Vierte Dimension

3/2021

Das Forth–Magazin

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

In dieser Ausgabe:

Dienstleistungen und Produkte

tematik GmbH
Technische
Informatik

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

Titelbild: Tomatensamen im Vergeich (Ausschnitt). Autor mk
Quelle: Irgendwo aus dem Internet und selbst modifiziert.

Seite 11: Comic. Autor xkcd
Quelle: https://xkcd.com/ — Abdruck genehmigt von Casey Blair, xkcd–Assistant, am 3. September 2021 (E–Mail).

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

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

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00 € + Porto u. Verpackung

Liebe Leser,

Regional hier bei mir sieht es in Punkto „Werkeln–mit–Forth“ grad so aus: *Das Labor* in Bochum, wo man löten, messen, schwatzen konnte — schon wieder geschlossen. Das *Unperfekthaus* in Essen mit dem Forthtreffen — immer noch zu. Und bei euch in den Regionen, wie ist es dort?

Tja, und unsere diesjährige Forthtagung wird auch schon wieder eine reine Videokonferenz sein. Bin gespannt, wie die Vorstandswahl ablaufen soll. Und unser Drachenrat. Denn *SWAP* wohnt ja immer noch bei mir ...

Um so erfreulicher war, dass ihr fleißig Beiträge eingereicht habt. Das Heft füllte sich wieder recht flott.

Christof Eberspächer gibt uns Einblick in seine Forschung im Umgang mit dem kostbaren Süßwasser und zeigt, wie er die Vorteile aus beiden Welten, Forth und C/C++ im Arduino, zusammengebracht hat.

Und Erich Wälde hat Wort gehalten. Teil II seines Weges ins Internet der Dinge, hier in den Garten, ist da. Und natürlich kommen dabei die Funkamateure auf ihre Kosten. In den Garten legt man keinen Draht. Was mich daran erinnerte, dass der Maulwurf bei mir wieder als Drainagehelfer arbeitet. So einen Sensor für dessen Grabtätigkeiten hätte ich gern.

Überrascht hat mich Jens Storjohan mit seinem Beitrag. Gar nicht so einfach, Vektoren in Forth zu formulieren, will man mit Permutationen spielen. Da muss man seine (große) Maschine schon gut kennen, will man das mit großen Zahlen machen. Und eine bewährte Syntax dafür, gibt es die in Forth überhaupt?

Auch gut kennen muss man seine (kleine) Maschine, um deren Blinken dem Auge angenehmer zu machen. Mit ihrem Ausflug in die Welt der *Differentialgleichungen* haben Robert Clausecker und Matthias Koch dazu Grundlegendes gezeigt. Und das verbraucht nicht einmal viel Platz in einer MCU, auf diese Weise LEDs blinken zu lassen.

Carsten Strotmann und Philip Zembrod haben sich die Arbeit geteilt und so dafür gesorgt, dass das komfortable *VolksForth* schon mal da ist, wenn die Retro–Fangemeinde ihren *X16* verwirklicht. Und weil der Platz da knapp ist, wird auf dem `tmpheap` kompiliert — Erinnert mich an den AIM65 damals. Da war Forth auch schon da, als der noch mit Basic in ROMs ausgeliefert wurde. Die ROMs konnte man tauschen, Basic raus, RSC–Forth rein, schon bootete man ins `ok`.

So, und damit es zuhause an den Bildschrimen in diesen Zeiten nicht allzu langweilig wird, hat James netterweise für *Gforth* ein hübsches `SNAKE` spendiert. Selbstverständlich WASD–gesteuert.

`https://github.com/howerj/snake/blob/master/snake.fth`

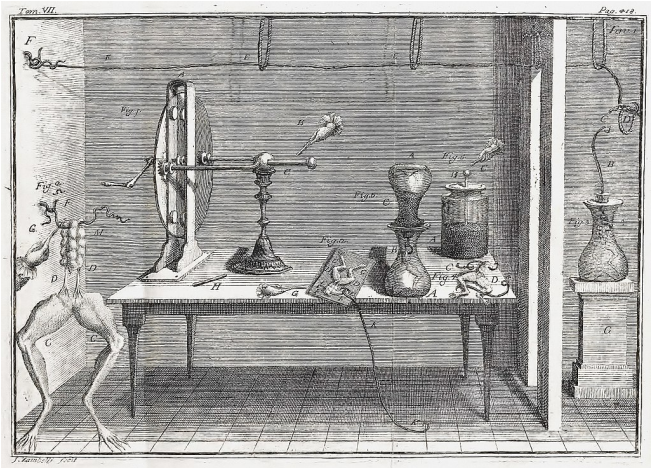
Bis bald, euer Michael

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-2021-03`

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

Ulrich Hoffmann
Bernd Paysan
Carsten Strotmann

Kontakt: `Direktorium@Forth-ev.de`



Grundlegende Experimente

Am 6. November 1780 seziert Galvani die Schenkel eines Frosches. Er notiert:

> „Als einer meiner Leute mit der Spitze des Skalpells die Schenkelnerven ganz leicht berührte, schienen sich alle Muskeln wiederholt derart zusammenzuziehen, als wären sie von heftigen Kräften geschüttelt.“ [3]

Im Jahre 1792 erfuhr Volta von den Frosch–Experimenten des angesehenen Anatomen Luigi Galvani, die dieser auf animalische Elektrizität zurückführte. Volta erkannte aber die Ursache der Muskelzuckungen in äußeren Spannungen, und es entsprang ein Streit um den Galvanismus, der die Wissenschaftler in ganz Europa in Lager teilte. Für Galvani lag die Erklärung darin, dass der Frosch eine Art Leidener Flasche (also ein Kondensator) war, für Volta war er nur eine Art Detektor. [1]

Volta gab nicht auf. Seine langjährigen Untersuchungen zur Kontaktelektrizität mündeten in der Erfindung von Batterien. Und im Januar 1800 schreib er dann einen Brief an Banks. [2]

> „Nach langem Schweigen, für das ich mich nicht entschuldigen will, habe ich das Vergnügen, Ihnen, Sir, und durch Sie der Royal Society einige bemerkenswerte Ergebnisse mitzuteilen, zu denen ich bei der Fortsetzung meiner Experimente mit durch einfachen Kontakt angeregter Elektrizität gelangt bin. Metalle verschiedener Art und sogar durch die anderer Leiter, die auch voneinander verschieden sind, entweder flüssig oder die etwas feucht sind, wodurch sie leitend werden. Das Wesentliche dieser Resultate, das fast alle anderen einschließt, ist die Konstruktion eines Apparates, der hinsichtlich der Wirkungen, d. h. hinsichtlich der Zuckungen, die er im Arm verursachen kann usw. den Leydener Flaschen ähnlich ist, … die dennoch unaufhörlich wirkten und deren Ladung sich nach jeder Explosion von selbst erholt. …“

Volta schwärmt von den Vorzügen seiner Batterien.

> „Wer würde nicht, mit einem Wort, eine unzerstörbare Ladung, eine Wirkung auf die elektrische Flüssigkeit oder einen ständigen Impuls genießen wollen?“

Dass seine Stapel aus Kupfer– und Zinkplatten, mit salzwassernassen Lederlappen dazwischen, dazu führen würden, dass wir Smartphones betreiben, konnte er vor 200 Jahren nicht ahnen! Philosophie ist wichtig.

Quellen

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

[2] „On the electricity excited by the mere contact of conducting substances of different kinds.“ In a letter from Mr. Alexander Volta, F. R. S. Professor of Natural Philosophy in the University of Pavia, to the Rt. Hon. Sir Joseph Banks, Bart. K.B. P. R. S ; Veröffentlicht am 01. Januar 1800. (Aus dem Französischen ins Englische übersetzt und von dort ins Deutsche; mk) `https://royalsocietypublishing.org/doi/10.1098/rstl.1800.0018`

[3] `https://www1.wdr.de/stichtag/stichtag-galvani-froschschenkel-100.html`

mk

Präfix, Infix, Postfix — `number` im Mecrisp

Ihr Lieben!

Das `number` vom Mecrisp ist schon ziemlich ausgefuchst! Da ist alles möglich. Damit kann bei einer Zahleneingabe sogar mitten in der Zahl die Basis gewechselt werden:

```
%111#78$ff%1100
```

Bei negativen Zahlen ist es außerdem egal, wo das Minuszeichen steht. Es kann am Anfang, am Ende oder mitten in der Zahl stehen:

```
-%111#78$ff%1100
 %111#78$ff%1100-
%111#78-$ff%1100
```

Die Affixe[1] `$ % #` ändern die Zahlenbasis natürlich nicht rückwirkend! Und es gilt auch: Sie dürfen beliebig oft in der Zahl vorkommen.

```
-%111--#78$$$$ff%%%1100
```

Das gilt auch für den Punkt, durch den doppelt genaue Zahlen resultieren … Gruß, Martin

…

Hallo Martin et al.,

ja, da ist der Matthias Koch sehr freizügig mit den Syntaxregeln zur Zahlenbildung umgegangen. Ich habe ihn vor vielen Monden danach befragt, und wenn ich mich noch recht erinnere, findet er das in Ordnung und möchte es auch so lassen.[2] Frei nach dem Motto, den

[1] So ein Affix kann ein Prä–, In–, oder Postfix sein. `https://de.wikipedia.org/wiki/Affix_(Naturwissenschaften/Informatik)`

[2] Hab bei ihm nachgefragt: Ja, es bleibt dabei.

Forth–Programmierer mit möglichst wenig Regeln zu beschränken.

Hier, zusätzlich zu Martins Beispielen, noch ein paar mehr aus der Trickkiste von `number` :

```
(1) $   %   ---   -$-
(2) .--..-...---...-
    $.....$
    ##-##-##...##-##-##
```

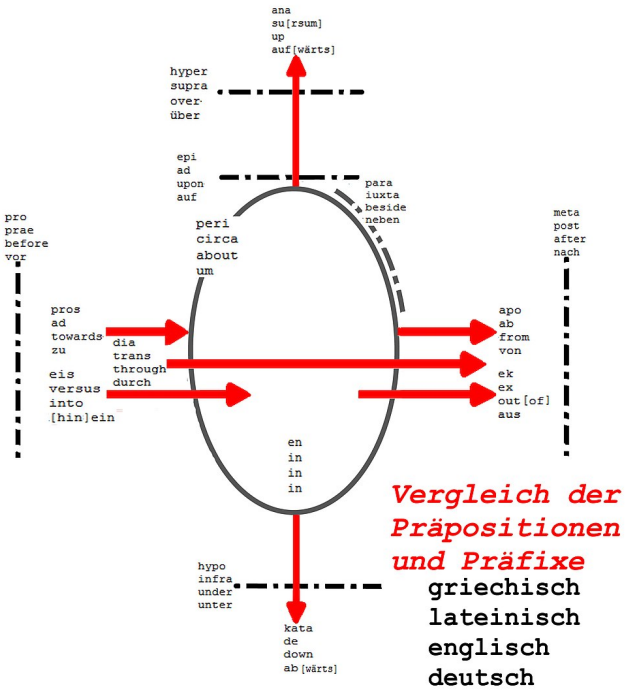
Die Beispiele aus (1) legen je eine Null auf den Stack, die aus (2) jeweils zwei. Warum das geht? Na, weil es `number` auch völlig egal ist, ob überhaupt eine Ziffer in der Zahl vorkommt. :-) ... Viele Grüße, Wolfgang

Quelle

Im Juli des Jahres aus der permanenten Konversation der rührigen Montags–Forth–Gruppe bei Senfcall gefischt, die Wolfgang Strauss organisiert, und druckbar aufbereitet.

`https://sourceforge.net/projects/mecrisp/files/mecrisp-quintus-0.36-experimental.tar.gz/download`

mk



(Bild: Von GRIPS — Eigenes Werk, CC BY-SA 4.0, `https://commons.wikimedia.org/w/index.php?curid=63444790`)

Präfixe, Spezifizierer — Partikel in Forth

Ab und zu sieht man statt eines Präfixes, das ja direkt an einer Zahl dran ist, auch ein vorangestelltes Wörtchen, das die Basis der folgenden Zahl festlegt. Streng genommen sind das Partikel, also Worte, die für sich genommen keinen Sinn ergeben, aber einem nachfolgenden Wort einen Sinn verleihen.

noForth

So im *noForth*. Doch heißen sie dort einfach nur „Präfixe“ und werden damit sprachlich nicht von den „echten“ Präfixen unterschieden. Beschrieben sind sie dort so:

> „Prefixes are incomplete words. They become a complete word in combination with the immediately following word or text in the input stream. Prefixes are input tools. They read the input stream, both compiling and interpreting. They are not compiled.“ [1]

noForth unterscheidet mehrere Arten solcher Partikel alias „Präfixe“.

Base prefixes `HX DM` and `BN` cause a temporary base–change only while the next word in the input stream is being executed or compiled.

Double number prefix `DN` makes double number input possible, both compiling and interpreting

Value prefixes `TO +TO INCR ADR`

Character prefix `CH` can be used always when the character immediately follows.

Hier die Gebrauchsbeispiele dazu, auch entnommen aus [1] — übrigens eine exzellente Dokumentation des noForth.

```
hx 10 . ↩ 16 OK
: HUNDRED hx 64 ;
hundred . ↩ 100 OK

dn 13579753 d. ↩ 13579753 OK

value KM
3 to km  km . ↩ 3 OK
4 +TO km km . ↩ 7 OK
INCR km  km . ↩ 8 OK
ADR km    @ . ↩ 8 OK

ch A . ↩ 65 OK
: .... key dup ch ? = if ... ;
```

Gforth

Auch Gforth verwendet Partikel in solcher Weise. Da sehen wir es bei der Zuweisung von Zahlentypen zu *Locals*. Auch da gilt, dass der Partikel selbst *nicht* compiliert wird, sondern nur das Ergebnis.

Bekanntlich kann man Locals ja so definieren [2] :

```
{ local1 local2 ... -- comment }
```

Oder so:

```
{ local1 local2 ... }
```

Also mit Hilfe der geschweiften Klammern, was sehr praktisch ist. Ein Beispiel aus der Dokumentation illustriert das:

```
: max { n1 n2 -- n3 }
n1 n2 > if n1 else n2 endif ;
```

„The similarity of locals definitions with stack comments is intended. A locals definition often replaces the stack comment of a word. The order of the locals corresponds to the order in a stack comment and everything after the -- is really a comment. ... The name of the local may be preceded by a type specifier, e.g., `F:` for a floating point value. ... Gforth currently supports cells (`W:`, `W^`), doubles (`D:`, `D^`), floats (`F:`, `F^`) and characters (`C:`, `C^`) in two flavours: a value–flavoured local (defined with `W:`, `D:` etc.) produces its value and can be changed with `TO`. A variable–flavoured local (defined with `W^` etc.) produces its address (which becomes invalid when the variable's scope is left) ...“ [2]

Gforth kennt demnach *zwei verschiedene Sorten* von Locals. Die einen benehmen sich wie *Values*, die anderen wie *Variablen*. Beide können entweder zu Cell–, Double–, Float– oder Character–Typen gemacht werden durch *specifier* genannte Partikel. Default ist der Typ local cell value, er braucht keinen Spezifizierer. So ist z. B. `{ x0 }` und `{ W: x0 }` eine gleichwertige Definition des Values `x0`. Diese Art und Weise, Locals zu machen, stammt aus TILE[3].

values	W:	D:	F:	C:
variables	W^	D^	F^	C^

Tabelle 1: „Specifier“ der Locals im Gforth

Es ist wohl so, dass Sprachen ohne Präpositionen[4] nicht auskommen können.

