Übertragung von Binärdateien, „32“ die Nummer des ersten Zielblocks im I²C–EEPROM. Abschließend folgt der Dateiname des ROM–Images.

Zu 3): Der Quelltext der Betriebssoftware wird mit folgendem Aufruf in den Blockspeicher transferiert:
my4th write /dev/ttyUSB0 10 flasher.fs
Hier fehlt das „binary“, deshalb wird eine Textdatei übertragen. Im Beispiel wird die Datei „flasher.fs“ ab Block „10“ im EEPROM abgelegt. Das Tool zeigt bei der Übertragung den Fortschritt und am Ende (hoffentlich) eine Erfolgsmeldung. Interessant ist hier auch die Nummer des letzten beschriebenen Blocks. Den braucht man später für das Compilieren als einen Parameter für „thru“.

Zu 4): ROM–Image und der Quelltext sind durch die Schritte 2) und 3) übertragen worden. Alles Weitere passiert interaktiv im Forth–Interpreter von My4TH. Dazu einen Terminal–Emulator starten (z. B. picocom):
picocom -b 4800 /dev/ttyUSB0
Nach einem Druck auf den Reset–Knopf von My4TH meldet sich das Forth:
EEPROMs: 24LC512,none (64 blocks)
Welcome to My4TH(nfd4) v1.4 / 10 MHz

Zu 5): Das Compilieren der Worte ist schnell erledigt. Dazu „10 22 thru “ eintippen. In Schritt 3) wurde der Block 10 als Startblock zur Ablage des Quelltextes gewählt. flasher.fs enthält 12 Screens, diese wurden in die Blöcke 10 bis 22 des I²C–EPROMs geschrieben und werden nun mit „10 22 thru “ ins Dictionary compiliert.

Zu 6): das Wort disk>flash übernimmt nun das eigentliche „Flashen“. Dabei werden die 32 KB des ROM–Images in den Blöcken 32 bis inklusive 63 in den Flash–Baustein geschrieben. Das dauert ca. 16 Minuten. Wem das zu lang vorkommt, dem sei verraten, dass meine erste Version dafür fast 2 Stunden brauchte. Durch verschiedene Optimierungen konnte der Zeitbedarf um den Faktor 7 verringert werden.

Zu 7): Das optionale Verify mit disk<>flash-verify dauert „nur“ 4 Minuten und gibt die Sicherheit, dass alle Bytes ordnungsgemäß angekommen und aufgenommen wurden.

Zu 8): Fertig. Das Flash hat das ROM–Image aufgenommen und kann ausprobiert werden. Das Flash bitte nur im spannungslosen Zustand aus dem Programmiersockel ziehen und mittels des Adapters (Abb. 4) anstelle des EPROMs in den 28–poligen Sockel der My4TH–Platine einsetzen.

Sonstiges

Damit ist die Hauptfunktion der Software beschrieben: ein aktualisiertes ROM–Image in einen Flash–Baustein schreiben. Aufmerksame Leser des Quelltextes werden weitere nützliche Worte finden, die zum Experimentieren einladen. So gibt es das Wort rom>flash, welches das EPROM in ein Flash klont. Kleine Tools wie Speicherdumps von Flashinhalt oder EEPROM–Blöcken, Formatieren des Flashs, ein Leertest oder das Prüfen auf Vorhandensein der beiden IO–Expander runden die Software ab. Aktuelle Versionen der Software gibt es auf der My4TH–Website.

Ich wünsche viel Vergnügen beim Nachbau und Einsatz des Programmiergerätes.