Quellen

[1] `noforth documentation.pdf` `https://home.hccnet.nl/anij/nof/noforth.html`

[2] `gforth-0.7.0.pdf`, S. 131 `https://gforth.org/`

mk

Von Typografen und Typisten

Dem aufmerksamem Leser wird aufgefallen sein, dass in diesem Heft entgegen unserer typografischen Tradition, die wir nun schon seit 2006 pflegen, einer der Beiträge ein klein wenig anders gesetzt worden ist als die anderen — auf Wunsch des Autors.

Dabei ging es um die Striche. Die Gedanken–, Binde– und Trennstriche. Der traditionelle typografische Stil[5] benutzt dafür Geviert–, Halbgeviert– und Viertelgeviertstriche, um diese darzustellen. Der kürzeste von allen, der Trennstrich, kommt dabei zum Einsatz am Rande des Textes bei der Silbentrennung und soll da die Randoptik möglichst wenig stören. Der etwas längere, der Divis, teilt die Worte bei zusammengesetzten Begriffen, sodass sie nicht zu einem seltsamen Wurm verschmelzen (C–Dur, Konrad–Adenauer–Straße ...). Und der Geviertstrich schließlich ist ein waagerechter Strich, dessen Breite der Geviertlänge entspricht — das war einmal ein quadratisches Stückchen Blindmaterial im Bleisatz, dessen Kantenlängen der Schriftgröße entsprach.

Abbildung 1: Das Geviert im Bleisatz

Bernd Paysan, unser Typograf und Hüter der Tradition, meinte dazu:

„Da geht's um Ästhetik ... die Einschränkungen des Computersatzes [in den Anfängen] und der daraus resultierende ästhetische Verlust haben übrigens zur Entwicklung von TEX geführt, weil Donald Knuth mit der Situation so unzufrieden war, dass er das Problem selbst lösen wollte.“

Und so machen wir das, seit wir die technischen Möglichkeiten dafür wiedererlangt haben, eben besser als die gängigen Zeitungen. Wir benutzen TEX und schreiben daher in LYX — der besseren Textverarbeitung. Da kann ich — als der Typist — einfach ein `-` für den Trennstrich, zwei `--` für den Bindestrich, und für den Gedankenstrich drei `---` mit Leerzeichen drum herum eintippen. Und Lyx macht mir daraus dann automatisch das typografisch richtige Zeichen. Bernd:

„Eben, bei LYX kriegst du das auch als Typist hin, während in normalen Textverarbeitungen diese Zeichen einfach nicht erreichbar sind. Und dann halt auch nicht mehr benutzt werden. Und um diese Unfähigkeit zu kompensieren, erklärt man nun das aufgrund der technischen Beschränkungen geänderte Layout zum neuen Standard — ohne mich.“

Die Recherche ergab, dass sich die Abkehr von der guten alten Strichelei der Typografen schon arg verbreitet hat. *Typografie.info* behauptet von sich „Die meistbesuchte deutschsprachige Typografie–Website — seit 20 Jahren online“ zu sein. Und hat den Geviertstrich im Deutschen praktisch eliminiert. Da sind Trenn– und Binde–Strich eins geworden. ...

*** Fortsetzung auf Seite 11

[3] THREADED INTERPRETIVE LANGUAGE ENVIRONMENT (TILE) FORTH; 1990, Mikael R.K. Patel. `https://github.com/cstrotm/tile-forth`

[4] Von Lateinisch: praepositio „Voranstellung“

[5] Die Typografie wird unterteilt in Mikrotypografie und Makrotypografie. Die Arbeit des Typografen besteht darin, beide Gestaltungsmerkmale in geeigneter Weise zu kombinieren. `https://de.wikipedia.org/wiki/Typografie`

Forth & Arduino messen Wasserbedarf von Tomatenpflanzen

Christof Eberspächer

In der Anwendung eines Mikrocontrollers für ein vom Programmierer selbst durchgeführtes Experiment zum Wasserbedarf von Tomatenpflanzen ist der interaktive Zugriff, den ein Forthsystem ermöglicht, besonders nützlich. Auf der anderen Seite werden über die Verwendung von Routinen, die andere geschrieben haben, Funktionalitäten zugänglich, die ein reines Forthsystem kaum bieten kann. Die Arduino–Umgebung bietet hier eine Fülle von Libraries unter anderem für den Betrieb in einem WLAN. Ein in Arduino–C/C++ geschriebenes Forth kann die Vorteile beider Welten zusammenführen. Bild 1 zeigt den Aufbau schematisch.

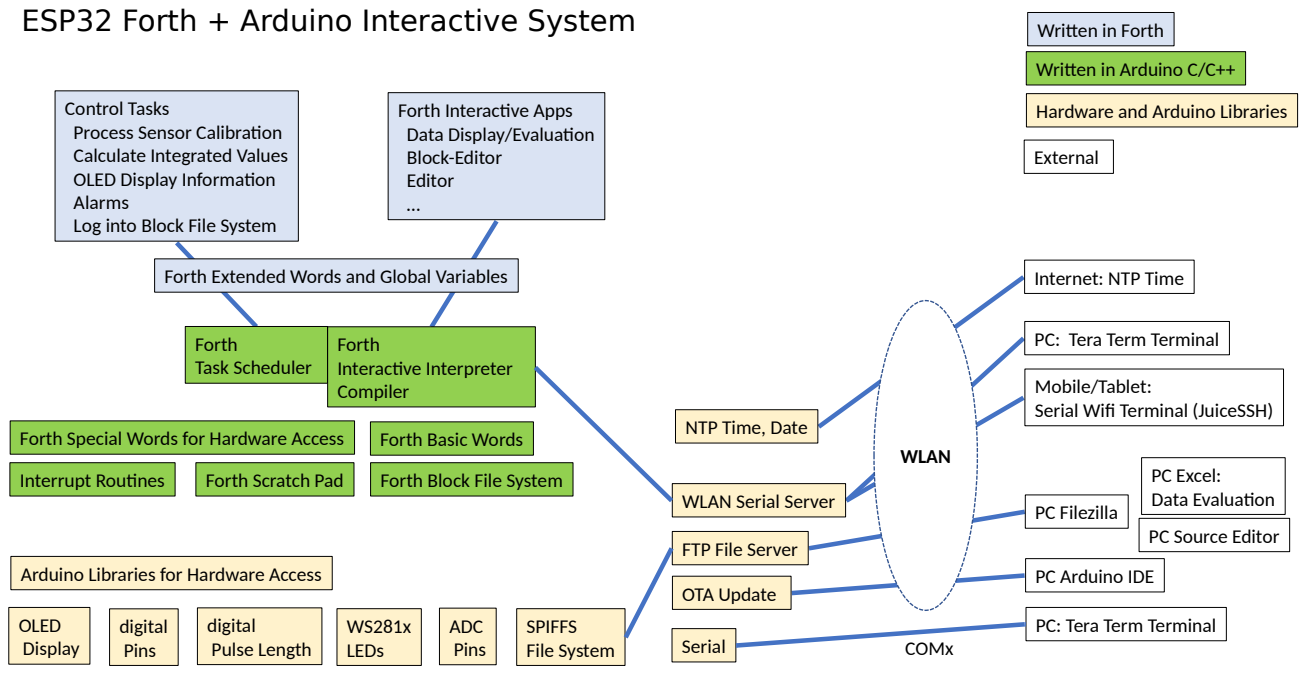


Abbildung 1: Struktur des Systems

Das Experimentierfeld

Das hier beschriebene Experiment beschäftigt sich mit Bewässerungstechnik für Pflanzen. Die optimierte Verwendung von Süßwasser ist — nach meinem Empfinden — eines der wichtigsten und spannendsten derzeitigen Forschungsfelder.

Vorversuche mit Erdfeuchtesensoren

In 2019/2020 hatte ich versucht, über preiswerte kapazitive Erdfeuchtesensoren eine automatische Bewässerung zu installieren. Es hat sich herausgestellt, dass sich so bei weitem keine ausreichende Reproduzierbarkeit darstellen lässt. „Erde“ ist etwas sehr Inhomogenes, sowohl über die verschiedenen Bodentypen hinweg, als auch schon innerhalb eines Blumentopfs. Ihre Porengrößenverteilung entscheidet über Kapillarwirkungen und Strömungswiderstände. Risse sind möglich und verändern alles drastisch. Gießwasser fließt leicht in Sekunden senkrecht von oben nach unten, verteilt sich jedoch seitlich sehr viel langsamer. Das *kapazitive Sensor–Wirkprinzip* bewertet die Feuchtigkeit in unmittelbarster Nähe (µm) sehr viel stärker als wenige Millimeter entferntere Tröpfchen. Dieser Sensor misst die feuchteste Stelle unmittelbar an seiner Oberfläche. In Verbindung mit der Inhomogenität der Erde entsteht kein brauchbares Ergebnis. Das beste Sensorprinzip für Erdfeuchte, das *Tensiometer*, besteht aus einem porösen, aber sonst dichten, wassergefüllten Gefäß in der Erde. Über die Kapillarkräfte saugt die umgebende Erde Wasser aus dem Gefäß und es bildet sich ein Gleichgewichtszustand mit Unterdruck im Gefäß, der gemessen werden kann. Dieser Sensor misst die trockenste Stelle an seiner Oberfläche und gleicht die Inhomogenität der Feuchtigkeit selbst aus. Das Ergebnis ist kein absoluter Feuchtewert, sondern die Saugspannung. Diese Sensoren sind leider teuer, nicht frostbeständig und zerbrechlich.

Das Experiment: Wasserbedarf messen

Statt die Erdfeuchte zu messen, kann versucht werden, den *Wasserbedarf* der Pflanzen zu ermitteln. Dies kann in drei Schritten geschehen: Nach der Klimaerfassung durch Sensoren wird zunächst als *Evapotranspiration* der Wasserbedarf von Grasbewuchs ermittelt in Liter/m²/Tag. Hierzu gibt es verschiedene Formeln. Allgemein anerkannt ist ein Ansatz von Penman–Morteith [1]. Hier soll jedoch die für Frankreich und Nordafrika entwickelte empirische Formel von Turc (1961) [2] untersucht werden, die sich für Luftfeuchte $>= 50\,\%$ dadurch auszeichnet, dass nur die Lufttemperatur in °C und die Strahlungsleistung der Sonne in W/m² gemessen werden muss. Es ist ein Größenbereich für die Evapotranspiration von ungefähr $0{,}5 \ldots 6\,\ell/\mathrm{m}^2$/Tag zu erwarten. Im dritten Schritt wird umgerechnet von Grasbewuchs auf einen anderen Pflanzentyp und deren Wachstumsstadium, was nach der *Geisenheimer Methode* [3] über Tabellen gemacht werden kann. Beispielweise benötigen Tomaten mit Pflanzenhöhe 1 m die 1,2–fache Wassermenge gegenüber Gras pro m². Dabei entsprechen $5\,\ell/\mathrm{m}^2$/Tag ungefähr $2\,\ell$/Tomatenpflanze/Tag. Es ergeben sich pro Saison immerhin 350 Liter pro Pflanze im professionellen Anbau. [4]

1. Sensorik: LDR für Sonnenstrahlung, NTC für lokale Lufttemperatur, Ultraschall für Füllhöhe/Wasserentnahmemenge aus einem Fass.
2. Umrechnung der Sensor–Rohdaten auf Strahlungsleistung in W/m², Temperatur 10tel °C, Wasserverbrauch nach Turc in 100stel ℓ/m²/Tag, jeweils über Kalibrierdaten.
3. Umrechnung des Wasserbedarfs für Tomatenpflanzen.
4. Aufzeichnung der Daten zeitgesteuert mehrfach täglich im CSV–Format für mögliche Auswertung und Darstellung mit Excel. Warnfunktion, wenn der Wasservorrat im Fass zur Neige geht.

Versuchspflanze im konkreten Fall sind Buschtomaten in Töpfen auf einem sonnigen Westbalkon. Diese stehen wegen der *Krautfäule* unter einem Vordach und würden ohne Wasserversorgung innerhalb von Tagen eingehen. Auf dem Balkon staut sich bei Sonne die Hitze, dieses örtliche Mikroklima besitzt sehr hohe Temperaturschwankungen. Die eigentliche Wasserversorgung der Pflanzen erfolgt über das mechanische System „Tropfblumat“, das interessanterweise nach dem Tensiometerprinzip arbeitet, und zusätzlichen manuellen Gießvorgängen und Eingriffen, die notiert werden müssen. Controller ist ein *ESP32 mit WLAN-Anbindung*, was eine gewisse räumliche Trennung erleichtert. Das Experiment soll mir aufzeigen, wie gut der Wasserbedarf der Tomaten mit den Sensoren und Schritten 1–3 bestimmt werden kann, indem der tägliche Wasserverbrauch mit dem errechneten verglichen wird. Abb. 2[1] zeigt eine Gegenüberstellung.

Update Ende August: Es zeigt sich insgesamt, dass für den hier vorliegenden Fall „Hitzestau auf Westbalkon“, das vereinfachte Turc–Modell ohne variable Luftfeuchte an sonnigen Nachmittagen den Wasserbedarf unterschätzt. Ein zusätzlicher *Luftfeuchtesensor* wäre vermutlich stark hilfreich.