Links

My4TH–Website:
http://mynor.org/my4th

Expanderboard von Adafruit:
https://www.adafruit.com/product/5611

Bezugsquelle SST39SF010, Sockel, Kleinteile:
https://www.reichelt.de/

Listing 1: Betriebssoftware „flasher.fs“

```
1   ----[010]-----------------------------------------------------------
2   \ Flash Tool for My4TH                                    wost 2023-12-01
3   \ Supported Flash chips: SST39SF010A, SST39SF020A, SST39SF040
4   \ IO-expander used: 2x PCF8575 (I2C, 16 IOs each)
5   \
6   \ Schematic:     Expander#1 ($40) : Address lines A0-A14
7   \ ==========    P01-P07 --> A0-A7       P10-P16 --> A8-A14
8   \                                       P17 --> n.c. (not connected)
9   \                Expander#2 ($42) : Data/Control lines
10  \                P01-P07 --> D0-D7
11  \                P10 --> n.c.       P11 --> n.c.       P12 --> n.c.
12  \                P13 --> /CE        P14 --> /OE        P15 --> /WE
13  \                P16 --> n.c.
14  \                P17 --> Status LED, green (optional)
15  \
16  \ Address lines A15-A18 are connected with +5V, so only the
17  \ upper 32KB of the chip are supported. See doku for details.
18  ----[011]-----------------------------------------------------------
19  \                                                         wost 2023-12-01
20  base @   hex   \ switch to hex for compilation
21  \ All literals in this driver are in hex.
22  \ To highlight this, all literals have a leading zero
23  040         constant EXP1W   \ Expander#1 write address
24  EXP1W 1+ constant EXP1R   \ Expander#1 read address
25  042         constant EXP2W   \ Expander#2 write address
26  EXP2W 1+ constant EXP2R   \ Expander#2 read address
```

```
27
28  : expander-check ( -- flag ) \ test if both expanders are online
29     EXP1W i2c-start 2 ms i2c-stop dup 0= if
30        cr ." Address Expander not present." then
31     EXP2W i2c-start 2 ms i2c-stop dup 0= if
32        cr ." Data/Control Expander not present." then
33     and dup if      cr ." Success. Both Expanders found." then
34     cr    ;
35  ----[012]----------------------------------------------------------
36  \                                                          wost 2023-12-01
37  : led+ ( -- )           EXP2W i2c-start drop       \ switch on the LED
38     0ff i2c-send drop         07f i2c-send drop        i2c-stop     ;
39  : led- ( -- )           EXP2W i2c-start drop       \ switch off the LED
40     0ff i2c-send drop         0ff i2c-send drop        i2c-stop     ;
41
42  \ Low level Flash write routine
43  : flash-w ( byte address -- )              \ write a byte to Flash chip
44     EXP1W i2c-start drop       \ address A0-A14 expander (write)
45        dup i2c-send drop       08 rshift i2c-send drop
46     i2c-stop
47     EXP2W i2c-start drop       \ data/control expander (write)
48        dup i2c-send drop     0d7 i2c-send drop      \ /CE low,       /WE low
49        i2c-send drop       0ff i2c-send drop            \ /CE high, /WE high
50     i2c-stop ;
51
52  ----[013]----------------------------------------------------------
53  \                                                          wost 2023-12-01
54  \ Low level Flash read routines
55
56  : flash-r{ ( -- )              \ open a Flash read sequence
57     EXP2W i2c-start drop       \ data/control expander (write)
58          0ff i2c-send drop         0e7 i2c-send drop      \ /CE low, /OE low
59     i2c-stop ;
60  : flash-r ( address -- byte )              \ read one byte from Flash
61     EXP1W i2c-start drop       \ address A0-A14 expander (write)
62          dup i2c-send drop       08 rshift i2c-send drop          i2c-stop
63     EXP2R i2c-start drop       \ data/control expander (read)
64          00 i2c-recv       01 i2c-recv drop         i2c-stop ;
65  : }flash-r ( -- )              \ close a Flash read sequence
66     EXP2W i2c-start drop       \ data/control expander (write)
67          0ff i2c-send drop         0ff i2c-send drop       \ /CE high, /OE high
68     i2c-stop ;
69  ----[014]----------------------------------------------------------
70  \                                                          wost 2023-12-01
71  : chip-erase ( -- )             \ erase whole Flash chip
72     cr ." Erasing the entire Flash chip ...                  "
73     0aa 05555 flash-w       055 02aaa flash-w       080 05555 flash-w
74     0aa 05555 flash-w       055 02aaa flash-w       010 05555 flash-w
75     064 ms   \ 100 ms max. for chip erase
76     ." Done." cr       ;
77
78  : sector-erase ( sector-address -- )               \ erase 4K-sector
79     base @ swap   hex    cr ." Erasing 4K Flash sector starting at $"
80     0f000 and dup u. ." ...             "
81     0aa 05555 flash-w       055 02aaa flash-w       080 05555 flash-w
82     0aa 05555 flash-w       055 02aaa flash-w
83     ( sector-address ) 030 swap flash-w
84     019 ms   \ 25 ms max. for sector erase
85     ." Done." cr       base !     ;
86  ----[015]----------------------------------------------------------
87  \                                                          wost 2023-12-01
88  \ High level Flash store/fetch routines
89
90  : flash! ( byte address -- )               \ write one byte to Flash
91     0aa 05555 flash-w       055 02aaa flash-w       0a0 05555 flash-w
92     ( byte address ) flash-w
93     ;     \ 20 us max. for byte write, so no delay needed
94
95  : flash@ ( address -- data )             \ read one byte from Flash
96     flash-r{       flash-r       }flash-r ;
97
98
99
100
101
102
```

```
103  ----[016]----------------------------------------------------------
104  \                                                        wost 2023-12-01
105  : disk>flash ( -- )             \ copy 32K to Flash, begin with block 32
106     base @   decimal     chip-erase
107     cr ." Writing 32K from disk to Flash, using blocks 32-63."
108     00                                                    ( Flash-addr )
109     020 00 do     \ 32 blocks
110        i 020 + dup       cr ." Block: " u.                ( Flash-addr block# )
111        block                                         ( Flash-addr block-addr )
112        dup 0400 + swap do     \ 1024 bytes each            ( Flash-addr )
113           i c@     over    flash!      1+                 ( Flash-addr+1 )
114        loop
115     loop
116     drop    \ remove Flash-addr from stack                        ( )
117     cr ." Ready." cr          base !     ;
118
119
120  ----[017]----------------------------------------------------------
121  \                                                        wost 2023-12-01
122  : rom>flash ( -- )           \ clone EPROM to Flash
123     base @   decimal     chip-erase
124     cr ." Writing 32K from EPROM to Flash, starting at addr. 0."
125     cr 08000 00 do      \ copy 32KB
126        i c@      i     flash!
127        i 03ff and 0= if          [char] * emit       then    \ every KB a star
128     loop    cr ." Ready." cr          base ! ;
129
130  : rom>disk ( -- )           \ copy EPROM to EEPROM, begin with block 32
131     cr ." Sorry. Not implemented yet." cr ;
132
133  : flash>disk ( -- )             \ copy Flash to EEPROM, begin with block 32
134     cr ." Sorry. Not implemented yet." cr ;
135
136
137  ----[018]----------------------------------------------------------
138  \                                                        wost 2023-12-01
139  : disk<>flash-verify ( -- )               \ verify EEPROM against Flash
140     base @   decimal     00 flash-r{                      ( Flash-addr )
141     cr ." Comparing 32K disk <-> Flash, using blocks 32-63."
142     020 00 do     \ 32 blocks
143        i 020 + dup cr ." Checking block: " u. ( Flash-addr block# )
144        block                                         ( Flash-addr block-addr )
145        dup 0400 + swap do     \ 1024 bytes each            ( Flash-addr )
146           dup flash-r       i c@     <> if      hex       ( Flash-addr )
147              cr ." Error at addr.: $" dup u.               ( Flash-addr )
148              cr ." Disk: $" i c@ u. ." ; Flash: $" flash-r u.          ( )
149              cr ." Exiting verify." cr
150              }flash-r       unloop unloop    base !     exit
151           then    1+                                   ( Flash-addr+1 )
152        loop    ."     --> passed."
153     loop drop }flash-r cr ." Verify was successful." cr base ! ;
154  ----[019]----------------------------------------------------------
155  \                                                        wost 2023-12-01
156  : rom<>flash-verify ( -- )              \ verify EPROM against Flash
157     base @   hex    flash-r{
158     cr ." Comparing 32K EPROM <-> Flash, starting at addr. 0." cr
159     08000 00 do
160        i c@ i flash-r <> if
161           cr ." Error at addr.: $" i u.
162           cr ." ROM: $" i c@ u. ." ; Flash: $" i flash-r u.
163           cr ." Exiting verify." cr }flash-r unloop                 base ! exit
164        then
165        i 03ff and 0= if          [char] * emit       then    \ every KB a star
166     loop      }flash-r       cr ." Verify was successful." cr              base ! ;
167
168  : rom<>disk-verify ( -- )
169     cr ." Sorry. Not implemented yet." cr ;
170
171  ----[020]----------------------------------------------------------
172  \                                                        wost 2023-12-01
173  : flash-blanktest ( -- )              \ check for all bytes = $ff
174     base @   hex     flash-r{
175     cr ." Testing the Flash to be blank." cr
176     08000 0 do
177        i flash-r 0ff <> if
178           cr ." Flash is not blank at addr.: $" i u.
```

```
179             cr ." Expected: $FF ; found: $" i flash-r u.
180             cr ." Exiting blanktest" cr }flash-r unloop                  base ! exit
181           then
182           i 03ff and 0= if          [char] * emit       then    \ every KB a star
183       loop    }flash-r       cr ." Flash is completely blank." cr                base ! ;
184
185
186
187
188   ----[021]-----------------------------------------------------------
189   \                                                                    wost 2023-12-01
190   : block-dump ( u -- )           \ dump 1K bytes of EEPROM block u
191       base @   swap     cr ." Dump of block #" dup         decimal u.      hex
192       block     0400 0 do
193           i 01f and 0= if         cr i 0 <# # # # # #> type           then
194           i     0f and 0= if      space    then
195           space    dup i + c@      0 <# # # #> type
196       loop    drop    cr    base !     ;
197
198   : flash-dump ( start-addr -- )              \ dump 1K bytes of Flash memory
199       base @   swap   hex    0fc00 and dup 0400 + swap 2dup      flash-r{
200       cr ." Flash dump $" u. ." - $" 1- u.                  do
201           i 01f and 0= if         cr i 0 <# # # # # #> type           then
202           i     0f and 0= if      space    then
203           space    i flash-r        0 <# # # #> type
204       loop    }flash-r       cr    base !     ;
205   ----[022]-----------------------------------------------------------
206   \                                                                    wost 2023-12-01
207
208   : test ( -- )            \ write a random byte to a random address.
209                              \ Then read back the byte before,
210                                \    the current byte and the next byte
211       base @   hex
212       expander-check    dup ." Result: " . cr
213       if
214           chip-erase
215           0e3 01f30 flash!
216           01f2f flash@ u.         01f30 flash@ u.         01f31 flash@ u.
217       then
218       base !     ;
219
220   base !     \ restore original base
221
222   ----[EOF]-----------------------------------------------------------
```

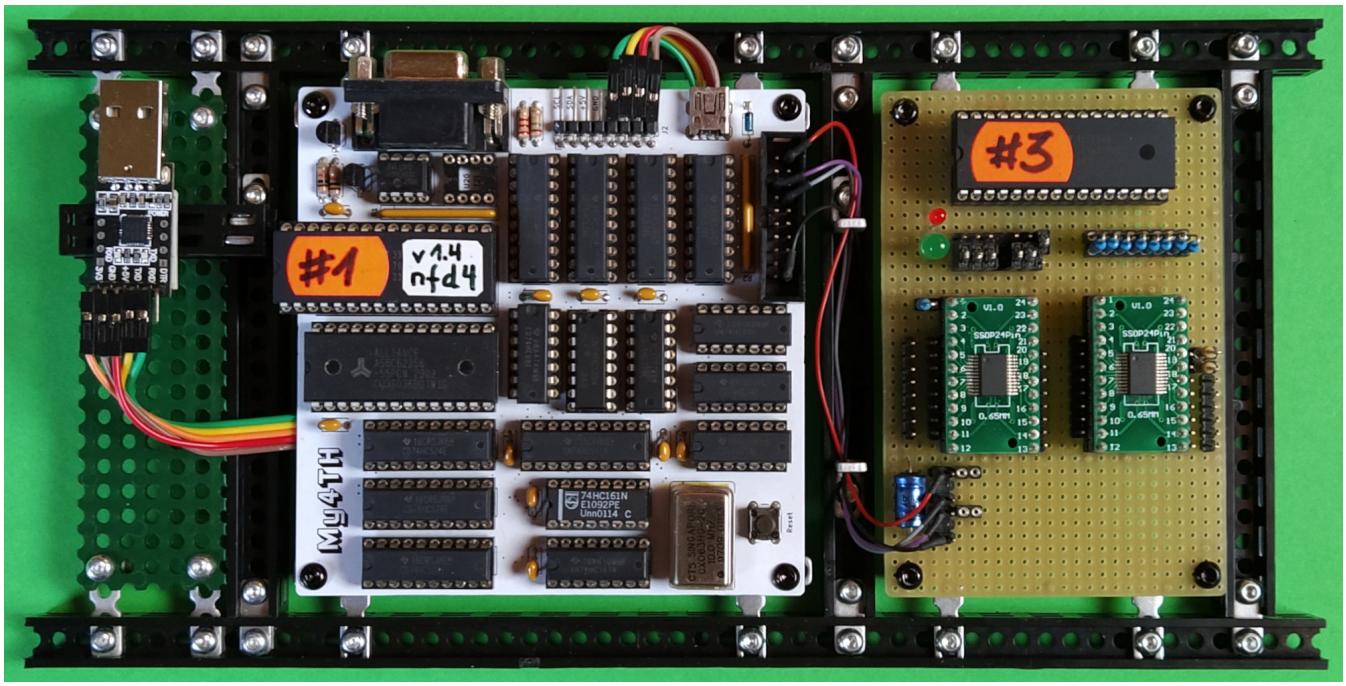


Abbildung 6: Der Aufbau mit drei Platinen wartet darauf, an den PC angeschlossen zu werden. Von links nach rechts: USB–Seriell–Wandler, My4TH–Einplatinenrechner, Programmier–Erweiterung. Um Fragen zuvorzukommen: Die Rahmenbauteile sind von https://totemmaker.net/

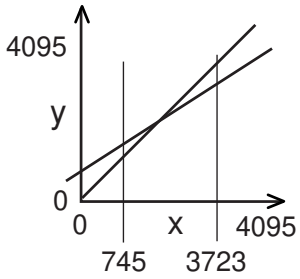
Fehlerkorrektur in einer 4 – 20mA Schnittstelle

Rafael Deliano

Diese Stromschleife wird im industriellen Bereich weiterhin häufig verwendet. Die Bürde gegen GND führt oft zu umfangreichen Schaltungen (Abb. 6) und die über Bauteilstreuung zu Fehlern. Hochauflösende A/D– und D/A–Wandler sind preiswert geworden und in Controllern ist EEPROM–Speicher verfügbar. Damit bietet sich der Abgleich in Software an.