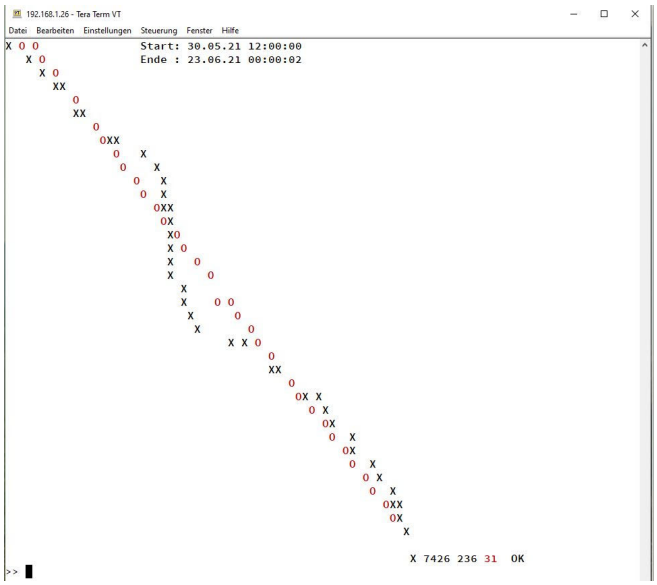


Abbildung 2: Manchmal ist schon eine wenig perfekte Grafik aufschlussreich, wie sie bei diesem Projekt von PC oder Mobiltelefon interaktiv abrufbar ist. Die schwarzen „X“ zeigen vertikal den berechneten Wasserverbrauch und horizontal den tatsächlichen Wasserbrauch jeweils aufsummiert bis zum Messzeitpunkt vom Anfang des gesamten Zeitraums. Der zeitliche Abstand zwischen zwei Punkten beträgt jeweils 12 Stunden. „7426“ bedeutet, dass insgesamt ein Wasserbedarf von 74,26 Liter/m² für Grasbewuchs berechnet wurde. „236“ bedeutet, dass der Wasserpegel im Behälter insgesamt um 236 mm abgesunken ist. Die roten „O“ zeigen eine Regressionsgerade, wobei „31“ deren Steigung kennzeichnet. Der Abschnitt, in dem die Punkte eher vertikal untereinander liegen, war eine Phase mit bedecktem Himmel. In diesem Zeitraum hat die Rechenmethode den Wasserverbrauch überschätzt. Anzumerken ist, dass zum Ende des Zeitraums die Pflanzen noch $< 8\,\mathrm{cm}$ klein sind und daher vor allem die Verdunstung der brachliegenden Erdoberfläche bewertet wird. Es ist noch kein echter Soll/Ist–Vergleich dargestellt, da sich noch zeigen muss, wie von Feldbepflanzung auf relativ kleine runde Einzeltöpfe mit Einzelpflanzen umzurechnen ist.

Warum Forth?

Wir sprechen hier über ein übertragbares 32–Bit–Forth, geschrieben in C. Mit *ATLAST* [5] und *YAFFA* [6] habe ich jeweils gute Erfahrungen gemacht. Das aktuelle Projekt verwendet ein stark weiterentwickeltes YAFFA–Forth. Diese Forthsysteme haben einen relativ kleinen Grundumfang und sind naturgemäß deutlich langsamer als optimierter Maschinencode.

Es ist das Wesen eines Experiments, dass man zur Versuchsplanung zwar eine Hypothese haben muss, jedoch nicht weiß, wie sich die Dinge entwickeln werden. In diesem Fall ist eine interaktive Eingriffsmöglichkeit, eine schnelle, leichte Veränderbarkeit von Parametern und

[1] In der PDF–Version ist die Grafik farbig.

von Software, sowie ein modularer Softwareaufbau, z. B. Trennung der Messdatenerfassung von der Auswertung und Darstellung, hochinteressant. Das Forthsystem mit einem primitiven kooperativen Multitasking ermöglicht das dynamische Laden z. B. von Auswerteroutinen und dynamisches Verändern des Verhaltens über z. B. globale Variablen. Da der Versuch in einem Hintergrundtask läuft, bleibt das System währenddessen offen zur interaktiven Bedienung über die „Konsole“ per WLAN.

Abb. 1 zeigt schematisch den Aufbau des Systems mit seiner Aufteilung in die Ebenen „Hardware und Libraries“, „Written in C/C++“, „Written in Forth“.

Obwohl ein Dateisystem vorhanden ist, hat sich die Verwendung einer Blockdatei als nützlich gezeigt. Protokolldaten werden zu festen Zeitintervallen zeilenweise von unten nach oben in Blöcke geschoben. Indem nur stündlich ein `flush` erfolgt, wird auf die begrenzte Speicherzyklenzahl des Flash–ROMs Rücksicht genommen.

Warum Arduino?

Charles Moore hat dafür plädiert, Software von Grund auf selbst zu entwickeln. Ganz im Gegensatz dazu wird hier als eine Hauptstärke der Arduino–Umgebung gesehen, dass sie die Möglichkeit bietet, Code zu verwenden, den andere dankeswerter Weise (!!!) zur Verfügung gestellt haben. Dabei ist eine Übertragbarkeit zwischen verschiedenen Controllern oft leicht möglich. Mein Zweck ist das Experiment, nicht die Softwareentwicklung, die hier einige Mannjahre benötigen würde.

Funktionen, welche hier Arduino für ESP32 mit seinen Libraries bereitstellt:

- Hardware–Abstraktion für leichte Portierbarkeit des Forth, das fast ohne Änderungen auf verschiedenen 32–Bit–Systemen laufen kann. Der Compiler kümmert sich darum, wie auf Portpins zugegriffen werden kann, oder ob eine FPU genützt werden kann.
- Library für ein kleines OLED–Display.
- Dateisystem im Flash des ESP32 (begrenzte Schreibzyklenzahl).
- Serielle Konsole über WLAN. Mit dem PC verwende ich TeraTerm. Vom Tablett und Smartphone aus ist ebenfalls ein einfacher Zugang möglich.
- Zeit und Datum über NTP aus dem Internet.
- WLAN FTP–Serverfunktion zur beidseitigen Dateiübertragung (PC–seitig: Filezilla).
- Software–Update via WLAN (OTA).
- Interrupts werden auf Arduino–Ebene bearbeitet.

Warum nicht Micropython?

Es stellt sich natürlich die Frage, inwieweit man statt Forth auch Micropython verwenden kann. Hier sind meine Erfahrungen gemischt. Während ich in 2021 zwei Projekte mit ESP32 bzw. mit RPi–Pico mit Hilfe von Micropython sehr zügig und gut abschließen konnte — sie laufen als feste Programme, wenig interaktiv — bin ich beim dritten trotz Inlineassembler wegen Geschwindigkeitsproblemen innerhalb einer Library gegen eine Wand gelaufen.

Zeitvergleich mit RPi–Pico bei einem simplen Fibonacci–Programm, (Zeitvergleiche beim 2. Routinenaufruf, da Micropython beim ersten zunächst compiliert):

C–Code auf Arduino pur ist der Bezugswert: Faktor 1

ATLAST–Forth ruft C–Routine auf: Faktor 3,5

Micropython ruft Inlineassembler: Faktor 12
(Ist so langsam durch die lange Aufrufzeit der relativ kurzen Routine, müsste eigentlich mindestens so gut sein wie C, da alle Variablen in Register passen.)

Micropython Viper Code (Integer): Faktor 21

ATLAST–Forth: Faktor 27

Micropython normal: Faktor 135

Wer dazu in der Lage und willig ist, könnte theoretisch neue C–Code–Funktionen in Micropython integrieren und das Gesamtsystem neu compilieren. Da Forth viel einfacher ist als Micropython, ist jedoch eine solche Integration ebenfalls viel einfacher.

Schlussbemerkung — Verbesserung der Lesbarkeit von Forth durch lokale Variablen

Es sei mir eine Anmerkung gestattet: Auch wenn manche Forth–Freunde das Problem als Anfängerproblem sehen: Forth *ist* schlecht lesbar, auch da Operanden und Ergebnisse auf dem Stack verborgen sind.

Sehr herzlich möchte ich hier für die Verwendung von *lokalen Variablen* werben. Selbst bei einer simplen Methode [7], bei der diese Variablen über feste Wortnamen `>a` und `a>`, `>b`, `b>` usw. angesprochen werden, wird die Lesbarkeit schon stark verbessert.

Referenzen

[1] Food and Agriculture Organization (FAO) of the United Nations `http://www.fao.org/home/en`

[2] Pierluigi Calanca, Pascalle Smith, Annelie Holzkämper und Christof Ammann; „Die Referenzverdunstung und ihre Anwendung in der Agrarmeteorologie“; Forschungsanstalt Agroscope Reckenholz–Tänikon ART, 8046 Zürich; `https://www.agrarforschungschweiz.ch/wp-content/uploads/2019/12/2011_04_1655.pdf`

[3] „Geisenheimer Bewässerungssteuerung 2021 mit kc-Werten für Penman-Verdunstung“; Hochschule Geisenheim – Institut für Gemüsebau. `https://www.hs-geisenheim.de/fileadmin/redaktion/FORSCHUNG/Institut_fuer_Gemuesebau/Ueberblick_Institut_fuer_Gemuesebau/Geisenheimer_Steuerung/kc-Werte_PENMAN_2021.pdf`

[4] Schmid, U.; „Tropfbewässerung von Tomaten in Bodenkultur bei einer Pflanzung im April“; Tomatenbroschüre BW 2007; https://wiki.forth-ev.de/lib/exe/fetch.php/vd-archiv:tomatenbroschuere.pdf

[5] sdwood68; „Yet another Forth for Arduino (YAFFA) ARM based Boards“; https://github.com/sdwood68/YAFFA-ARM

[6] Walker J.; „Autodesk Threaded Language Application System Toolkit (ATLAST)“, A Forth based toolkit; https://www.fourmilab.ch/atlast/

[7] Willsy; „Forth: Local Variables for the Common Man“; December 13, 2017 in TI–99/4A Development; https://atariage.com/forums/topic/273092-forth-local-variables-for-the-common-man/

Fortsetzung der Leserbriefe von Seite 7

. . . Von Typografen und Typisten

. . . Da sind Trenn– und Binde–Strich eins geworden. Ganz so, wie es auch unser geschätzter Autor Robert Clausecker vertritt. Zu meinem Layout seines Beitrages meinte er beim Korrekturlesen:

> „Mir ist aufgefallen, dass Du an diversen Stellen aus Bindestrichen Gedankenstriche gemacht hast. Ich habe eigentlich alle Gedankenstriche bereits TEX–Konform als -- ausgeführt, die verbleibenden Striche sollten Bindestriche bleiben. Und ich sehe, dass Du zumindest an einer Stelle aus -- ein --- gemacht hast . . .“

Also auch für ihn sind Trenn– und Bindestrich bereits zum gleichen Zeichen in der jeweiligen Schrift geworden. Und er meinte, dass der Geviertstrich im Deutschen eigentlich fast nicht verwendet würde, vielleicht noch für Preisangaben (2,— EUR).

Nun fragen wir uns natürlich, wie denn unsere Leser — und die anderen Autoren — wohl darüber denken. mk

https://www.typografie.info/3/wiki.html/g/geviert-r165/

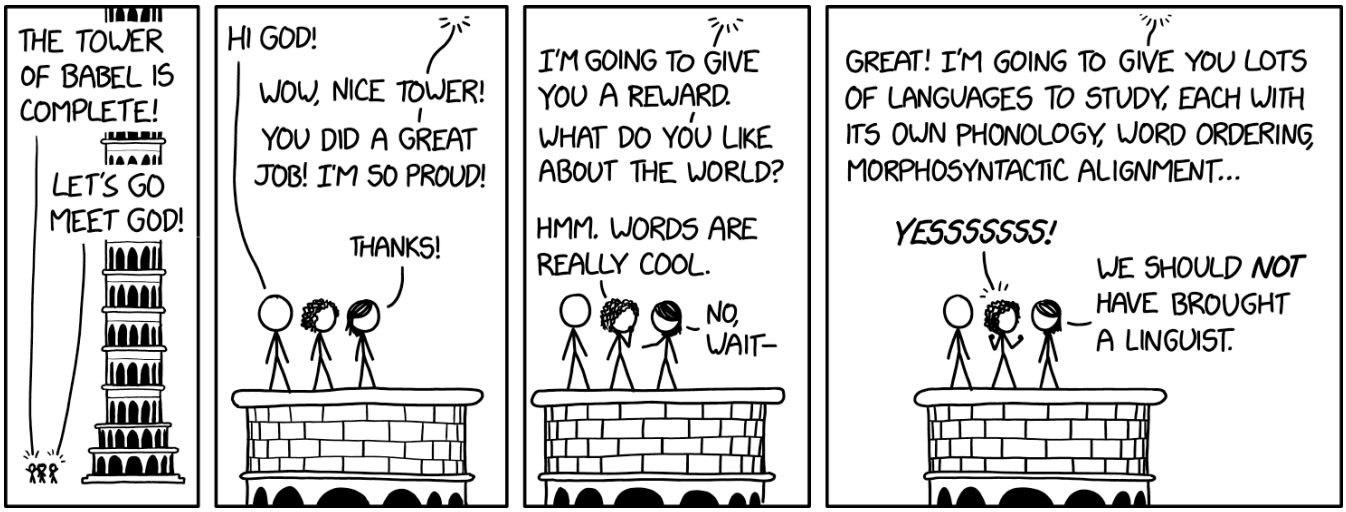
Hunderte von Apple II Source–Code–Floppies gerettet

In diesem Jahr war Kay Savetz auf der Jagd nach einer Kopie eines einzelnen Computermagazin–Artikels — und bekam einen riesigen Stapel von Apple II Quellcode–Disketten, alle in der Sprache Forth.

Wie ist das passiert? Was zum Teufel ist Forth überhaupt? Hat Kay diesen Artikel jemals gefunden? Schau selbst, was er auf dem *Kansas–Fest 2021* am 24. Juli 2021 erlebte: https://www.youtube.com/watch?v=vLv8XMxHr94

Die Quellen sind nun hinterlegt bei github.com/savetz

Fortsetzung der Leserbriefe auf Seite 17



https://xkcd.com/2421/

TheStack 2 — Die Außenstation

Erich Wälde

Im ersten Artikel [1] zu TheStack *[6] habe ich die Platinen und die Ansteuerung des Displays vorgestellt. In diesem Artikel beschreibe ich die erste Anwendung: eine Außenstation, die regelmäßig Daten vom Garten per Funk berichtet. Zwar steht die Station im Moment noch ohne Gehäuse neben meinem Schreibtisch, aber das wird sich finden.*

Wenn man so ein Platinenprojekt herstellt, und wenn man womöglich möchte, dass auch andere die Möglichkeiten sehen und die Platinen nutzen, dann kommt man nicht um ein paar detailliert vorgestellte Anwendungen herum. Die ursprünglich vorgesehene Veröffentlichung sollte ja in einem Heft mit Bezug zum Amateurfunk stattfinden. Daher hatte ich auch geschrieben, dass ich eine Stationsuhr gemacht hatte. Darunter ist eine Uhr zu verstehen, die die Zeit in *Universal Time Coordinated (UTC)* anzeigt und bei der Umstellung zwischen Normalzeit und Sommerzeit keinen Schluckauf bekommt. Aber diese Verwendung ist doch auf einen recht kleinen Kreis von Anwendern bezogen. Was also tun? Bernd Paysan schug einen Ultraschall–Abstandssensor vor, der die in Covid–Zeiten so wichtigen 1,5 m Abstand akustisch bewerten sollte. Aber solche Dinger gibt es längst und ich bewege mich derzeit auch wenig unter anderen Leuten. Also musste etwas anderes her.

Zu der Zeit war die Funkverbindung zwischen dem TechnoLine Außensensor (Temperatur, rel. Luftfeuchte) [3] mal wieder stundenlang gestört. Das ärgerte mich und so beschloss ich, dass es Zeit sei, eine Außenstation mit selbst verstandener Technik aufzubauen. Natürlich mit gewissen Extravaganzen, sonst kann's ja jeder.

Die Funktechnik ist die alte: RFM12–Module von HopeRF, die mit geringen Sendeleistungen im ISM–Band bei 434 MHz arbeiten. Die Module sind per SPI–Bus an den Controller angeschlossen [2].