Die 4–bis–20–mA–Stromschleife [1] zur analogen Übermittlung von Messwerten wurde Ende der 60er Jahre u. a. durch Honeywell populär. Sie signalisiert heute zudem auch Fehlerzustände mittels 2–mA– und 22–mA–Strömen.

Durch Strom wird man vom Kabelwiderstand unabhängig, aber der Störpegel begrenzt die Genauigkeit. Dafür gibt es keine gängige Spezifikation. 7 Bit (1%) waren wohl ehedem ausreichend und leicht erreichbar. 10 Bit hingegen (0,1%) nur unter Laborbedingungen.Die digitalen Feldbusse haben diese Einschränkung *analoger Übertragung* nicht, ihre Akzeptanz ist jedoch gering geblieben.



Sollwerte

4mA

$$s4 = 4095 * \frac{4}{22} = 745$$

20mA

$$s20 = 4095 * \frac{20}{22} = 3723$$

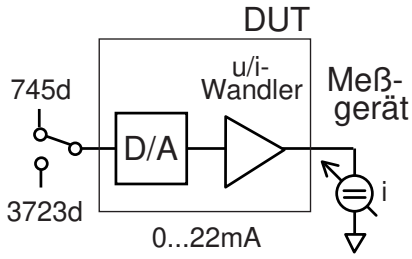
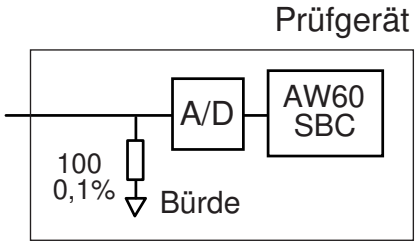


Abbildung 1: Sollwerte und Messwerte

Die Korrektur

Anders als für nichtlineare Sensoren genügt es hier, *Gain* und *Offset* zu berichtigen (Abb. 3). Also sind nur zwei 16–Bit–Worte im EEPROM nötig. Die Funktion ist für 12–Bit–A/D– und D/A–Wandler verwendbar. Wenn man auf eine vorhandene Routine für die 16x16–Bit–Multiplikation mit 32–Bit–Resultat zurückgreift, bietet sich für den Gain ein Defaultwert von 8000h an, der 1,0 entspricht. Entsprechend für den Offset 0000h für 0,0 . Letzteres ist eine 2er–Komplementzahl, kann also negativ sein.

Der Runtime–Code für die Formel ist simpel (Abb. 4, Listing 1). Allerdings wird danach Sättigungslogik benötigt. DurchF000 ANDist am Ergebnis leicht prüfbar, ob ein Überlauf eingetreten ist.Abhängig vom Vorzeichen der Konstante a ist auch leicht zu bestimmen, ob das Ergebnis auf 0FFF oder0000 korrigiert werden muss.



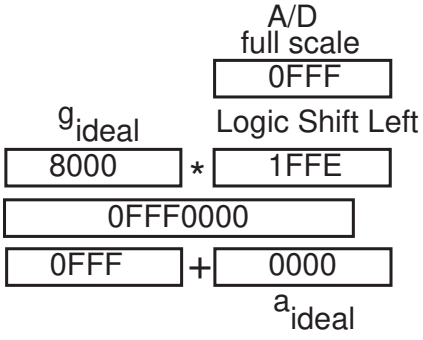


Abbildung 2: Runtime ohne Sättigungslogik

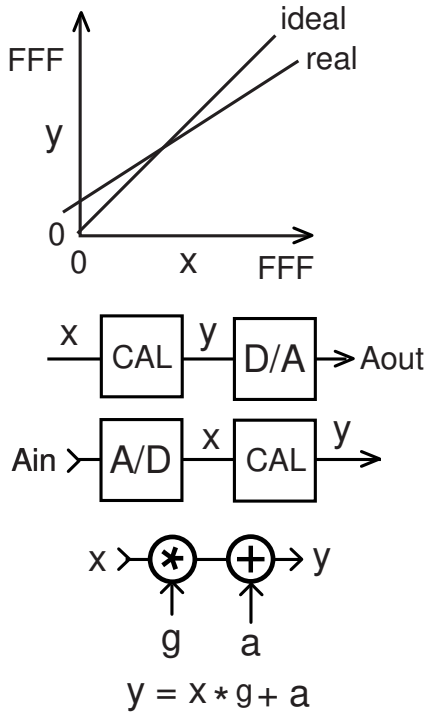


Abbildung 3: Korrektur

$$s4 = m4 * g + o$$
$$s20 = m20 * g + o$$
$$o'' = o/g$$
$$s4 = (m4 + o'') g$$
$$s20 = (m20 + o'') g$$
$$\frac{s4}{s20} = \frac{(m4 + o) \cancel{g}}{(m20 + o) \cancel{g}}$$
$$\frac{s4*m20 - m4*s20}{s20 - s4} = o''$$
$$\frac{s4}{m4 + o''} = g$$
$$o'' * g = o$$

Abbildung 4: Bestimmung der Formeln

Konstanten

Es sind zwei Testmessungen nötig, typisch bei 4,0 mA und 20,0 mA. Für einen Sender gibt man die Sollwerte aus und misst den sich tatsächlich ergebenden Strom (Abb. 6). Zwar kann man ein kalibriertes Messgerät verwenden, angenehmer ist es, den Wert automatisch ins Prüfgerät für den Funktionstest einzulesen. Dieses macht dann auch die Berechnung, schreibt ins EEPROM und verifiziert durch eine Kontrollmessung.

Bei linearer Korrektur ist eine direkte Bestimmung der beiden gesuchten Werte über Formeln möglich (Abb. 4, Listing 2).

Test

Hier wurde als simpler Zufallszahlengenerator der Programmspeicher von *nanoFORH* [1] verwendet. Genügt nur für derart einfache Anforderungen, da seine Werte keine gute Verteilung haben (Abb. 5).

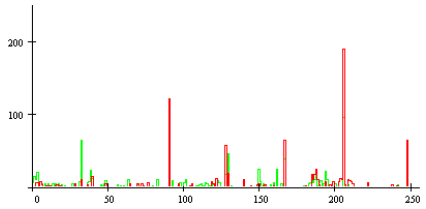


Abbildung 5: Verteilung Zufallszahlen

Die Bytes ergeben Streuwerte von +/-127 , die man auf die Sollwerte s4 und s20 addiert, um simulierte Messwerte m4 und m20 zu erhalten (Abb. 1, Listing 2: Test). Während in Gleitkomma die Korrektur exakt möglich ist, hat man in Festkomma einen kleinen, negativen Fehler (Abb. 8). Abschneiden statt runden ist schädlich.

Das konnte durch einen simplen Patch korrigiert werden. Wenn der Zielwert s4 nicht exakt erreicht wird, wird zum Offset+1 addiert. Dadurch trifft man das Ziel wieder mittig (Abb. 10). Bei s20 ist der Fehler unkritischer.

Temperatur

Die muss manchmal zusätzlich noch kompensiert werden. Z. B. ist die Stromquelle LM334 in Abb. 7 temperaturabhängig. Auch dann ist eine lineare Kennlinie wesentlich, um den Aufwand erträglich zu halten.