Funk–Telegramm

Im alten Funk–Telegramm [4] wurden 16–Byte–große Pakete verwendet:

Bytes		Feld
1		Sender–Adresse
1		Empfänger–Adresse
1		Pakettyp (unbenutzt)
1		Datenlänge (unbenutzt)
10		Daten
	2	Anzahl der eingegangenen Messwerte
	2	Minimum
	4	Summe der Werte (für den Mittelwert)
	2	Maximum
2		Checksumme

Die 10 Datenbytes habe ich anfänglich auch alle ausgewertet. Aber es stellt sich heraus, dass mehr Statistik bei langsam veränderlichen Werten keine weiteren Einsichten bringen. Letztlich habe ich nur den Mittelwert weiterverwendet. Dann muss man die übrigen Werte auch nicht verschicken.

Wenn ich schon unbedingt alles neu machen musste, dann könnte ich ja auch das Format des Telegramms ändern. Ist ja eh' alles kompatibel mit nixx. Zwar könnte ich auch das Protokoll des MySensors.org–Projekts [11] verwenden, die schon an fast alles gedacht haben, aber das war mir auch sofort zu kompliziert. *Einfach* sollte es sein. Also:

Bytes	Feld
1	Stationsadresse Absender
1	Nummer des Sensors
2	Datenwert
2	Checksumme

Die Checksumme besteht weiterhin aus den Prüfbytes einer Fletcher16–Checksumme. Für jeden Sensor wird ein vollständiges Telegramm übertragen, mit einer kleinen Pause zwischen den einzelnen Aussendungen. Die Werte aller Sensoren werden alle 64 Sekunden verschickt. Jegliche Statistik über die Messwerte ist Aufgabe des Empfängers oder der weiteren Verarbeitungskette.

Streng genommen kommen für die Übertragung noch 2 Byte zur Synchronisation des Empfängers und 2 Byte zur Kennzeichnung des Telegramm–Beginns (magic bytes) dazu. Das sind dann 10 Byte pro Übertragung.

Die Übertragung findet in FSK (frequency shift keying) Modulation bei 1200 Baud statt. Das ist wahrscheinlich ein wenig lahm. Ob es gegen sehr kurze Störungen hilft, sei mal dahingestellt.

Jetzt kann man noch überlegen, ob 16 Bit für alle gewünschten Messwerte ausreicht. Für Temperaturen und Ähnliches sind 65536 verschiedene Werte üppig. Lediglich die Zeit in Sekunden ist bald am Ende angelangt. Da die Laufzeit des Controllers aber nur akademisches Interesse befriedigt, reicht da auch die Laufzeit in Tagen. Das reicht dann ein paar Jahre. Es reicht auch, diese Information nur einmal am Tag zu versenden.

Im alten Protokollformat könnte ich die Werte von 5 oder 6 Sensoren *gleichzeitig* verschicken. Wird die Übertragung gestört, dann sind auch alle 5 oder 6 Messwerte gleichzeitig unbrauchbar. Daher erschien es mir besser, jeden Messwert einzeln zu verschicken.

Im Programm verwende ich eine Datenstruktur `data.frame:`, welche die drei Einträge eines Daten–Telegramms enthält. Dazu gehören Worte, die die Adressenoffsets addieren (für lesbaren Quelltext), den Bereich löschen oder (hübsch) ausgeben.

```
\ block to hold one sensors data
\ offset
\  0    sid   senders station<<8+sensorID
\  2    dat   1 cell:
\  4    crc   fletcher16 crc of complete frame
\  6 -- beyond end

#6 constant df.size
\ C:( cxxx -- )  R:( -- addr )
: data.frame:  variable df.size allot ;
\ define header offsets
#0 constant df:sid    : df:sid+  ( df:src + ) ;
#2 constant df:dat    : df:dat+    df:dat +   ;
#4 constant df:crc    : df:crc+    df:crc +   ;

: df.erase ( addr -- )  df.size erase ;
: df.dump  ( addr -- )
  df.size 1 cells / 0 do
    dup i cells + @ #4 u0.r space
  loop drop
;
: df.show  ( addr -- )
  hex
  ." ...sensorid:" space dup df:sid+  @ #4 u0.r cr
  ." .......data:" space dup df:dat+  @ dup #4 u0.r
                   space decimal         u. hex cr
  ." ........crc:" space dup df:crc+  @ #4 u0.r cr
  drop
;
```

Stromversorgung

Schick wäre der energie–autarke Betrieb, so dass niemand befürchten muss, die Kabel (Stromversorgung, Datenbus) würden Blitze oder andere böse Geister in's Haus leiten.[1] Also habe ich den arbeitslos im Schrank vor sich hin staubenden Solar–Laderegler von danjulio [8] herausgeholt und Dokumentation gelesen. Dazu habe ich ein Solarpanel (12 V, 10 W) und eine passende Batterie (12 V, 10 Ah) erstanden und angekabelt. Zwar habe ich die Batterie erst mal herkömmlich aufgeladen, aber seither halten der Laderegler und das Panel die Batteriespannung hoch. Das Panel sieht derzeit nur den Osten aus meinem Fenster. Damit hatte ich endlich meine erste, ernstzunehmende *Photo–Voltaik–Anlage* in Betrieb.[2]

Der Laderegler ist via I2C ansprechbar und mit erstaunlich wenig Quelltext konnte ich u. a. die Batteriespannung lesen. Diese wurde quasi als erster Sensorwert regelmäßig auf die Reise geschickt.

Die Dokumentation des Ladereglers und seiner via I2C lesbaren Register ist vorbildlich. Eine Registeradresse wird zuerst übermittelt, anschließend werden zwei Byte zurückgelesen. Das hat alles sofort funktioniert, kein Rumprobieren und Kopfkratzen.

```
include lib-ew/twi.fs
include lib-ew/i2c.fs
$12 2* constant a_solar \ i2c address 8 bit format
: (sol_read16) ( reg_addr -- n )
  ( reg_addr ) 1 a_solar >i2c  #2 a_solar <i2c
  swap #8 lshift +
;
\ register #10: battery voltage in mV
: sol_VB     ( -- VB[mV] ) #10 (sol_read16) ;
: VB.get  sol_VB 10 / ; \ 1/100 V
```

Die Übermittlung eines Messwertes ist in wenigen Zeilen erledigt — die etwas aufwendigere Ansteuerung des Funkchips ist hinter der Funktion `df.transmit` verborgen. `$7a` ist die Nummer der Station. Die Batteriespannung bekommt die Sensornummer `$01` zugewiesen.

```
data.frame: D_data
: once
  \ Supply Voltage 1/100 V (Battery)
  VB.get
  $7a01 D_data df.fill
  D_data dd \ debug
  D_data df.transmit

  \ ...
;
```

Sensoren

Ohne ordentliche Sensoren ist so eine Außenstation natürlich relativ unnütz. Ursprünglich wollte ich einen BME280–Sensor (Temperatur, relative Feuchte, Luftdruck) anschließen. Der hat aber eine sagen wir mal herausfordernde Schnittstelle. Zwar hat Martin Bitter mich auf eine Realisierung durch unsere niederländischen Nachbarn hingewiesen [5], aber ich muss zugeben, dass ich weder die Schnittstelle ausreichend verstanden habe, noch den von Bosch Sensortec bereitgestellten Quelltext in C, noch die noForth Implementierung. Nach vielleicht 20 Stunden Rumgemurkse habe ich das Zeug grantig in die Ecke gelegt.

Ich habe aus purem Trotz eine Handvoll der derzeit sündhaft teuren Sensirion SHT85–Sensoren einfliegen lassen (angeblich aus den USA). Deren Schnittstelle ist wesentlich einfacher gestaltet — nach circa vier Stunden redete so ein Ding mit mir. Inklusive herausfinden, dass es sich lohnt, die Busfrequenz auf 50 kHz zu reduzieren (statt 400 kHz). Die Rohwerte sind letztlich als Wert zwischen 0 und 1 zu verstehen, sie sind linearisiert und temperaturkompensiert. Und die Umrechnung ist mit zwei Koeffizienten (Geradengleichung) vergleichsweise ein Kinderspiel. So macht das Spaß!

```
\ +twi sets 400 kHz; try 50 kHz instead
\ 11059200 / 50000 = 222
\ 16 + 2 * 26 * 4  = 224 -> 49 kHz (f_cpu: 11059200)
: +twi.50kHz ( -- )
  $00 TWCR c!
```

[1] Just letzte Woche hat ein Blitzeinschlag in der Nähe die Fritz!Box gehimmelt — das arme Ding! Sehr vorausschauenderweise hatte ich aber eine baugleiche, konfigurierte Fritz!Box im Regal. Und noch viel vorausschauender hatte ein Spezl aus der Linux User Group eine arbeitslose solche Box herumliegen, die er mir als neuen Ersatz vermacht hat. Glück gehabt. Danke!

[2] Nicht erstzunehmende Exemplare in Form von Solarlämpchen gab es wohl. Eines Tages werde ich auch mal nachmessen, was die Billigdinger so an Störstrahlung aussenden.

```
  #26 TWBR c! \ bit rate reg
  $01 TWSR c! \ 01 -> prescaler == 4
;
$44 2* constant a_sht85
include lib-ew/i2c_sht85.fs

variable sht85_T
variable sht85_H
: sht85.get.H,T
  sht85.data        \ -- Th Tl crc Hh Hl crc
  drop
  swap #8 lshift + sht85.H  sht85_H !
  drop
  swap #8 lshift + sht85.T  sht85_T !
;
```

Die Umrechnung für die Temperatur sieht dann beispielsweise so aus:

```
\ T [°C] = -45 + 175 * S_{T}/65535
: sht85.T ( S_T -- T*100[°C] )
  0 ( s>d unsigned! )
  #17500 ud*
  #65535 um/mod
  swap ( remainder ) drop
  #4500 -
;
```

Aber auf die SHT85–Sensoren musste ich ja warten. Da gab es noch diese arbeitslosen Geiger–Müller–Zählrohre. Ich hatte zwei Bausätze, zum einen den Mighty Ohm [9], zum anderen einen Bausatz von AATiS, den AS622 [10]. Der größte Unterschied besteht in der Spannungsversorgung. Es sind mir 5 V einfach sympatischer als 3,3 V — die 3,3 V sind mir irgendwie zu schlaff. Wahrscheinlich völlig unbegründet, aber ach!

Beide Bausätze haben einen Ausgang, an dem man die Flanken zählen kann. Jedes registrierte Ereignis erzeugt einen kurzen Impuls. Diesen Ausgang habe ich an den 16–Bit Timer/Counter1 angeschlossen. Man kann den ATmega644PA–Controller sogar schlafen legen, der Counter zählt weiter. Diesen fortlaufenden Wert gebe ich alle 64 Sekunden (das hat mit dem Schlafintervall von 8 Sekunden zu tun) als Funk–Telegramm aus. In einem zweiten Telegramm wird zusätzlich die Differenz seit dem letzen Mal verschickt. Das ist reine Bequemlichkeit, dann kann man mitlesen und muss auf der Empfangsseite nichts umrechnen.

Um den Zähler in Betrieb zu nehmen, muss der zugehörige Pin korrekt konfiguriert werden. Der Zähler soll auf steigende, externe Flanken reagieren. Das Zählerregister wird auf Null zurückgesetzt.

```
PORTB 1 portpin: pin.t1
: tcnt1.pin.init ( -- )
  pin.t1 over over pin_input low \ highZ
;

: +tcnt1
  [ %00000111       \ CS1[210] ext. clock on T1
  ] literal TCCR1B c!             \ rising edge
  0 TCNT1 !         \ 16 bit word
;
: -tcnt1
  0 TCCR1B c!
;
```

Hauptschleife

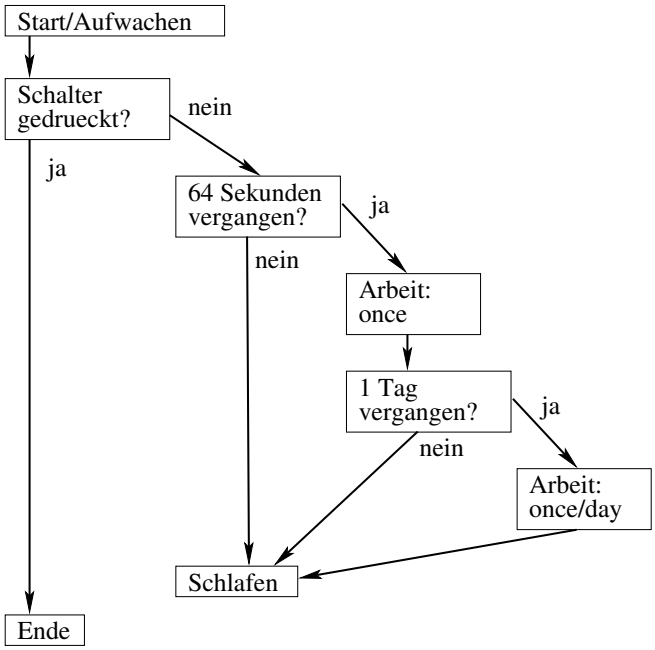


Abbildung 1: Ablaufdiagramm der Hauptschleife

Der Außensensor braucht keinen Multitasker, denn er wird nicht von außen abgefragt. Das Programm läuft in einer Schleife und versendet periodisch frisch erhobene Datensätze. Dabei schläft es die meiste Zeit, bis der abgelaufene Timer/Counter2 den Controller weckt. Timer/Counter2 wird durch einen Uhrenquarz (32768 Hz) getrieben. Der Takt des Quarzes wird durch 1024 geteilt, der Timer/Counter2 zählt auf 256 und läuft damit nach 8 Sekunden über. Der Überlauf löst einen Interrupt aus, welcher den Controller aus seinem Schlafzustand aufweckt.

```
\ use timer2 to wake up from sleep
: ticks2_1/8_isr ;
: +ticks2_1/8
  [ %00000000                          \ normal mode
  ] literal TCCR2A c!
  [ %00000111             \ 111 = 7 = clock_ts2/1024
  ] literal TCCR2B c!
  ASSR_AS2 ASSR  c!        \ source: 32 kiHz crystal
  ['] ticks2_1/8_isr TIMER2_OVFAddr int!
  TIMSK2 c@ $01 or TIMSK2 c! \  enable OVF2 interupt
;
```

Damit hat der Controller einen „8–Sekunden–Tick“. Es bietet sich an, periodische Tätigkeiten in Vielfachen von acht Sekunden zu verwalten. Die Struktur der Hauptschleife ist nicht kompliziert. Wenn eine Minute (also 64 Sekunden!) vergangen ist, verschicke alle Daten. Wenn außerdem ein Tag vergangen ist, verschicke noch mehr Daten. Ansonsten: weiterschlafen. Der Schalter, um die

Schleife zu verlassen, ist lediglich für die externe Bedienung bzw. Abfrage an der seriellen Schnittstelle gedacht. Den kann man auch einsparen.

Im Quelltext sind noch etliche Details für die Buchhaltung notwendig. Damit sieht die Hauptschleife in etwa so aus:

```
\ every N ticks transmit data (report)
#8 Evalue EEticks/report        \  8 * 8 =  64 sec
variable ticks/report.follow    \ 1 tick == 8 sec
: --ticks/report.follow  -1 ticks/report.follow +! ;

2variable uptime                \ in seconds
: ++uptime ( -- )    #8. uptime d+! ; \ 8 sec ticks!
variable uptime/d               \ in days

\ once per day transmit other data
#1350 Evalue EEticks/day        \ 86400 / 64 = 1350
variable ticks/day.follow
: --ticks/day.follow       -1 ticks/day.follow +! ;

: run
  begin
    \ leave loop when sw.user is pressed (pin low)
    sw.user pin_high? while

      --ticks/report.follow \ 8 sec ticks
      ticks/report.follow @ 1 < if  \ 0 <= really
        EEticks/report ticks/report.follow !
        once \ work!

        --ticks/day.follow \ 64 sec ticks!
        ticks/day.follow @ 1 < if  \ 0 <= really
          EEticks/day ticks/day.follow !
          once.per.day \ work!
          1 uptime/d +!
        then
      then

      $00 sleep \ idle mode, keep tcnt1 running
      ++uptime
  repeat
  leds.power.on  +leds  lederr on
;
```

Derzeit werden alle 64 Sekunden, d. h. 1350/Tag, folgende Messwerte als ganze Zahl verschickt:

- die Batteriespannung in 1/100 V
- der akkumulierte Zählerstand des Zählrohrs
- die Differenz des Zählerstands seit dem letzten Report, also Impulse/64 Sekunden.
- die Temperatur in 1/100 °C
- die relative Feuchte in 1/10 %
- der Luftdruck in 1/10 hPa

Einmal am Tag wird zusätzlich

- die Laufzeit des Controllers in Tagen

verschickt. Braucht kein Mensch, ist ganz klar aus der Kategorie: *because we can!*

Empfänger und Datensammlung

Da ich elektromagnetische Wellen bei 434 MHz nicht sehen kann, geschweige denn *flüssig mitlesen*, musste natürlich unverzüglich auch ein Empfänger her. Dieser sammelt die eingegangenen Telegramme ein, überprüft die Checksumme, bildet ggf. Mittelwerte und liefert die Daten auf Anfrage auf der seriellen Schnittstelle ab. Am anderen Ende der seriellen Schnittstelle läuft ein Perl–Programm auf einem Raspberry Pi, welches die Buchstaben auswertet und die Messwerte über Ethernet weitermeldet an den Rechner im Keller: perl → MQTT broker mosquitto → MQTT client telegraf, der die Werte in eine Datenbank schreibt → influxDB Datenbank, weil das ist heutzutage hip → Grafana Webserver, der mit viel Javascript verziert die Daten an einen Webbrowser ausliefert. Der Weg ist vielleicht lang, aber sicher modern. Die Stapelhöhe der Abstraktionen ist beträchtlich. Aber bequem ist es dann doch:

- es gibt einige von mir vorkonfigurierte Kurven, die Grafana serviert
- auf jedem Rechner im Haus reicht ein Webbrowser, um die Daten anzusehen
- man kann in die Daten zoomen
- man kann die angezeigten Daten/Kurven einzeln ein-/ausknippsen
- man kann alte Daten angucken, da ich alle verfügbaren Daten in die Datenbank migriert habe. Bis März 2010 kann man da gucken
- man kann Anmerkungen an die Kurven schreiben, also etwa, dass das Loch in den Daten ein Stromausfall war

Klar bin ich auf die Zulieferung von Software von anderen angewiesen, und klar musste ich auch erst wieder lernen, wie man das alles zusammenpuzzelt, und nein, da ist nichts abgesichert — per Funk kann mir jeder Nachbar Fake–Daten *injizieren*. Aber das macht nichts, weil es werden aus diesen Daten keine automatischen Aktionen abgeleitet. Einzig das Einsammel–Skript schreibt eine E–Mail, falls eine Station länger als eine Stunde keine Daten mehr abgeliefert hat. Aber das alte Skript, welches die Daten angezeigt hat, ist auf eine nicht mehr gepflegte Bibliothek angewiesen. Das wäre in jedem Fall Arbeit geworden.

Experimente mit dem Zählrohr

Wenn schon die komplette Kette vom Sensor bis zur bunten Kurve exisitert, dann kann man ja auch richtige Experimente machen, oder? Und was wäre denn hübscher, als irgendetwas, was zeigt, dass das Zählrohr auch mehr messen kann, als nur ca. 20 Impulse pro Minute? Eine altmodische Uhr mit Radium–Leuchtfarbe? Keine da. Ein paar alte Glühstrümpfe für Gaslaternen, die ein bisschen Thorium enthalten? Auch keine da. Ein Stück Pechblende? Könnte man im Mineralienhandel bestimmt bekommen …

Aber es gibt andere Möglichkeiten. Stellt sich nämlich heraus, dass alte Keramik, bevorzugt in schrillen Farben,

durchaus Uransalze enthalten kann. Auch Glaswaren aus einer bestimmten Zeit können Uran enthalten. Natürlich nur in winzigen Mengen … Beim Aufräumen der Garage fanden sich zwei Fliesen, die die Vorbesitzer in der Küche hatten. Rot und braun, made in Italy von ca. 1975. Ob die ticken?

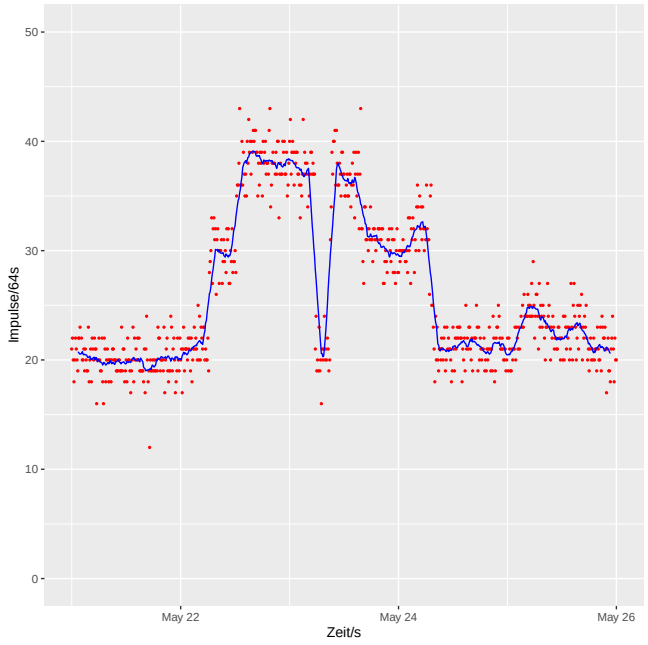


Abbildung 2: Die Zählwerte in Impulse/64 Sekunden (rote Punkte) und ein gleitender Mittelwert mit einem Radius von 8. Man kann die unterschiedlichen Messzeiten klar unterscheiden, von links: natürlicher Untergrund; rote Kachel; weiße Kachel; Untergrund; neue weiße Kachel; Bodenfliese; natürlicher Untergrund. Die Abbildung enthält die Daten von fünf Tagen. Die Zählrate hängt von der Temperatur ab. Das sieht man sowohl am Ende der Messdauer der Bodenfliese, als auch im Verlauf des nächsten Tages.

Die rote Fliese hab ich dann mit kaum Abstand auf das Zählrohr gelegt. Kurze Antwort: Die Rate steigt um ca. 8 Impulse pro Minute, ca. 40 % des Hintergrunds. Aha! Erstaunen. Und was ist mit den weißen Fliesen von Villeroy und Boch, ebenfalls circa 1975? Die Zählrate steigt um etwa 16 Impulse pro Minute. Aha?! Die ist doch gar nicht schrill orange? Hmmm. Die braunen Fußboden–Fliesen (Spaltklinker) bringen ebenfalls ca. 8 Impulse pro Minute zusätzlich. Und was ist mit den weißen Fliesen von 2007, die jetzt im Bad hängen? Die sind doch quasi neu und bestimmt ganz ohne Uran? Stellt sich raus, die bringen ebenfalls ca. 16 Impulse pro Minute. Sowas! Alle Mann Panik? Vermutlich heißt die Antwort, dass diese Fliesen alle ohne Uransalze gefertigt sind. Und ziemlich wahrscheinlich heißt die Antwort: Baustoffe enthalten in unterschiedlichen Mengen auch Kalium. Kalium besteht zu einem geringen Anteil (117 ppm) auch aus dem radioaktiven Isotop Kalium–40, welches über einen β–Zerfall in Kalzium oder Argon zerfällt. Halbwertszeit: 1,25 Milliarden Jahre. Das dauert also ein Weilchen. Was man alles lernt!

Wie weiter?

Eine Anzeige für die Küche wäre nicht schlecht, oder? Da bietet es sich an, eine weitere Empfängerstation aufzubauen, die die empfangenen Daten auf einem Display anzeigt. Diese 4x20–Zeichen–Displays wurden im letzten Artikel schon vorgestellt. Dazu hätte ich dann gerne eine Uhr (was sofort dieses bodenlose Loch namens Zeitmessung öffnet — und ich dachte, ich hätte eine Weile meine Ruhe). Ein weiterer SHT85 misst Temperatur und Luftfeuchte in der Küche. So der Plan. Eine solche Station könnte man auch guten Gewissens Nachbars geben. Und vielleicht haben die Nachbarskinder ja Interesse am Bastelkram — das wär ja auch mal was.

In einem Bastelkramheft habe ich gelesen, dass die rfm12–Transceiver sowas von out sind — da gibt's zum einen die rfm69–Reihe. Die können jetzt etliche Bytes als Paket vorbereiten, AES–Verschlüsselung betreiben, vor dem Senden horchen, ob die Luft rein ist (listen–before–talk) und wahrscheinlich noch mehr. Zum anderen gibt es die rfm9x–Reihe, die LoRa sprechen. Das weckt natürlich Begehrlichkeiten.

Außerdem würde ich gerne noch andere Programmiersprachen auf dem Controller ausprobieren.

- C mit der Arduino–IDE — Dazu müsste ich wissen, wie ich diese Platine in die IDE integrieren kann. Weiß das zufällig jemand?
- uLisp — das wäre dann auch erreichbar, weil diese Lisp–Implementierung als Arduino–C–Projekt vorliegt
- BasCom — irgendwo liegt noch eine CD rum. Möglicherweise ist das mit Linux/wine32 zum Leben zu erwecken

So Sachen

Eine Keramik *mit* Uransalz in der Glasur habe ich noch nicht gefunden. Deswegen würde ich gerne im Sommer mal auf einen Flohmarkt pilgern und den Mighty Ohm ein wenig an die frische Luft führen. Oder hat jemand was übrig? Oder ein paar altmodische Glühstrümpfe für Gaslaternen (nicht mehr im Handel, enthalten ein wenig Thorium)? Oder einen altmodischen Wecker mit Leuchtziffern? Gewöhnlich leuchten die gar nicht mehr. Nein, gewöhnlich ist die Radioaktivität der Radium–Leuchtfarbe nicht erloschen, sondern die Farbe ist kaputt gestrahlt worden.

Sensoren für Windrichtung und –geschwindigkeit wären noch nett. Allerdings ist der Standort für diese Dinger aufwendig, die mechanische Ausführung auch nicht ganz trivial. Und womöglich muss man stellenweise eine Heizung für den Winter vorsehen. Alles nicht so einfach.

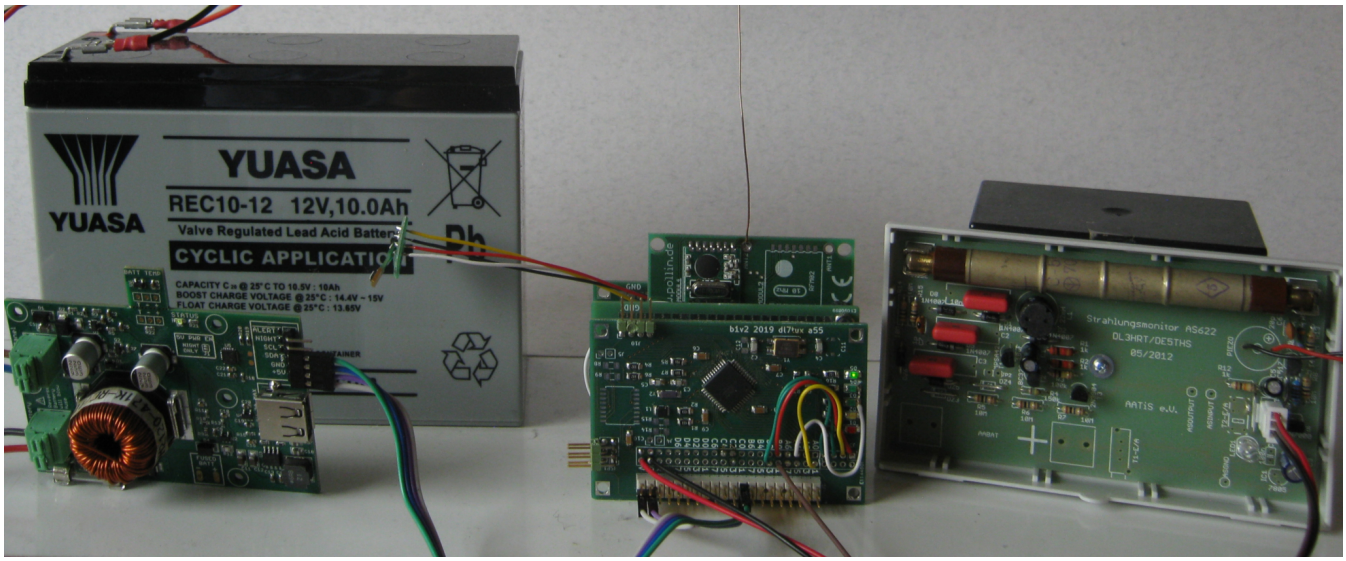


Abbildung 3: Die Teile der Außenstation — derzeit noch im Trockenen. Von links: Der Laderegler und die Batterie; der Mikrocontroller mit Funkchip und SHT85 (steht nach links raus); der AS622–Geigerzähler mit einem kleinen Zählrohr.