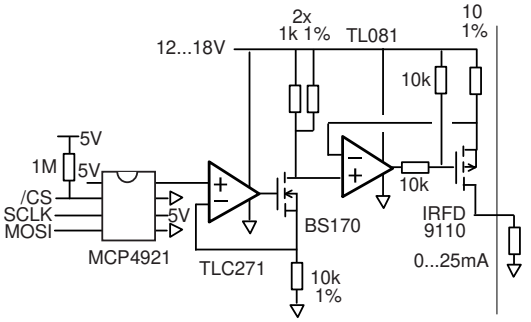


Abbildung 6: Schaltung Sender mit zwei Stromspiegeln

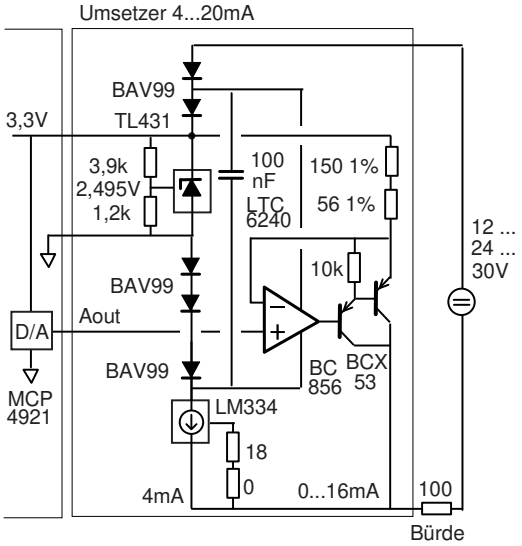


Abbildung 7: Sensor

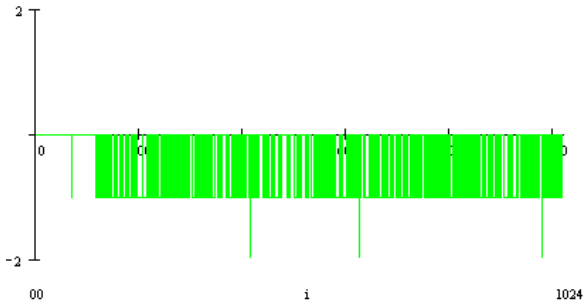


Abbildung 8: Fehler bei simulierter Streuung +/-127 auf m4

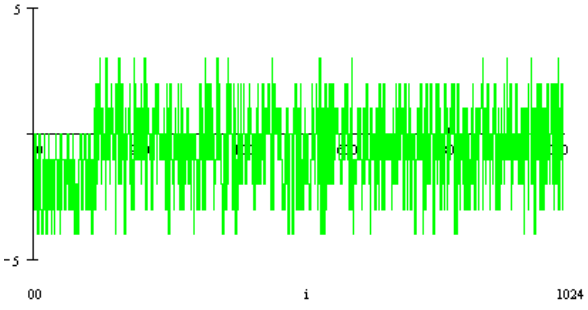


Abbildung 9: m20 ohne Patch

[1] Handbuch beim Autor erhältlich.

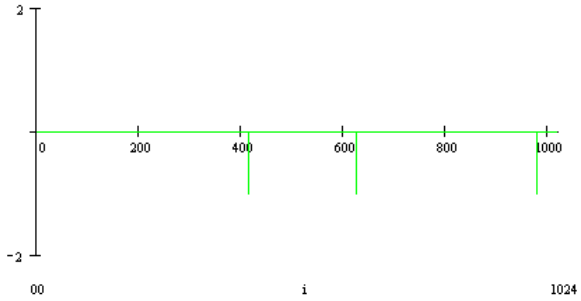


Abbildung 10: Fehlerrest aufm4 mit Patch

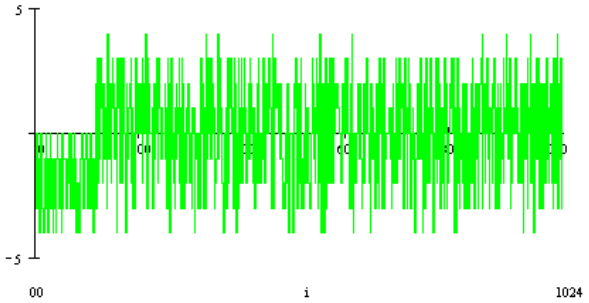


Abbildung 11: m20 mit Patch

Links

[1] Grundlagen z. B.: https://www.bapihvac.com/de/application_note/the-science-of-4-to-20-ma-current-loops-application-note/

Anhang

Da nicht jedem Leser die Typenbezeichnungen geläufig sein werden, hier eine kurze Aufstellung der Bauteile.

MCP4921/4922: 12–Bit DAC; with SPI Interface. TLC271: operational amplifier; combines a wide range of input offset voltage grades with low offset voltage drift and high input impedance. TL081: JFET input single operational amplifiers; incorporating well matched, high–voltage JFET and bipolar transistors, high–speed. BS170: MOSFET, N–Kanal, 60 V, 0,5 A, RDS(ON) 5,0 Ohm. IRFD9110: HEXFETs; schnelles Schalten. LM334: Einstellbare Konstantstromquelle; kann auch als Temperatursensor verwendet werden. TL431: Einstellbarer, präziser Shunt–Spannungsregler.LTC6240: Operationsverstärker für Präzisionsanwendungen wie Medizintechnik sowie Prüf– und Messgeräte.BAV99: Schaltdiodenarrays;niedrige Sperrverzögerungszeit für schnelles Schalten.

Listings

```
1   Listing 1: Runtime
2
3   : CAL       \ ( X --- Y )
4      1<SHIFT G% @ U* SWAP DROP A% @ + DUP F000 AND
5      IF DROP A% @ 8000 AND
6      IF 0 ELSE 03FF THEN
7   THEN ;
```

```
1   Listing 2: Berechnung der Konstanten
2
3   <| \ calc constants o & g
4
5   \ include
6   \ 1DSHIFT>    \ ( UD1 ---        UD2 ) LSR
7   \ D-          \ ( D1 D2 --- D3 )
8   \ DNEGATE     \ ( D1 --- D2 )
9
10  2 VARIABLE G%   \ UN1 8000 = 1,0
11  2 VARIABLE O%   \    N1
12
13  \ CAL \ ( UN1 --- UN2 )          UN1, UN2 = 0...3FF
14
15  D%  745 CONSTANT S4       D% 3724 CONSTANT S20
16  2 VARIABLE    M4   2 VARIABLE M20
17  2 VARIABLE SIGN
18
19  \ (s4*m20 - m4*s20)/s20-s4 = o"
20  : o"        \ ( --- N1 )
21  M20 @  S4 U*
22  M4   @ S20 U* D-
23  DUP SIGN ! DUP 8000 AND IF DNEGATE THEN
24  S20 S4 - U/
25  SIGN @ 8000 AND IF NEGATE THEN ;
26
27  \ s4/( m4 + o" ) = g
28
29  : G       \ ( o" --- UN1 )
30  m4 @ + 0 s4 1DSHIFT> ROT U/ ;
31
32  \ o = o" * g
33  : o       \ ( UN1 N1 --- N2 ) UN1 = g ; N1 = o"
34  DUP SIGN ! DUP 8000 AND IF NEGATE THEN
35  1<SHIFT U* SWAP DROP
36  SIGN @ 8000 AND IF NEGATE THEN ;
37
38  : G%!-O%!      \ (      --- )
39  o"      \ ( --- N1 )
40  DUP
41  G       \ ( o" --- UN1 )
42  DUP G% ! SWAP
43  o       \ ( UN1 N1 --- N2 ) UN1 = g ; N1 = o"
44  O% !
45  \ Patch
46  M4 @ CAL    S4 - 8000 AND IF 1 O% +! THEN
47  ;
48
49  \ D% 870 M4 ! D% 3552 M20 !
50  \ G%!-O%!
51  \ CR     M4 @ CAL ND.   S4 ND.
52  \ CR M20 @ CAL ND. S20 ND.
53  |>
54
55  <| \ Test with random numbers
56
57  : C->N \ ( C1 --- N1 ) extend sign
58  DUP 80 AND IF FF00 OR THEN ;
59
60  : TEST \ (         ---      )
61  C400 C000 DO CR
62  I              C@ C->N S4    + DUP ND.    M4 !
63  I 1000 + C@ C->N S20 + DUP ND. M20 !
64  G%!-O%!
65    M4 @ CAL ND.   S4 ND.
66  M20 @ CAL ND. S20 ND.
67  LOOP CR ;
68  |>
69
```

A small Forth Profiler

Philip Zembrod

Where does X spend its time? To answer this question I developed this profiler already 2 1/2 years ago, and talked about it at EuroForth and SVFIG, but I haven't written any of it down yet, so here goes:

It all started when I felt that my Small C compiler cc64, written in ITC [1] VolksForth on the C64, was slower than it should be. Of course, the well–known solution to slow programs is to optimise the hot spots. Only: Where were they in this case? I had some assumptions, of course, but, as Martin Fowler teaches us in "Refactoring", better measure, your assumptions are likely wrong. Mine were, mostly.

So, I wished for a profiler that could tell me at different levels — module group, module, word group, word — where cc64 spent how much time. And ideally, the thing should be self–hosted on 6502 VolksForth — I implicitly assumed that assembly code would be needed.

I did my due diligence and searched and asked for prior art. There was info on how to time individual words and advice to let NEXT log all words to stdout. I found one tool that tracked e2e time per word [2], but only e2e time, not net time spent in the word — which was what I was interested in.

So, my search came up essentially empty, giving me a perfect opportunity (or excuse, you decide!) to design and build something on my own. So I started to think. I thought, what could I do with NEXT ? I could

- count invocations of a given word.
- count NEXT cycles within a word, i.e. the number of instruction fetches at addresses between and ;.
- additionally sum up the time spent within a word.
- count NEXT cycles and sum up time within a range of words.
- split all of cc64's code into N ranges aka buckets and count NEXT cycles and sum up time per bucket.
- split a bucket into N sub–buckets, rinse & repeat.