Referenzen

1. TheStack— Stapelbare Platinen — Erich Wälde, VD 2021–02, S. 7ff
2. Funklöcher — Erich Wälde, Martin Bitter, VD 2011–03, S. 7ff
3. Technoline Funksensoren belauschen — Erich Wälde, Martin Bitter, VD 2011–04, S. 14ff
4. Eigene Funk–Sensoren empfangen — Erich Wälde, VD 2013–01, S.9ff
5. Read the BME280 Pressure, Temperature and Humidity Registers — F. L. van der Markt VD 2020–01 S. 21ff
6. TheStack (Schaltpläne, Layout) — `https://git.sr.ht/~ew/TheStack`
7. TheStack (Quelltext) — `https://git.sr.ht/~ew/TheStack-Code`
8. Crowdsupply maker Solar Power — `https://www.crowdsupply.com/danjuliodesigns/makerpower-solar`
9. Mighty Ohm — `https://mightyohm.com/blog/products/geiger-counter/`
10. AATiS 622 — `https://www.aatis.de/content/bausatz/AS622_Geigerz%C3%A4hler`
11. MySensors.org — `https://www.mysensors.org/`

Fortsetzung der Leserbriefe von Seite 11

Strahlung

> „Jede einmal begonnene Meßreihe wird durchkämpft, auch wenn sie noch so hoffnungslos aussieht. Man kann ja auch aus einem mißglückten Versuch viel neues lernen.“ (Geiger, Hans, 1933; in [1])

Angeregt durch die Lektüre von Erich's Beitrag wärend des Layouts dieses Heftes stieß ich auf einen Aufsatz von Sebastian Korff aus der Universität Flensburg vom Institut für Physik und Chemie und ihre Didaktik — frei erhältlich als PDF [1]. Korff zitiert nicht nur in seiner wissenschaftshistorischen Analyse, da wird nachgebaut und experimentiert mit damaliger Technik.

> „Trotz der ikonischen Funktion dieses Instruments im 20. Jahrhundert wurde das Geiger–Müller–Zählrohr von der Wissenschaftsgeschichte bisher nur vereinzelt zur Kenntnis genommen. Um insbesondere die apparativen und instrumentellen Aspekte des Zählrohrs näher zu untersuchen, bietet sich neben der detaillierten historischen Quellenanalyse die Anwendung der *Replikationsmethode* an. Deshalb werden in diesem Beitrag die Erfahrungen, die beim Nachbau von frühen Zählrohren von 1928 gemacht worden sind, im Detail dargestellt . . . “

Spannend. mk

[1] Korff, S. Das Geiger-Müller-Zählrohr. N.T.M. 20, 271–308 (2012).

Permutationen

Jens Storjohann

Permutation hat zwei nahe verwandte Bedeutungen, die gerne, auch in diesem Text, wechselnd gebraucht werden, im Vertrauen darauf, dass der Leser dem Zusammenhang entnimmt, welcher Aspekt gerade gemeint ist: entweder eine Anordnung von Dingen, oder die Abbildung, die eine Anordnung von Dingen in eine andere Anordnung überführt. Für den Informatik–Praktiker ist häufig der Aspekt des Sortier–Algorithmus im Vordergrund. Aber auch ein Sport–Turnier oder wechselnde Zuordnungen von Mitarbeitern zu Arbeitsplätzen lassen sich mit Hilfe von Permutationen beschreiben.

Was sind und was sollen die Permutationen?

In dem Titel steckt eine Anspielung auf ein sehr bemerkenswertes Mathematik–Buch von Richard Dedekind, der einer der Pioniere der modernen Algebra war. [1]

Zur Benennung der permutierten Dinge werden gerne die ersten natürlichen Zahlen oder die ersten Buchstaben des Alphabets herangezogen, weil sie *unterscheidbar* sind und einer natürlichen oder alphabetischen *bekannten Reihenfolge* unterliegen.

Die Wikipedia gibt ausführliche deutsche und englische Informationen. [2] [3]

Spezielle Permutationen

Am wichtigsten ist die *identische Permutation*, die alle Elemente unbeeinflusst lässt, sie also identisch abbildet. Ein Extremfall ist die totale Umordnung, die aus $(1\ 2\ \ldots\ n)$ die umgekehrte Reihenfolge $(n\ \ldots\ 2\ 1)$ macht.

Ansonsten gibt es viele, oft aus Anwendungen herrührende Notwendigkeiten, Klassen zu definieren, wie zum Beispiel die *Fixpunkt–Freie*, und sie zu untersuchen. Die würde bei einer Anwendung auf Arbeitsplatzwechsel bedeuten, dass niemand auf seinem gewohnten Platz bleibt.

Beschreibung von Permutationen

Zwei–Zeilen–Darstellung

Verwendet man eine „Vorher–Nachher“–Schreibweise, die *Zwei–Zeilen–Form*, kann man für ein in diesem Text mehrfach verwendetes Beispiel schreiben

$$P_1 = \begin{pmatrix} 1\ 2\ 3\ 4\ 5\ 6 \\ 2\ 1\ 5\ 6\ 3\ 4 \end{pmatrix}$$

Diese von dem französischen Mathematiker Augustin Louis Cauchy[1] stammende Beschreibung ist leicht verständlich und besonders für Untersuchungen mit Hilfe von „Papier und Bleistift“ geeignet.

Fehlstände

Fehlstand [4] bezeichnet eine Größe, die sehr nützlich für die Beschreibung von Permutationen ist.

[1] * 21. August 1789 in Paris; † 23. Mai 1857 in Sceaux

[2] Englisch: inversion table

Wir erklären den Begriff *Fehlstand* durch ein Beispiel aus dem Supermarkt–Alltag. Nehmen wir an, dass eine geordnete Schlange von Kundinnen
$P_0 = (1\ 2\ 3\ 4\ 5\ 6)$ sich vor einer Kasse gebildet hat. Plötzlich kommt eine Durchsage, dass man bitte aus technischen Gründen zu einer neu geöffneten Kasse wechseln möge. Der nun einsetzende Positionskampf führt durch Permutation zu einer neuen Anordnung $P_1 =$**(2 1 5 6 3 4)**.

Wir betrachten nun die Frage, wer Grund hat, sich wegen durch die Permutation verursachter erhöhter Wartezeit zu ärgern.

- 1 ärgert sich, weil nun die 2 vor ihr steht, was eine Verschlechterung darstellt.
- 2 ärgert sich nicht, weil niemand vor ihr steht.
- 3 ärgert sich, weil nun 5 und 6 vor ihr stehen. Die schon zu Beginn vorne stehenden 1 und 2 verursachen keinen Ärger.
- 4 ärgert sich, weil 5 und 6 vor ihr stehen.
- 5 ärgert sich nicht, weil sie keine erhöhte Wartezeit hat.
- 6 ärgert sich nicht. Die letzte Person in einer Warteschlange kann nie in eine Situation kommen, in der sie sich ärgert, weil sie sich gar nicht weiter verschlechtern kann.

Wir bilden also aus einer Permutation $(a_1, a_2, a_3, a_4, a_5, a_6)$ einen *Fehlstandvektor* $(b_1, b_2, b_3, b_4, b_5, b_6)$, der auch Inversionvektor[2] genannt wird. Der Fehlstandvektor enthält genau die Zahlen, die wir eben als *zusätzliche* Wartezeiten kennen gelernt haben:

$$B_1 = (b_1, b_2, b_3, b_4, b_5, b_6) = (1, 0, 2, 2, 0, 0)$$

Eine abstraktere Definition der Elemente des von P_1 gebildeten Fehlstandvektors erhält man so:

$$b_i = \#\left\{j |\ i < j\ \wedge\ p_1(j) < p_1(i)\right\}$$

Das Symbol # bedeutet dabei die Anzahlbildung. Es wird also die Anzahl der Elemente einer Menge von Fehlständen gebildet. Es lassen sich verwandte Kenngrößen definieren, die in [3] aufgeführt sind.

Welchen Nutzen haben die Fehlstände?

Man kann anhand des eben beschriebenen Beispiels nachprüfen, dass die Fehlstände b_j einer Permutation $(a_1, a_2, \ldots, a_n)$ folgendermaßen eingeschränkt sind:

$$0 \le b_1 \le (n-1), \quad 0 \le b_2 \le (n-2), \ \ldots \\ , \ 0 \le b_{n-1} \le 1, \quad 0 = b_n \quad (1)$$

Dies ist in Abb. 1 für unser Beispiel dargestellt.

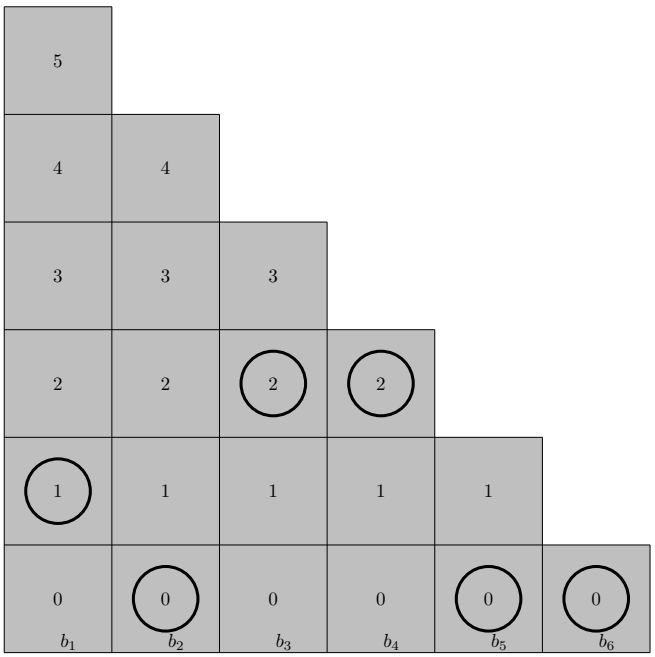


Abbildung 1: Fehlstände für 6 Elemente, speziell für (1, 0, 2, 2, 0, 0)

Wenn man sich in den Spalten jeweils einen Eintrag auswählt, hat man

$$6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 6! = 720$$

Möglichkeiten.

Weil die Fehlstände innerhalb ihrer Grenzen unabhängig voneinander vorgegeben werden können, ist der Umgang mit ihnen einfacher als mit den Permutationen in zum Beispiel der oben genutzten Zwei–Zeilen–Schreibweise, wo man immer darauf achten muss, keine „Doppelverwendung“ zu erhalten.

Also lassen sich Zufalls–Permutationen, zum Beispiel zum Testen von Algorithmen, gewinnen, indem man in jeder Spalte der Abbildung Abb. 1 genau einen zufallsgesteuerten Eintrag wählt.

Das Fakultätensystem, eine besondere Stellenschreibweise

Wir kennen durch langjährige Tradition[3] die Bedeutung von Zahlenangaben in Stellenschreibweise, wie z. B.

$$n = 243$$

nämlich

$$n = 2 \cdot 100 + 4 \cdot 10 + 3 \cdot 1.$$

[3] Seit Adam Ries [5]

Eine natürliche Zahl wird dargestellt als eine Summe von Produkten. Diese sind gegeben durch Koeffizienten gewichtet mit jeweils einem Basiselement. Im hier gegebenen Fall sind die Basiselemente in einfacher Systematik gegeben durch Zehnerpotenzen.

Man verlangt, um eine *Eindeutigkeit der Schreibweise* zu erhalten, dass die Koeffizienten nur die Werte $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ annehmen und führende Nullen nicht betrachtet werden.

Für unser *fakultät–basiertes Stellensystem* legt die Abb. 1 folgende Regeln nahe, um eine eindeutige Schreibweise zu erhalten:

- Die erste Spalte, die die Koeffizienten 0 bis $n-1$ enthält, wird mit $(n-1)!$ gewichtet.
- Die zweite Spalte enthält die Koeffizienten 0 bis n-2 und wird mit $(n-2)!$ gewichtet.
- Dies wird fortgesetzt bis zur (formalen) Gewichtung $0! = 1$ für den immer verschwindenden Koeffizienten 0.

Wählen wir in jeder Spalte den maximalen Koeffizienten, erhalten wir als Formel für die Summe, wie z. B. mit vollständiger Induktion zu beweisen ist

$$(n-1)(n-1)! + (n-2)(n-2)!+, \ldots, (1)1! = n! - 1.$$

Wir haben also minimal 0 und maximal $n! - 1$, also insgesamt $n!$ verschiedene Werte erhalten. (Dass jeder Zwischenwert angenommen wird, lässt sich beweisen.) Dies widerspiegelt unsere Feststellung, dass n Elemente $n!$ Permutationen zulassen.

Zerlegung einer Zahl im Fakultäten–System

Wir wählen als Beispiel die Zahl 136. Wir erhalten die Ergebnisse in Tabellenform. Die Koeffizienten können unmittelbar als Fehlstände interpretiert werden.

136/120=	1	Rest	16
16/24=	0	Rest	16
16/6=	2	Rest	4
4/2=	2	Rest	0
0/1=	0	Rest	0
per def.	0		

Tabelle 1: Zerlegung der Zahl $136 = 1 \times 5! + 0 \times 4! + 2 \times 3! + 2 \times 2! + 0 \times 1! + 0 \times 0!$

Unserem oben mehrfach gebrauchten Beispiel P_1 für eine Permutation ist also die Zahl 136 zugeordnet. Denn aus ihr lassen sich die Fehlstände, wie eben gezeigt, wiedergewinnen.

Rekonstruktion einer Permutation aus dem Fehlstandvektor

Auch hier lässt sich die Methode am einfachsten an einem Beispiel erläutern. Wir betrachten wieder unser schon oben genutztes Beispiel $P_1 = (1, 0, 2, 2, 0, 0)$. Man beginnt bei dem Element $p_1(n)$ mit der höchsten Ordnung $n = 6$. Dieses Element hat grundsätzlich den Fehlstand 0. Von oben zählend hat das nächste entweder den Fehlstand 0 oder 1. Es wird dann entsprechend links oder rechts vom platzierten Element n angeordnet. Weil fortlaufend Elemente mit niedrigerer Nummer angeordnet werden, können die aktuellen Fehlstände festgelegt werden, ohne die bereits festgelegten Fehlstände zu verändern. So erhalten wir die Rekonstruktion der Tupel–Darstellung aus der Fehlstandtabelle in folgender Reihenfolge, wie in Tab. 1 gezeigt.

(x,x,x,x,x,0)	6
(x,x,x,x,0,0)	56
(x,x,x,2,0,0)	564
(x,x,2,2,0,0)	5634
(x,0,2,2,0,0)	25634
(1,0,2,2,0,0)	215634

Tabelle 2: Gewinnung der Tupel–Darstellung aus der Fehlstandtabelle

Zyklen–Schreibweise

Die *Zyklen–Schreibweise* lautet für unser Standardbeispiel

$$P_1 = (1\ 2)(3\ 5)(4\ 6) = (6\ 4)(5\ 3)(2\ 1)$$

Die Idee, dargestellt an unserem Beispiel, ist: „1 wird auf die 2 abgebildet, 2 wird auf die 1 abgebildet.“ Damit ist ein Zyklus geschlossen. Ähnlich wird 3 auf 5 abgebildet, 5 auf 3 abgebildet; Zyklus geschlossen. Entsprechend wird (4 6) gebildet. Es ist willkürlich, welches Element man als das dargestellte Start–Element eines Zyklus wählt. Auch ist die notierte Reihenfolge der Zyklen willkürlich.

Um eine *Eindeutigkeit der Darstellung* zu erhalten, setzt man die Elemente mit der höchsten Nummer jeweils an den Anfang und sortiert auch die Zyklen entsprechend ein.

Die Elemente, die unmittelbar auf sich selbst abgebildet werden (Fixpunkte), werden oft gar nicht notiert.