When I had arrived at this point, after a number of good dog walks with time for thinking, I thought to myself: Wow, this might actually fly . . .

Further dog walks hashed out some design details:

- NEXT should remain reasonably fast, so I would only have buckets made of single address intervals, meaning one start and end address per bucket. I would have only 8 buckets, and they mustn't overlap, so a binary search with 3 compare instructions would find the right bucket, and I would unroll the 3–iteration search loop.
- I decided that NEXT cycles spent in Forth kernel words would be counted towards the cc64 word calling the kernel word. Reason: If a cc64 word called a slow Forth kernel word a lot, I would want to optimise the cc64 word, not the kernel word.
- I would also have a default bucket into which all NEXT cycles outside the 8 cc64 buckets and outside the Forth kernel would be put — the "rest" of the time in cases I drilled down into sub–buckets. Reason: I wanted the cycle and time sum of all buckets together to be roughly equal to the e2e time and cycle count of the measurement.
- For time measurement I would use 2 cascaded 16–bit timers in one of the C64's *MOS 6526 CIA*, running at the 1 MHz CPU clock speed, so I could measure microseconds.
- All the code for instrumenting NEXT would be put into macros, not subroutines, to get both best runtime performance and good testability [3]. I had quite a few bugs to chase in those macros — unit testing, like so often, was my friend.

b#	nextcounts	clockticks	name
0	475419	**52822277**	(etc)
1	1037243	114416784	[MEMMAN]
2	1384162	153154008	[FILE-HDL]
3	797224	**122822197**	[INPUT]
4	2695157	**299076306**	[SCANNER]
5	153639	17403250	[SYMTAB]
6	1826434	197679185	[PARSER]
7	1100491	121509788	[PASS2]
8	0	0	[SHELL]

Table 1: Module group buckets

So what does all this give us? Let's assume we've split up cc64's code into 8 buckets: memory handling, file handling, input, scanner, symbol table, parser incl. code-gen, pass2 and shell. I call them *module groups*, as most contain more than one module. Together they cover almost the entire compiler source. Let's also assume we

[1] ITC — indirect threaded code; DTC — direct threaded code.

[2] By end–to–end time or gross time spent in a word I mean simply the time from start to finish. In a word foo : foo bar ; this would include the time spent in bar. By net time I mean the time spent directly in foo, excluding the time spent in bar. So if bar is slow, I don't want foo to be shown as slow; there's nothing to optimise in foo. One exception: In : foo dup bar ; the time spent in the kernel word dup is counted to the net time of foo, since I don't want to optimise dup.

[3] Pointer into the code: The instrumented NEXT is called prNext, and it contains the 6 macros calcTime , findBucket , incCountOfBucket, addTimeToBucket, setPrevTime and incMainCount.

have an automated e2e test scenario that we can use for measuring. Here's the profiling report I got with a NEXT that is instrumented in the way described above. Tab. 1.

Bucket 0 named (etc) is the aforementioned default bucket. The profiler is only activated inside the compiler, hence the shell bucket (from where the compiler is invoked) ended up empty. Nextcounts are the NEXT cycles spent in each bucket, and clockticks are the corresponding 6502 CPU clock cycles.

This first measurement already told me 2 interesting things. First: The input module group has a reasonably low NEXT cycle count but unreasonably many CPU cycles. Turned out this was because I printed the compiled source to the screen while compiling. Apparently the character output routines in the C64 BIOS (known to CBMers as KERNAL) use a noticeable amount of CPU. Switching off listing while compiling got me to

3 313708 **34373231** [INPUT]

and a reduction in overall time of ~10%. 10% of time just spent printing on the screen! Who'd have thought?

The second finding is that the scanner module group is the top time sink. The scanner which just recognizes keywords, numbers, operators etc. is taking longer than the parser module group which handles the C grammar and also generates code. That doesn't seem right. Time to drill down into the scanner bucket with sub–buckets.

Buckets are defined with annotations profiler-bucket and end-bucket in the cc64 source code, prefixed by the conditional comment ?prof which comments out all profiler code in normal build mode. A bucket is a create structure containing (of course) its name and the start and end addresses of the code range represented by the bucket.

The 8 top–level buckets are defined in the top–level load file cc64.fth , and each contains one or a few module include statements. To drill down into the scanner, I placed 8 sub–buckets into scanner.fth , which of course all overlap with the main scanner bucket.

I now needed a way to activate only a selection of buckets, since the instrumented NEXT can only handle 8 buckets, and they mustn't overlap. So I created the concept of a *metric*.

A metric is a word containing a set of up to 8 buckets. Invoking a metric activates its buckets. Each measurement, i.e. each instrumented run of my e2e test suite, produces one metric. Metrics are defined with profiler-metric[and end with]profiler-metric . They must list non–overlapping buckets in ascending address range order; a check in]profiler-metric ensures that. The same cc64 binary can contain multiple metrics, so I only have to build it once for multiple measurements. Back to the long runtime of the scanner. The scanner metric yielded this result: Tab. 2.

b#	nextcounts	clockticks	name
0	6291642	691659740	(etc)
1	247516	29308255	[ALPHANUM]
2	66366	7040651	[ID]
3	1075233	**116814924**	[OPERATOR]
4	27320	3487498	[NUMBER]
5	237738	26065205	[CHR/STR]
6	150400	15851841	[NEXTWORD]
7	43874	5189985	[COMMENT]
8	844379	**95121698**	[REST]

Table 2: Scanner metric buckets

Again two surprises: The vast majority of time is spent scanning operators, and the bucket called rest spends almost as much.

The cause of the slow operator scanning turned out to be a set of nested loops that I wasn't aware of and that I eliminated because of this measurement. This shaved off another ~13% of the overall run time. And further drilling into the "rest" bucket showed that the time there was spent in a pair of words called nextword and backword. I realised: the parser was sending the scanner one token forward and backward all the time, until it found a matching rule and accepted the token. The solution was to replace them with a word pair thisword and accept. This refactoring of the scanner/parser interface was a bit of work on the parser side, but totally worth it: Now the parser can call thisword (which is super–cheap) until it can accept the current token. Though nextword and backword weren't particularly run time expensive per call, they were called so often that they did become expensive. The refactoring shaved off another 10% of overall run time.

Already pretty good improvements. I next decided to replace the colon definitions alpha? , num? and alphanum? with code words — the only assembly optimization I ended up needing. And I replaced my earlier blunt linear search in scanner string tables with a string–length indexed search. These two improvements together yielded 8% run time reduction. I felt I was already getting diminishing returns, so I decided to leave it at that.

My final profile looked like this: Tab. 3.

b#	nextcounts	clockticks	name
0	216506	22950222	(etc)
1	114310	12976436	[STRINGS]
2	742175	83029931	[MEMMAN]
3	1396517	154575151	[FILE-HDL]
4	316896	34740882	[INPUT]
5	832390	90869725	[SCANNER]
6	155294	17623890	[SYMTAB]
7	1858844	201608564	[PARSER]
8	1109737	122235615	[PASS2]

Table 3: Final profile buckets

This seems to be a much more reasonable time distribution than before. And a few very limited optimizations achieved 31% time reduction or 45% speed gain. Not a bad result, I think, not bad at all.

The current status of the profiler is that it is part of cc64. I will doubtlessly make it a part of 6502 VolksForth eventually. It could certainly be ported to VolksForth's other CPUs, and probably to most ITC and DTC Forths where NEXT is accessible to patching, e.g. for a debugger.

Links

All code is found at
https://github.com/pzembrod/cc64

The profiler itself: src/common/profiler.fth
Profiler unit test: tests/unit/profilertest.fth

Usage in cc64:
Main bucket definitions: src/cc64/cc64.fth
Scanner buckets: src/cc64/scanner.fth
Profiler start and end: src/cc64/invoke.fth

And tests/e2e/profile-register contains an archive of measurements.

EuroForth talk:
https://youtu.be/watch?v=TXKyTYI6b60

Listing

The core code inside NEXT with the unrolled binary search for the right bucket:

```
\ Compares instruction pointer IP to the start of the bucket indexed by
\      the X register, 0 <= X <= 8.
\ Result in carry and zero flag:
\      CC if IP < bucket start, CS if IP >= bucket start
\      EQ if IP = bucket start, NE if IP != bucket start
: compareIp
   IP 1+ lda      >buckets[ ,x cmp      0= ?[ IP lda        <buckets[ ,x cmp ]?
;


\ Finds the bucket into which current IP falls.
\ Result: bucket number * 4 in X register and in currentBucket.
: findBucket
   0 # ldx      compareIp   CC ?[     \ start of bucket 0 contains end of kernel
      currentBucket ldx
   ][ inx       compareIp   CC ?[
         dex    \ If IP < start of bucket 1 then return default bucket 0.
      ][
         5 # ldx
         compareIp   0<> ?[     CC ?[ dex dex ][ inx inx ]?
         compareIp   0<> ?[     CC ?[ dex          ][ inx          ]?
         compareIp                CC ?[ dex          ]?                    ]? ]?


         \ After finding bucket start, check against end of that bucket.
         \ If IP > end of that bucket, then return default bucket 0.
         IP 1+ lda      >]buckets ,x cmp      0= ?[ IP lda        <]buckets ,x cmp ]?
            CS ?[ 0 # ldx ]?


         txa    .a asl      .a asl      tax
      ]?
      currentBucket stx
   ]?
;
```

Rätsel: Was ist falsch im Code „21 Sticks“ ?

Michael Kalus

Das Rätsel gab mir chatGPT *neulich auf. Beim letzten Sommertreffen hatte ich es abends schon kurz vorgestellt und es gab erstauliche Stellungnahmen aus der Runde — technische, philosophische und politische. Aber macht euch selbst ein Bild. Bin gespannt auf eure Rückmeldungen.*

Zum Code

Der ist weiter unten als Original–Listing wiedergegeben, so wie ihn der Bot geliefert hatte, samt der Fragestellung. Ihr könnt es ja mal selbst probieren und vergleichen, was er euch wohl antwortet.

Der Code sieht tatsächlich sehr nach Forth aus. Feine Colon–Definitions, und alles sauber kommentiert und hübsch arrangiert — beispielhaft geradezu. Und eingegeben in Gforth, so wie er ist, compiliert das Stück sogar *fehlerfrei* !

Finde den Fehler im Listing

Aber der Code läuft *so* nicht, diegame-loopstürzt sofort ab mit der Meldung:

```
game-loop                    \ Start the game
21 Your turn. Enter number of sticks to pick (1-3):

:33: Stack underflow
>>>game-loop<<<              \ Start the game
Backtrace:
$7F538796B3B0 dup
$7F53879B4638 accept
$7F53879B4840 player-turn

:33: User interrupt
>>>game-loop<<<              \ Start the game
Backtrace:
$7F5387961F10 key-file
$7F5387983D68 (key)
$7F538796B348 xkey
$7F538796B400 edit-line
$7F538796BB20 accept
$7F538796B4D8 perform
  ok
```

Hm, was geht da schief? Habt ihr schon entdeckt, was da nicht stimmt?

Ich gebe euch auch die Regeln, nach denen gespielt / programmiert / werden sollte — von keinem geringerem als chatGPT höchstpersönlich formuliert. Und im Listing ist sein originaler Forth–Code.

Die Spielregeln

> Send a message: Tell me the rules of the game „21 sticks“.

„21 Sticks“, also known as „Nim“, is a two–player mathematical strategy game. The objective of the game is to avoid being the player who has to pick up the last stick. The game is played with a certain number of sticks,usually 21, arranged in a pile. Players take turns removing a certain number of sticks from the pile. The player who is forced to take the last stick loses.

Here are the rules for playing „21 Sticks“:

Setup: Start with a pile of 21 sticks.

Turns: Players take turns alternately.

Move: On each turn, a player must remove 1, 2, or 3 sticks from the pile. The player cannot remove more than 3 sticks or less than 1 stick in a single turn.

Objective: The goal of the game is to avoid being the player who takes the last stick from the pile. The player who takes the last stick loses the game.

Strategy: „21 Sticks“ is a mathematical strategy game. There’s a winning strategy for the first player (the one who goes first) if the initial number of sticks is not a multiple of 4. The winning strategy involves forcing the opponent into a losing position by always maintaining a multiple of 4 sticks in the pile after each of the opponent’s moves.

End of Game: The game ends when there are no more sticks left in the pile. The player who has to take the last stick loses the game.

Variations: The game can be played with a different initial number of sticks,such as 15 or 25, and the rules still apply. The key to winning is understanding the pattern and strategy of keeping the pile in multiples of 4.

The game’s strategy involves a bit of mathematical thinking and planning ahead. If you’re playing against an opponent who knows the winning strategy, it can be quite challenging to win unless you also understand the strategy.

Remember that „21 Sticks“ is a game with a clear mathematical solution, so if both players know the optimal strategy, the outcome will always be the same [ChatGPT].

Listing

```
1   anfrage war: "write code in forth that plays 21 sticks"
2
3   antwort:
4
5   variable sticks             \ Number of sticks remaining
6   21 sticks !                 \ Initialize with 21 sticks
7
8   : display-sticks            \ Display the current number of sticks
9     sticks @ . ;
10
11  : player-turn               \ Player's turn to pick sticks
12    ." Your turn. Enter number of sticks to pick (1-3): " cr
13    accept number ;
14
15  : computer-turn             \ Computer's turn to pick sticks
16    ." My turn. I'll pick " sticks @ 4 mod 1 + . ." sticks." cr
17    sticks @ 4 mod 1 + sticks ! ;
18
19  : game-loop                 \ Main game loop
20    begin
21      display-sticks
22      player-turn
23      sticks @ number min sticks !
24      sticks @ 0= if
25        ." You picked the last stick. You lose!" cr
26        exit
27      then
28      display-sticks
29      computer-turn
30      sticks @ 0= if
31        ." I picked the last stick. I lose!" cr
32        exit
33      then
34    again ;
35
36  game-loop                   \ Start the game
```

Resümee

Alles sehr erstaunlich, oder? Und was meint ihr, warum Forth für den Bot so schwer ist?

Viel Vergnügen beim Rätseln.

Die Gesetze des Raumfahrzeugdesigns

Dave Akin

Ich habe mich während meines gesamten Berufslebens mit der Entwicklung von Raumfahrzeugen und –systemen beschäftigt. Zehn Jahre am MIT und jetzt dreißig Jahre an der University Maryland.

1. Ingenieure arbeiten mit Zahlen. Eine Analyse ohne Zahlen ist nur eine Meinung.
2. Ein Raumfahrzeug perfekt zu entwerfen, erfordert unendlich viel Aufwand. Aus diesem Grund ist es eine gute Idee, es so zu bauen, dass es auch mit Fehlern noch funktioniert.
3. Entwicklung ist ein iterativer Prozess. Die erforderliche Anzahl an Iterationen ist um eins höher als die Zahl der Schritte, die man durchgeführt hat. Dies gilt zu jedem Zeitpunkt.
4. Ihre besten Bemühungen werden im endgültigen Entwurf unweigerlich nutzlos sein. Lernen Sie, mit der Enttäuschung zu leben.
5. (Millers Gesetz) Drei Punkte bestimmen eine Kurve.
6. (Mars Gesetz) Alles ist linear, wenn es auf doppelt logarithmischem Papier mit einem dicken Filzstift gezeichnet wird.
7. Die Person, die unbedingt Teamleiter sein möchte, ist am wenigsten dazu geeignet.
8. In der Natur liegt das Optimum fast immer irgendwo in der Mitte. Annahmen, dass es an den Rändern liegt, sollte man misstrauen.
9. Wenn nicht alle benötigten Informationen verfügbar sind, ist dies keine akzeptable Entschuldigung dafür, nicht mit der Analyse zu beginnen.
10. Schätzen Sie im Zweifelsfall. Raten Sie im Notfall. Aber räumen Sie später den Pfusch auf, sobald die tatsächlichen Zahlen verfügbar sind.
11. Manchmal ist der schnellste Weg zum Ziel, alles wegzuwerfen und nochmal anzufangen.
12. Es gibt nie eine einzige richtige Lösung. Es gibt aber immer mehrere Falsche.
13. Design basiert auf Anforderungen. Es gibt keine Rechtfertigung dafür, etwas zu entwerfen, das ein bisschen besser ist, als es die Spezifikation fordert.
14. (Edisons Gesetz) Besser ist der Feind von gut.
15. (Sheas Gesetz) Die Möglichkeit, ein Design zu verbessern, besteht hauptsächlich an den Schnittstellen. Dort ist auch der beste Ort, es zu verpfuschen.
16. Vorgänger, die eine ähnliche Analyse durchgeführt haben, hatten keinen direkten Zugang zu ewiger Weisheit. Es gibt daher keinen Grund, deren Analyse mehr Glauben zu schenken als Ihrer. Und es gibt insbesondere keinen Grund, deren Analyse als Ihre eigene auszugeben.
17. Die Tatsache, dass etwas publiziert worden ist, beweist nicht seine Richtigkeit.
18. Erfahrungen aus der Vergangenheit eignen sich hervorragend für eine Plausibilitätsprüfung. Zu viel derartiger Prüfung kann jedoch eine neue, lohnenswerte Entwicklung verhindern.
19. Die Chancen sind nicht hoch, dass Sie schlauer sind als alle anderen. Wenn Ihre Analyse behauptet, dass Ihre Endgeschwindigkeit doppelt so hoch ist wie die Lichtgeschwindigkeit, haben Sie eventuell den Warp–Antrieb erfunden. Aber die Wahrscheinlichkeit, dass Sie sich verrechnet haben, ist höher.
20. Ein schlechtes Design mit einer guten Präsentation ist irgendwann später zum Scheitern verurteilt. Ein gutes Design mit einer schlechten Präsentation ist sofort zum Scheitern verurteilt.