An der Zyklen–Schreibweise erkennt man, dass zweimaliges Anwenden dieser Permutation P_1 wieder zum Ausgangszustand führt. P_1 ist also seine eigene inverse Permutation:

$$P_1^{-1} = P_1$$

Die Zyklen–Schreibweise bietet weitere Vorteile, die in diesem Text nicht behandelt werden.

Permutationsmatrizen

Eine weitere Beschreibung von Permutationen gebraucht *Permutationsmatrizen.*

$$P_1 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$P_1 \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 5 \\ 6 \\ 3 \\ 4 \end{pmatrix}$$

Permutationsmatrizen haben in jeder Zeile und jeder Spalte genau eine 1; sonst enthalten sie nur Nullen.[4]

Operationen mit Permutationen

Kompositionen

Wie oft bei der Untersuchung von Abbildungen ist mit den Worten Komposition oder Verknüpfung das „Hintereinander–Ausführen“ zweier Abbildungen, hier Permutationen, gemeint.

Wichtig ist dabei, dass die Reihenfolge, in der die Abbildungen ausgeführt werden, beachtet wird.

Wir wählen als erklärendes Beispiel zusätzlich zum beschriebenen Beispiel P_1 eine Permutation P_2. In Zwei–Zeilen–Form gilt

$$P_1 = \begin{pmatrix} 1\ 2\ 3\ 4\ 5\ 6 \\ 3\ 1\ 6\ 5\ 2\ 4 \end{pmatrix}$$

$$P_2 = \begin{pmatrix} 1\ 2\ 3\ 4\ 5\ 6 \\ 5\ 1\ 4\ 3\ 2\ 6 \end{pmatrix}$$

$$P_1\ P_2 = \begin{pmatrix} 1\ 2\ 3\ 4\ 5\ 6 \\ 2\ 3\ 5\ 6\ 1\ 4 \end{pmatrix}$$

Wichtig ist, beim Operanden P_2 anzufangen:

$$1 \to 5 \to 2$$

$$2 \to 1 \to 3$$

$$3 \to 4 \to 5$$

$$4 \to 3 \to 6$$

$$5 \to 2 \to 1$$

$$6 \to 6 \to 4$$

[4] Für Mathematik–Liebhaber: Sie sind orthogonal, lassen sich also durch Transponieren invertieren

Invertieren

Dies ist zu verstehen als das Berechnen einer inversen Permutation P^{-1} zu einer gegebenen Permutation P, so dass die Komposition $P\ P^{-1}$ die identische Permutation ergibt.

$$P\ P^{-1} = Id$$

Mit „Papier und Bleistift“ kann man in der Zwei–Zeilen–Form die zweite Zeile umsortieren, so dass man dort die normale Anordnung 1 2 3 . . . n erhält. In der ersten Zeile entsteht dann die Inverse.

Ebenso einfach ist, die Permutationsmatrix aufzustellen und diese durch Transponieren zu invertieren.

Konjugieren

Bilden wir aus einer Permutation P_1 und einer weiteren P_2 den Ausdruck

$$\tilde{P_1} = P_2^{-1} P_1 P_2$$

so sprechen wir von einer *Konjugation*. Ähnlich wie in der Matrizenrechnung bildet diese Operation eine Permutation, die wie P_1 wirkt, aber mit „anderen Akteuren“. Die wird in der Zyklendarstellung deutlich. Weiteres sei auf einen späteren Beitrag verschoben.

Speicher und Programme

Zu klären ist, wie viel Speicherplatz für das Arbeiten mit Permutationen benötigt wird. Die Anzahl möglicher Permutationen von n Elementen ist durch $n!$ gegeben.[5]

$$n! = 1 \times 2 \times 3 \cdots \times n$$

So ist es im Prinzip möglich, alle für eine Anzahl n von Elementen möglichen Permutationen mit den Zahlen 0 bis $(n! - 1)$ zu nummerieren.

Die Funktion *Fakultät* wächst mit steigendem Argument sehr schnell.

Mit der Formel von Stirling [6] lässt sich die Fakultät für großes n auf eine einfache Weise asymptotisch berechnen.

$$n! \sim \sqrt{2\pi n} \left(\frac{n}{\mathrm{e}}\right)^n$$

Für Nichtmathematiker genügt es, diese Formel als gute Näherung für große n zu betrachten

Speicherbedarf für einen einfachen Demonstrator

Wenn man nur einen einfachen Demonstrator für Unterrichtszwecke benötigt, kann man sich entscheiden, mit Permutationen von maximal 16 Elementen zu arbeiten. Damit gibt es 16! = 2.092E13 verschiedene Permutationen. Weil $2^{64} = 1.845\text{E}19$, kann man in einem Speicherwort von 64 Bit diese Permutationen nummerieren. Weil $2^4 = 16$, kann man mit den zur Verfügung stehenden 4 Bit je Element, diese 16 Elemente eindeutig kennzeichnen. Ebenso kann man den Fehlstandvektor in einer Speicherzelle halten.

Organisation eines weitgehend unbeschränkten Systems

Für wissenschaftliche Anwendungen oder solche, die über elementare Lehraufgaben hinausgehen, wie das Testen von Sortier–Algorithmen, sollte man Permutationen mit einigen Tausend Elementen untersuchen können.

Allerdings steigt die Zahl der möglichen Permutationen dann wie die Fakultätsfunktion mit der Zahl der Elemente sehr stark an.

Damit ist man gezwungen, für die Nummerierung der Permutationen eine Stellenschreibweise in einem *Fakultät basierten System* zu verwenden oder eine andere Möglichkeit zu wählen, die mehr Zahlen unterscheiden kann, als eine direkte Darstellung mit einfachgenauen oder doppeltgenauen ganzen Zahlen[6].

Eine Permutation von n Elementen würde man dann als Vektor modellieren, der aus n ganzen Zellen gebildet wird. Zusätzlich muss noch die Zahl n der Elemente gespeichert werden.

Das stellt zwar eine Verschwendung von Speicherplatz dar, solange die Zahlen noch relativ klein sind — siehe Codebeispiel im Listing — ist aber in den Operationen ganz unkompliziert, weil nur das Standard–Wordset gebraucht wird.

Als Forth–Programmierer mag man versucht sein, einen *getrennten Stapel für Permutationen* einzurichten. Aber man muss dann Stapel–Einträge unterschiedlicher Länge verwalten. Es fragt sich auch, ob Programme aller Wandel– und Analyse–Prozeduren für alle Anwender relevant sind.

So entschied sich der Autor dieser Einführung, das Gesagte in nur wenigen Codezeilen zu illustrieren am Beispiel der Komposition von zwei Vektoren, die schlicht im Wörterbuch residieren.

Referenzen

[1] Wikipedia. Richard Dedekind — Wikipedia, die freie Enzyklopädie, 2021. [Online; Stand 13. April 2021].

[2] Wikipedia. Permutation — Wikipedia, die freie Enzyklopädie, 2021. [Online; Stand 13. April 2021].

[3] Wikipedia contributors. Permutation — Wikipedia, the free encyclopedia, 2021. [Online; accessed 13-April-2021].

[4] Wikipedia. Fehlstand — Wikipedia, die freie Enzyklopädie, 2021. [Online; Stand 13. April 2021].

[5] Wikipedia. Adam Ries — Wikipedia, die freie Enzyklopädie, 2021. [Online; Stand 13. April 2021].

[6] Wikipedia. Fakultät (mathematik) — Wikipedia, Die freie Enzyklopädie, 2021. [Online; Stand 15. April 2021].

[5] Im Deutschen gesprochen „n Fakultät“; Englisch: factorial, in Österreich auch Faktorielle. [6]

[6] Englisch: integer numbers

Listing

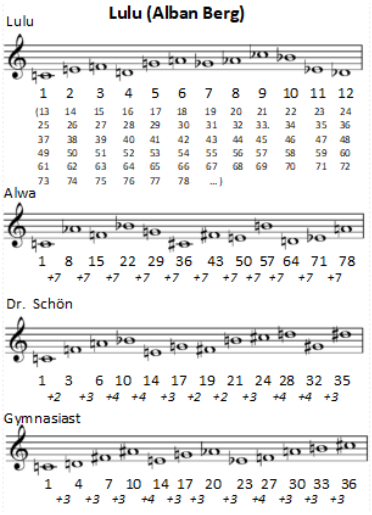
```
\ Beispielhafte Komposition zweier Vektoren


\ ---
\ Ein Marker erlaubt es, den Quellcode wiederholt
\ mit copy&paste auszuführen.

[IFDEF] permutations
permutations
[ENDIF]
marker permutations

\ Eine andere Technik ist es, Gforth und Quelldatei
\ von der Kommandozeile aufzurufen.
\ Dann startet es sowieso frisch.
\ ---


.( Wörter für Permutationen als Beispiele)
\ Damit nur die neu definierten Forthworte
\ angezeigt werden, kommen sie in ein eigenes
\ Wörterbuch.
vocabulary perm    perm definitions


\ Für die beispielhafte Komposition zweier
\ Vektoren zu einem Dritten werden
\ 10-dimensionale Vektoren definiert und mit
\ Werten vorbelegt.

\ Gebrauch der Vektoren:
\   k p1
\   Bringt Adresse des k-ten Elements von z. B.
\   p1 auf den Stapel.
\
\   k p1 @
\   Bringt Inhalt des k-ten Elementes von z. B.
\   p1 auf den Stapel.
\
\   w k p1 !
\   Schreibt w in das k-te Element von z. B. p1

\ Die Zahlenwerte sind Dezimalwerte.
decimal

\ Wir benutzen ein definierendes Wort, um die
\ Vektoren anzulegen.
\ Der Vektor bekommt 10 Elemente mit Zellgröße.
\ Sein Index ist auf k={0..9} festgelegt.
: 10-dim-vektor
  { x0 x1 x2 x3 x4 x5 x6 x7 x8 x9  -- }
  create
  x0 , x1 , x2 , x3 , x4 ,
  x5 , x6 , x7 , x8 , x9 ,
  does> swap cells + ( k -- kadr )  ;

\ Damit werden nun die drei Vektoren mit ihren
\ vorbelegten Werten definiert.
\ 0 1 2 3 4 5 6 7 8 9   Indexlineal
  0 3 1 6 5 2 4 0 0 0   10-dim-vektor p1
  0 5 1 4 3 2 6 0 0 0   10-dim-vektor p2
  0 0 0 0 0 0 0 0 0 0   10-dim-vektor p1p2
\ Verwendet werden sollen die Werte der Indizes
\ k={1..6}. Die übrigen sind mit null belegt.

\ Daraus wird die Komposition von p1 mit p2
\ berechnet und in p1p2 abgelegt.
: compos ( -- )
  7 1 do  \ ist k={1..6}
  i p2 @  \ holt das nächste k
    p1 @  \ holt damit das komponierte k
  i p1p2 !  \ und legt es ab.
  loop ;


\ Das Ergebnis der Komposition sollte nun
\ 2 3 5 6 1 4  lauten.
: writep1p2 ( -- )
  7 1 do i p1p2 @ dup . loop ;



\ ---

\ Ein Test für die Überprüfung der Vektoren.
: vdump ( -- ) \ Die Vektoren dumpen
  0 p1   10 cells dump
  0 p2   10 cells dump
  0 p1p2 10 cells dump
;

\ Und für meine Bequemlichkeit noch ein Alias.
  ' writep1p2 alias ..

\ Und Gforth soll nun anzeigen, was es getan hat.
words cr cr

\ finis -- checked by mk 02.08.2021: ok

```

Permutation in der Musik — Lulu

Bei der Suche nach Beispielen für die Anwendung von Permutationen stieß ich auf *Lulu*.

> „Alle, die Lulu lieben, sterben an dieser Liebe. Lulu ist den Männern Befriedigungsinstrument ihrer sexuellen Begierden, sie selbst aber bleibt dabei auf irritierende Art und Weise autonom und unangetastet und scheint dadurch umso attraktiver. (Wikipedia)“

Alban Berg verwendet in seiner Oper Permutationen in der Zwölftontechnik. Nach einem numerischen Auswahlmodus werden nacheinander einzelne Töne herausgenommen, bis eine neue Zwölftonreihe entsteht. Und das klingt dann so … `https://www.youtube.com/watch?v=oTCSxYCUe_c`



Angenehmes Blinken II

Robert Clausecker und Matthias Koch

Im ersten Teil des Artikels haben wir eine einfache Idee besprochen, basierend auf einem Sägezahngenerator, einer angenäherten Exponentialfunktion `bitexp` *und einem Sigma-Delta-Modulator eine LED angenehm blinken zu lassen. In diesem Teil wollen wir noch das Sahnehäubchen aufsetzen und aus dem Sägezahn-Blinken ein echtes Pulsieren machen. Dazu machen wir einen kleinen Exkurs in die Welt der Differentialgleichungen. Wer hätte gedacht, dass Blinken so vielseitig sein kann?*

Der erste Teil des Artikels[1] beschreibt ein paar einfache Tricks, wie man aus dem schroffen an-aus-an-aus einer LED ohne besondere Hardware-Unterstützung ein angenehmes Pulsieren machen kann. Die Grundidee ist dabei, die LED in einer Dreieckswelle blinken zu lassen, realisiert über einen Zähler und ein Inkrement, das bei Überlauf mit Hilfe der Absolutwertfunktion negiert wird. Diese Dreieckswelle wird zunächst über eine angenäherte Exponentialfunktion dem menschlichen Sehempfinden angeglichen und dann über Sigma-Delta-Modulation auf die LED ausgegeben.

So schön dieser Ansatz auch ist, hat die Dreiecksfunktion leider ihre Schwächen. Bei genauerem Hinsehen zerfällt die Illusion eines warmen Pulsierens und es wird klar, dass die LED nur geradlinig zwischen an und aus hin- und herpendelt. Aber auch das können wir besser machen.