Abbildung 1: Chesley Bonestell; „Winged Rocket Ferry“ aus: „The Exploration of Mars“ 1956

21. (Larrabees Gesetz) Die Hälfte von allem, was man in einem Klassenzimmer hört, ist Müll. Bildung ermöglicht herauszufinden, welche Hälfte welche ist.
22. Im Zweifelsfall dokumentieren. (Die Aktivtäten für Dokumentation erreichen kurz nach Beendigung eines Programms ihr Maximum.)
23. Der Zeitplan, nach dem Sie entwickeln, wird Ihnen wie eine reine Fiktion erscheinen. Bis Ihr Kunde Sie feuert, weil Sie ihn nicht eingehalten haben.
24. Das Schema wird *Work Breakdown Structure* genannt, weil die verbleibende Arbeit anwächst, bis Sie einen Breakdown haben. Es sei denn, Sie erzwingen Struktur.
25. (Bowdens Gesetz) Nach einem misslungenen Test ist es immer möglich, die Analyse so zu verfeinern, dass bewiesen wird, dass die Toleranzgrenzen ohnehin immer negativ waren.
26. (Montemerlos Gesetz) Vermeide offensichtlichen Blödsinn.
27. (Varsis Gesetz) Zeitpläne bewegen sich nur in eine

Richtung.
28. (Rangers Gesetz) Es gibt nichts umsonst.
29. (von Tiesenhausens Gesetz der Programm–Planung) Für eine genaue Vorhersage des endgültigen Aufwands multiplizieren Sie die erste Zeitschätzung mit PI und schieben Sie bei der Kostenschätzung den Dezimalpunkt um eine Stelle nach rechts.
30. (von Tiesenhausens Gesetz des technischen Designs) Wenn Sie den größtmöglichen Einfluss auf den Entwurf eines neuen technischen Systems haben möchten, lernen Sie zeichnen. Am Ende entwerfen die Ingenieure das Raumschiff immer so,wie die Illustration der Werbeagentur aussah.
31. (Mo's Gesetz der evolutionären Entwicklung) Man kommt nicht zum Mond, indem man auf immer höhere Bäume klettert.
32. (Atkins Gesetz der Vorführungen) Wenn die Hardware perfekt funktioniert, erscheinen die wirklich wichtigen Besucher nicht.
33. (Pattons Gesetz der Programmplanung) Ein guter Plan, der heute gewaltsam umgesetzt wird, ist besser als ein perfekter Plan nächste Woche.
34. (Roosevelts Gesetz der Aufgabenplanung) Tun Sie, was Sie können, wo Sie sind, mit dem, was Sie haben.
35. (De Saint–Exupérys Gesetz des Designs) Der Entwickler weiß, wann er Perfektion erreicht hat: nicht wenn es nichts mehr hinzuzufügen gibt, sondern wenn es nichts mehr wegzunehmen gibt.
36. Jeder durchschnittliche Ingenieur kann etwas elegant entwerfen. Ein guter Ingenieur entwirft Systeme so, dass sie effizient sind. Ein großartiger Ingenieur entwirft sie effektiv.
37. (Henshaws Gesetz) Ein Schlüssel zum Erfolg einer Mission ist die klare Festlegung von Verantwortlichkeiten und Schuldigen.
38. Die technischen Möglichkeiten definieren die Anforderungen.Unabhängig davon,was in Lehrbüchern zu Systemtechnik geschrieben steht.
39. Jedes Forschungsprogramm,das zufällig eine neue Trägerrakete erfordert,ist de facto ein Trägerraketen-Programm.
39. (alternative Formulierung) Die drei Anforderungen, um ein neues bemanntes Raumfahrtprogramm erschwinglich und termingerecht zu halten:1. Keine neuen Trägerraketen. 2. Keine neuen Trägerraketen. 3. Was auch immer Sie tun, entwickeln Sie keine neuen Trägerraketen.
40. (McBryans Gesetz) Bevor es nicht funktioniert, können Sie es nicht verbessern.
41. Es findet sich nie genug Zeit, es richtig zu machen, aber irgendwie immer genug Zeit, es noch einmal zu machen.
42. Bevor das Entwicklungsprogramm startet, mangelt es an Budget. Danach mangelt es an Zeit.
43. Man versteht etwas erst richtig,wenn man es zum dritten Mal sieht. Oder wenn man es zum ersten Mal unterrichtet.
44. (Lachances Gesetz) Reichlich Zeit wird sehr schnell zu nicht genug Zeit.
45. Weltraum ist gnadenlos. Wenn man die Technik verpfuscht, stirbt jemand. Und es gibt keine teilweise Anerkennung, weil das meiste funktioniert hat.

Quelle: https://spacecraft.ssl.umd.edu/akins_laws.html

Dort stehen die Gesetzte im Original und eben in Englisch. Die Mühe der Übersetzung hat dankenswerterweise Rafael Deliano diesen Sommer auf sich genommen. Warum? Weil er mal Lust dazu hatte. Dass das so ohne Weiteres auch gedruckt werden darf, erlaubt Akin auf seiner Website gleich selbst, da wo er auch kurz erklärt, was es mit den Gesetzen so auf sich hat und wo die überall schon hingewandert sind:

> „Anyone is welcome to link to these, use them, post them, send me suggestions of additional laws, but I do maintain that this is the canonical set of Akin's Laws . . . "

Lieber Dave Akin, Danke für dein Werk.

Mein „Starting FORTH“

Rafael Deliano

Es ist zwar schon eine Weile her, aber noch klar, wie mein Kontakt zustande kam. Das Gymnasium endete absehbar irgendwann mit Abitur und ohne berufstaugliches Wissen. Wer aufmerksam war, konnte 1977 als das Jahr des Mikroprozessors wahrnehmen. Das sah nach Zukunft aus.

Wissen ergab sich damals wie heute daraus, dass man liest. Und, was man liest. Ich hatte ab 1980 die *Franzis Elektronik*, die monatliche dicke Fachzeitschrift der Industrie im Abo. Mit 120 DM/Jahr nicht billig, aber relevant, weil mit gutem Draht nach Kalifornien/USA. Dort fand ich, was ich suchte: Einen bezahlbaren Computer für Anwendung in der Elektronik. Der *AIM65* wurde Weihnachten 1980 gekauft. 1000 DM für eine Platine ohne Gehäuse und Netzteil! Aus Kostengründen wollte ich die Stromversorgung selber bauen. Da ich auch mit Schaltungstechnik bei Null anfing, bremste das erstmal. Das Zubehör hat sicherlich noch mal 500 DM gekostet. Das statische RAM wurde von 1 kByte auf 4 kByte aufgerüstet, indem man die 2114 in die vorhandenen Sockel steckte, 1024×4 Bit im 18–Pin–DIL.

Es war da ein *Microsoft BASIC* in zwei 4–kByte–ROMs vorhanden, das mit Mühe und PEEK und POKE die IO ansprechen konnte, am VIA (Versatile Interface Adapter) hinten im Erweiterungsstecker. Der 6502–Assembler in einem weiteren 4–kByte–ROM entsprach da eher den praktischen Anforderungen. Immerhin steuerte das Gymnasium leidlich gute Sprachkenntnisse bei — die Manuals waren ausführlich, aber natürlich in Englisch. Rockwell als *aerospace company* hatte da deutliche Ansprüche an seine Produkte. Über die Zeitschrift wurde ich Anfang 1981 informiert, dass es neue ROMs gab (Abb. 1), die ich dann auch für ca. 250 DM samt Manual beim Distributor bestellte. Das überarbeitete *figFORTH* enthielt bereits den FORTH–Assembler und kam eventuell bereits von Randy Dumse (†2020), der damals Entwickler bei Rockwell war. Er implementierte später das *R65F11 RSC–FORTH*. Und dann in seiner eigenen Firma *New Micros Inc* den 68HC11 mit FORTH im ROM. Seine Abneigung gegen Public–Domain–Software war ausgeprägt. Mein Eindruck von FORTH war damals positiv, weil ich eine *Implementierung in Industriequalität* erhielt. Die dann eben kostete, aber ideal für die Ansteuerung von Elektronik geeignet war. Im gleichen Jahr liefen in der Franzis *Elektronik* mehrteilige Artikelserien zu Pascal und Pearl, der deutschen Echtzeitsprache für Minicomputer. Beide warteten seit Mitte der 70er auf ihre große Stunde. Die würden nach Einschätzung der Redaktion relevant werden. Es wurde nur sporadisch, aber wohlwollend über FORTH berichtet. Diese Sprache galt als neu, „cutting edge“ und speziell für industrielle Anwendung von Mikroprozessoren.

Was ist davon 2023 noch relevant? Für den Anwender ist das abstrakte Konzept einer Programmiersprache nicht wichtig, er entscheidet sich anhand der konkreten Implementierung. Wer Qualität will, wird bereit sein, Kosten dafür zu akzeptieren. Mainstream à la BASIC oder C kann „für alles und jeden“ beworben werden. Alles andere ist Nische und für Anwender nur interessant, wenn es seine Anforderungen besser abdeckt. Werbung erfolgt über konventionelle Kanäle wie auflagenstarke Zeitschriften oder Messen. Viel Verständnis ist bei Redaktionen und Publikum da aber nicht zu erwarten, wenn die Außendarstellung eben nicht stimmig ist.

FORTH:
Höhere Programmiersprache für AIM-65

FORTH ist eine erweiterbare höhere Programmiersprache, die sich speziell für Regelungsanwendungen eignet. Sie wird resident in einem Festwertspeicher für das Mikrocomputersystem AIM 65 von der Firma Rockwell angeboten. FORTH kann vom Anwender leicht erweitert werden, um neue anwenderbezogene Befehle zu kreieren. Programme können dann in dieser neuen Sprache geschrieben werden, so daß Ingenieure ohne Programmiererfahrung einen Mikrocomputer effizient einsetzen können.

Die Anwendungen schließen Industriesteuerungen, Videografik und periphere Schnittstellen ein. FORTH ist eine in hohem Maß interaktive Sprache, die einen Compiler, Testaufbereitungsprogramm und einen Interpreter mit direkter Befehlsausführung enthält. Durch den interaktiven Aufbau ermöglicht FORTH die sofortige Programmfehlersuche und verringert dadurch den Programmieraufwand.

AIM-65-FORTH ist in zwei ROMs mit 4 KByte enthalten, die direkt auf die Platine aufgesteckt werden können.

Abbildung 1: Die damalige Anzeige in der *Elektronik*.

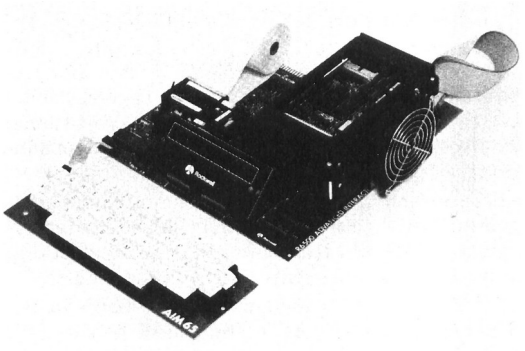


Abbildung 2: Mit leistungsfähiger Software kann der Mikrocomputer AIM 65 für Steuerungsaufgaben verwendet werden. (Original Slogan)

Forth–Gruppen regional

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

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

München **Bernd Paysan**
Tel.: (0 89) – 41 15 46 53
bernd@net 2o. de
Treffen: Jeden 4. Donnerstag im Monat um 19:00 auf http://public.senfcall.de/forth-muenchen , Passwort over+swap.

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

Ruhrgebiet **Carsten Strotmann**
ruhrpott-forth@strotmann. de
Derzeit keine Treffen.

Dienste der Forth–Gesellschaft

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

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

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

µP–Controller–Verleih **Carsten Strotmann**
microcontroller verleih@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

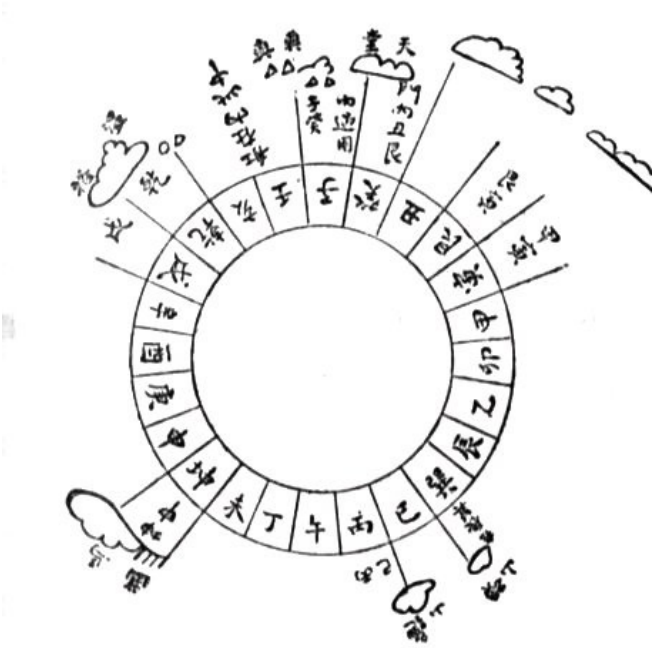
Termine

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*

Jeder 1. Montag im Monat ab 20:30 Uhr
Forth–Abend
Videotreffen (nicht nur) für Forthanfänger
Info und Teilnahmelink: E–Mail an wost@ewost.de

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

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.

EuroForth 2023

Anton Ertl

Es war schön, endlich wieder einmal die Leute persönlich zu sehen, die wir seit 2019 nur mehr online getroffen hatten, auch wenn weniger Leute gekommen waren als erwartet wurden. Es gab aber auch einige neue Gesichter. Auch inhaltlich waren die persönlichen Gespräche hilfreich, während bei Online–Meetings die Diskussionen in der Kaffeepause oft verlorengehen bzw. eben nicht in der Form stattfinden, so sehr wir uns auch bemüht hatten.

Touristisches

Die EuroForth fand diesmal bei prächtigem Wetter in Ariccia in der Nähe von Rom statt. Ariccia is bekannt für Porchetta, und obwohl mir Schweinsbraten zuhause nicht zusagt, fand ich die Porchetta–Variante köstlich. Auch sonst kann man dort gut essen, die Stadt ist hübsch, und bietet gute Aussichten.

Die Via Appia war eine der ältesten und wichtigsten Straßen der römischen Republik (erbaut ab 312 v. C.). Die moderne Variante davon folgt immer noch zum guten Teil der alten Strecke und führt durch *Ariccia*. Unser Hotel („Aricia“ mit einem c) liegt an dieser Straße.

Ein paar km von Ariccia ist *Castel Gandolfo*, die Sommerresidenz des Papstes, und einige von uns haben am Sonntag nach der Konferenz einen Spaziergang dorthin und zurück gemacht.

Als „social event“ sind wir am Samstag Nachmittag nach Rom gefahren und haben eine Führung durch das Forum Romanum und das Collosseum erlebt.

Vorträge

- Nick Nelson erklärte, wie sich der Zugriff auf *Oracle* vom Zugriff auf *MySQL* unterscheidet (sehr; beides SQL–Datenbanken), und wie er das Problem gelöst hat.
- Ich sprach über eine neue Optimierung in Gforth, nämlich den *instruction pointer* (IP) der virtuellen Maschine nicht bei jedem Primitive auf dem aktuellen Stand zu halten, sondern nur dann, wenn es notwendig ist. Bei einigen Benchmarks bringt das auf modernen Prozessoren erstaunlich viel.
- Glyn Faulkner zeigte den Forth–Generator *4g*, der aus dem Quellcode für Primitives in der *Sprache F.A.I.L.* den Assemblercode für die Primitives erzeugt, wobei man die Threading–Methode, die Registerbelegung und die Dictionary–Struktur über Parameter angeben kann.
- Gerald Wodni berichtete über das neue *Terminal User Interface* (TUI) von VFX Forth.
- Francois Laagel schrieb ein Programm, um Hexadokus (Sudokus mit Hexadezimalziffern und 4x4–Raster) zu lösen. Für das Debugging dieses Programms verwendete er unter anderem stack digests: Eine Art Quersumme (er verwendet SHA1) über den gesamten Datenstack. Auf diese Weise stellt er fest, ob beim Backtracking tatsächlich der ursprüngliche Zustand wiederhergestellt wurde.
- Ulrich Hoffmann zeigte eine Implementierung von value–flavoured structures: Auf die Felder dieser Strukturen greift man ähnlich zu wie auf value s.
- Bill Stoddart präsentierte eine formale Semantik einer Forth–Variante mit Backtracking.
- In seinem zweiten Vortrag zeigte Nick Nelson eine Syntax für das vom C–Feature enum inspirierten Konstrukt enum< ... >> , wo er durch den Einsatz eines Recognizers vermeidet, für jeden definierten Wert ein Definitionswort aufrufen zu müssen.
- In meinem zweiten Vortrag ging es nicht um Forth im Speziellen, sondern um den Hardware–Bug *Spectre*. Ich erklärte, warum dieser Bug in Hardware korrigiert werden muss, statt auf Software–Mitigations zu setzen, und wie das ohne nennenswerte Performance–Einbußen gemacht werden kann.
- Ulrich Hoffmann ließ sich von unserem Konferenzort Rom zu einer *lateinischen Variante von Forth* inspirieren, die natürlich auch die Eingabe von Zahlen in römischer Schreibweise unterstützt (mit Hilfe eines Recognizers).
- Ich erzählte über die Lösung des Magischen Sechsecks mit Hilfe von *Constraints*. Wem das bekannt vorkommt: Ich habe darüber schon bei der Forth–Tagung gesprochen. Seither ist allerdings noch eine Variante mit stärkeren Constraints dazugekommen.

Mehr dazu

Wer mehr wissen will, findet derzeit die vorläufigen Proceedings auf http://www.euroforth.org/ef23/papers/. Die endgültigen Proceedings (mit Vortragsfolien, Videos und teilweise mit überarbeiteten Papers) werden an der selben Stelle erscheinen, wenn sie fertig sind.