Von Kreisen und Schüben

Wir beginnen mit etwas Geometrie. Angenommen, wir wollen einen Punkt P auf einer kreisförmigen Ortskurve um den Ursprung bewegen. Parametrisiert in der Zeit t hat der Punkt dann die Koordinaten:

$$P(t) = (\sin t, \cos t)$$

Das heißt, der Punkt startet bei Koordinaten $(0, 1)$ und bewegt sich auf der x-Achse nach dem Sinus und auf y-Achse nach dem Kosinus. Möchten wir diese Bewegung simulieren, so müssen wir uns anschauen, zu welcher Änderung der Koordinaten eine Änderung von t führt. Ist die Änderung h von t sehr klein, so können wir die Änderung von P durch die komponentenweise Ableitung der Ortskurve beschreiben. Es ist gemäß der Rechenregeln von Sinus und Kosinus:

$$P'(t) = (\sin' t, \cos' t) = (\cos t, -\sin t)$$

Damit ergibt sich eine einfache Annäherung der Koordinaten von P, wenn t um einen kleinen Betrag h fortgeschritten ist:

$$\begin{aligned} P(t+h) &= (\sin(t+h), \cos(t+h)) \\ &\approx (\sin t + h\cos t, \cos t - h \sin t) \end{aligned}$$

Da die Ableitungen der Koordinaten wieder auf die Koordinaten selbst zurückführen, sind diese Inkremente besonders leicht zu berechnen. Setzen wir jetzt h noch auf eine kleine Zweierpotenz $h = 2^{-c}$, so werden wir auch die Multiplikationen los und können diese durch günstige Schübe ersetzen:

$$\begin{aligned} x_{t+h} &= x_t + y_t \ggg c \\ y_{t+h} &= y_t - x_{t+h} \ggg c \end{aligned}$$

Auch in Forth ist das schnell implementiert:

```
3 constant schubweite

: rundschritt+ ( x y -- x' y' )
  swap over
  schubweite arshift +  \ x' = x + y  >>> c
  swap over
  schubweite arshift -  \ y' = y - x' >>> c
;
```

Dabei stellen sich mehrere interessante Dinge heraus: Selbst, wenn man hier mit endlicher Rechengenauigkeit und sogar mit Fixkomma- oder Ganzzahlen arbeitet, ist diese Iterationsvorschrift eine perfekte Bijektion. Jedem Punkt wir eindeutig ein rotierter Punkt zugeordnet, egal welches c man wählt. Dabei ist es kritisch, dass man in der Berechnung von y_{t+h} den bereits verschobenen Wert x_{t+h} statt x_t nimmt, da so ggf. verloren gegangene Genauigkeit wieder zurück in die Rechnung geschoben wird. Man kann so durch Rückrechnung der Iterationsvorschrift den Punkt auch ganz einfach rückwärts über seine Ortskurve schieben.

$$\begin{aligned} y_{t-h} &= y_t + x_t \ggg c \\ x_{t-h} &= x_t - y_{t-h} \ggg c. \end{aligned}$$

Und in Forth:

```
: rundschritt- ( x y -- x' y' )
  over
  schubweite arshift + \ y' = y + x  >>> c
  swap over
  schubweite arshift - \ x' = x - y' >>> c
  swap
;
```

Auch die richtige Wahl der Konstanten c ist von besonderem Augenmerk. Je höher c, desto langsamer folgt der Punkt seiner Ortskurve und desto höher ist die benötigte Genauigkeit, aber desto runder ist der Kreis. Ein kleines c braucht weniger Genauigkeit, führt jedoch zu eckigen Kreisen. Wichtig ist bei der Anwendung auf Ganzzahlen das Verhältnis des Kreisradius zur Schubweite – ist

[1] Heft 4d2021-01, S.17 ff

der Schub zu groß, werden die kleinen Koordinatenwerte komplett herausgeschoben und der Algorithmus bleibt auf dem Startwert stehen.

Eine Konsequenz dieser Eigenschaft ist, dass die Ortskurve jedes Startpunktes irgendwann wieder zu diesem zurückfindet. Eventuell nicht sofort, aber nach ein paar Ehrenrunden auf jeden Fall. Das heißt also, dass eine geschlossene Ortskurve gebildet wird und keine Spirale, die in den Ursprung stürzt, oder in die Unendlichkeit ausbüchst. Das erweitert die Nutzbarkeit auf einige spannende Anwendungsgebiete, wie z. B. der Rotation von Bildern.

Leider hat die Iterationsvorschrift gegenüber einem idealen Kreis einen gewissen Fehler. Der x-Koordinate wird auf dem I. und IV. Quadranten zu viel und auf dem II. und II. Quadranten zu wenig zugeschlagen. Analog gilt dies für die y-Koordinate auf dem I./II. und III./IV. Quadranten. Als Resultat „eiert“ der Kreis ein wenig, sein Ursprung scheint in Richtung des I. Quadranten versetzt (Abb. 1).

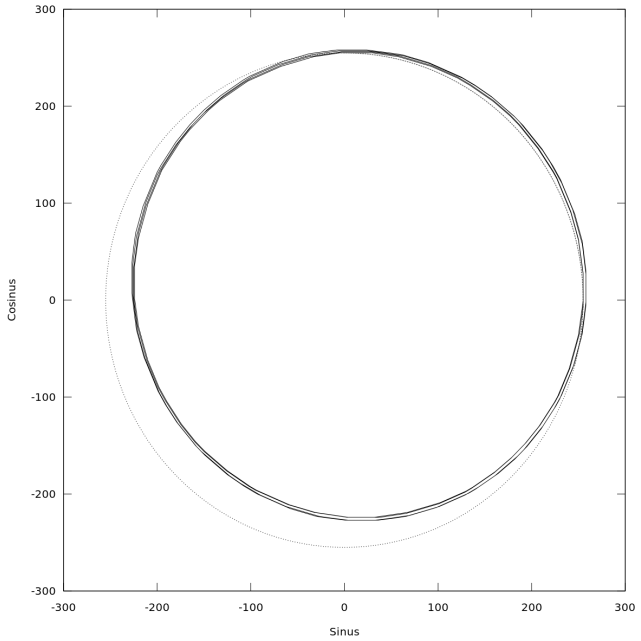


Abbildung 1: Plot einer Ortskurve

Man kann diesem Eiern beikommen, indem man zur Erhöhung der Genauigkeit das x-Inkrement in zwei Schritten berechnet. So heben sich die Rechenfehler weitestgehend auf und der Kreis wird zentrierter, behält aber seine Bijektivität:

$$\begin{aligned} x_{t+h/2} &= x_t + y \ggg (c+1) \\ y_{t+h} &= y_t - x_{t+h/2} \ggg c \\ x_{t+h} &= x_{t+h/2} + y_{t+h} \ggg (c+1) \end{aligned}$$

```
: kreisschritt+ ( x y -- x'' y' )
  swap over
  schubweite 1+
        arshift + \ x'  = x  + y  >>> c+1
  swap over
  schubweite
        arshift - \ y'  = y  - x' >>> c
  swap over
  schubweite 1+
        arshift + \ x'' = x' + y' >>> c+1
  swap
;
```

Auch die Rückrechnung ist wieder möglich:

$$\begin{aligned} x_{t-h/2} &= x_t - y \ggg (c+1) \\ y_{t-h} &= y_t + x_{t-h/2} \ggg c \\ x_{t-h} &= x_{t-h/2} + y_{t-h} \ggg (c+1) \end{aligned}$$

```
: kreisschritt- ( x y -- x'' y' )
  swap over
  schubweite 1+
        arshift -  \ x'  = x  - y  >>> c+1
  swap over
  schubweite
        arshift +  \ y'  = y  + x' >>> c+1
  swap over
  schubweite 1+
        arshift -  \ x'' = x' - y' >>> c+1
  swap
;
```

Ausprobiert werden können die Rechenschritte mit diesem kleinen Helferlein, welches sich auch zum Zeichnen von runden Dingen eignet, wenn statt der Ausgabe der Koordinaten jeweils ein Punkt gezeichnet wird. Zum Zeichnen muss allerdings noch der Mittelpunkt durch Addition geeigneter Offsets verschoben werden.

```
: kreisprobe ( startx starty -- )
  2dup 2>r cr
  begin
  over . dup .  \ Aktuelle Koordinaten anzeigen
  cr rundschritt+ \ Einen Schritt weitergehen
  2dup 2r@ d=      \ Zurück am Ausgangspunkt?
  until
  2rdrop 2drop
;
```

Vom Eiern und Atmen

Und nun die Idee, warum wir oben so rumgeeiert haben. Eigentlich interessiert uns fürs „atmende“ Blinken so ein Kreis nicht wirklich. Aber wenn wir nur die y-Koordinate nehmen, dann haben wir eine recht genaue Annäherung einer Kosinus-Funktion, die im Vergleich zur Dreieckskurve für ein viel wärmeres Blinken sorgt. Das Prinzip ist in Abb. 3 dargestellt, und das Resultat in Abb. 4, bei den Listings.

Die Implementierung ist sehr ähnlich zur Dreieckskurve aus dem ersten Teil im vorherigen Heft und braucht auf ARM sogar überraschenderweise noch einen Maschinenbefehl weniger. Als kleiner Bonus ist die Hauptschleife komplett frei von Sprüngen und hat so eine exakt vorhersehbare Laufzeit. Ideal für einen kleinen Lückenbüßer in taktgenauen Routinen!

Die Listings

Im Listing könnt ihr sehen, wie es gemacht worden ist. Listing I ist für den ARM-Assembler. Auch auf RISC-V ist der Algorithmus einfach zu realisieren, benötigt jedoch zwei Maschinenbefehle mehr als die Variante mit der Dreieckswelle – Listing II. Und zum Ausprobieren nochmal eine Implementierung in Forth, vorgesehen für eine Zellengröße von 32 Bit – Mecrisp, Listing III. Der Mecrisp-Quelltext ist portabel und sollte auf jedem Forth mit 32-Bit-Zellengröße laufen (zumindest aber auf allen entsprechenden Mecrisp-Varianten).

Wir haben den Code für die ARM-Boards auf einem Raspberry Pi Pico und für RISC-V auf dem Icestick mit dem FemtoRV32-Quark[2] getestet – Abb. 2.

Diese Assembler-Quellen sind übrigens in den veröffentlichten Mecrisp-Paketen enthalten – die Links sind weiter unten angegeben.

Historisches und Weiteres

Das hier vorgestellte Verfahren geht gemeinhin auf den KI-Pionier und Turing-Preisträger Marvin Minsky (1927-2016) zurück, der dies am MIT als *Display Hack* für ein frühes Graphik-Display erfand und im Memo `HAKMEM` von 1972 publizierte.

Dem geneigten Leser empfehlen wir, auch mal mit h-Werten zu experimentieren, die zunächst sinnlos erscheinen. Was passiert, wenn $h = 2$? Was, wenn die Präzision sehr gering ist?

Neben diesem Verfahren gibt es selbstverständlich noch weitere Möglichkeiten, eine Sinuskurve schrittweise zu simulieren. Ein besonders nützliches Verfahren, insbesondere zum Füllen von Sinus-Tabellen basiert auf der geschickten Verwendung des Additionstheorems für den Sinus. Fangen wir mit einer einfachen Rechnung an:

$$\begin{aligned}\sin(x-h)+\sin(x+h) &= (\sin x\cos h-\cos x\sin h)+\\ &\quad (\sin x\cos h+\cos x\sin h)\\ &= 2\sin x\cos h\end{aligned}$$

Wenn wir nun x um h verschieben, erhalten wir eine einfache Vorschrift, um $\sin(x+2h)$ aus $\sin x$ und $\sin(x+h)$ zu bestimmen. Damit kann man bequem alle Werte des Sinus in Inkrementen von h bestimmen. Der Startwert $\sin h$ und der Koeffizient $2\cos h$ müssen vorab berechnet werden:

$$\sin(x+2h) = 2\cos h\sin(x+h)-\sin x.$$

Dieses kleine Beispiel gibt vier Schwingungen einer Sinuswelle mit einer Periode von 256 Punkten aus:

```
0,0 2variable vorletzter
360,0 256,0 f/ sin 2variable letzter
360,0 256,0 f/ cos
            2,0 f* 2constant wellenschritt

: sinuswelle ( -- )
  cr
  0 . vorletzter 2@ f. cr
  1 . letzter    2@ f. cr

  1024 2 do
    wellenschritt letzter 2@ f*
    vorletzter 2@ d-
      i . 2dup f. cr
      letzter 2@ vorletzter 2!
    letzter 2!
  loop
;
```

Links

`mecrisp-stellaris-2.5.9a/stm32f051/tinyblinky/`

`mecrisp-quintus-0.36-experimental/hx1k/tinyblinky/`

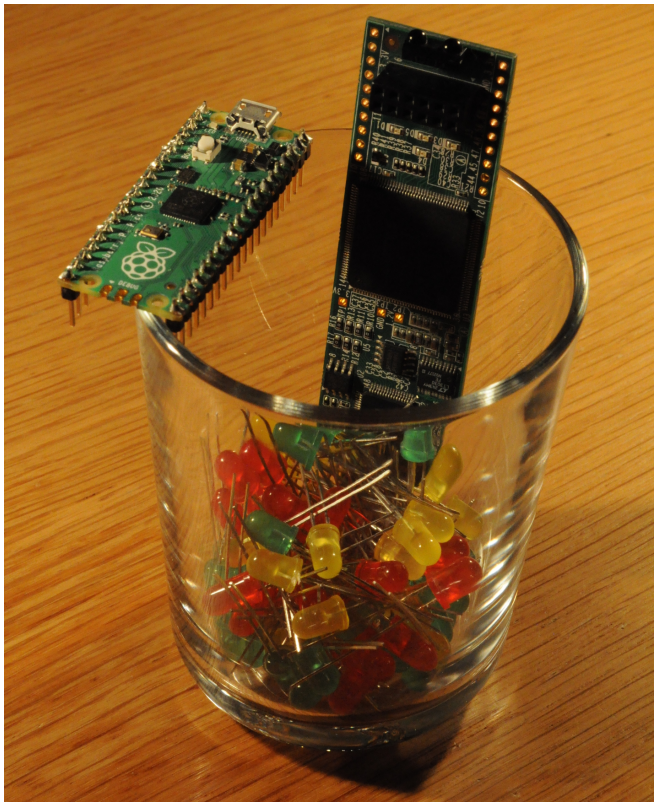


Abbildung 2: Raspberry Pi Pico (li), Icestick (re), darunter im Glas LEDs

[2] Matthias Koch, FemtoRV32-Quark – ein RISC-V in 400 Zeilen Verilog; Heft 4d2021-02, S. 15 ff

Listing I – ARM

```
## ARM - Raspberry Pi Pico

   ldr r2, =leds @ LED register

   movs r5, 1                   @ Set up initial x, y for Minsky circle algorithm
   lsls r6, r5, 19

@ ---------------------------------------------------------------------------
breathe_led: @ Generate smooth breathing LED effect
@ ---------------------------------------------------------------------------

    @ Register usage:

    @ r0 : Unused
    @ r1 : Scratch
    @ r2 : Initialised with IO address for GPIO
    @ r3 : Scratch
    @ r4 : Unused
    @ r5 : Minsky circle alg x = sin(t)
    @ r6 : Minsky circle alg y = cos(t)
    @ r7 : Phase accumulator for sigma-delta modulator

                                 @ Minsky circle algorithm x, y = sin(t), cos(t)
    asrs r1, r5, 17              @ -dx = y >> 17
    subs r6, r1                  @  x += dx
    asrs r1, r6, 17              @  dy = x >> 17
    adds r5, r1                  @  y += dy

    asrs r1, r6, 13              @ -49 <= r4 <= 64   --> scaled cos(t)
    adds r1, 167                 @ 118 <= r4 <= 231  --> scaled cos(t) with offset

    movs r3, 7                   @ Simplified bitexp function.
    ands r3, r1                  @   Valid for inputs up to 231
    adds r3, 8                   @   Gives too small values above 231
    lsrs r1, r1, 3               @   Input  in r1
    lsls r3, r1                  @   Output in r3

    subs r7, r3                  @ Sigma-Delta phase accumulator
    sbcs r3, r3                  @ Sigma-Delta output through carry, which is inverted for subs

   str  r3, [r2]                 @ Set output accordingly
   b.n breathe_led
```

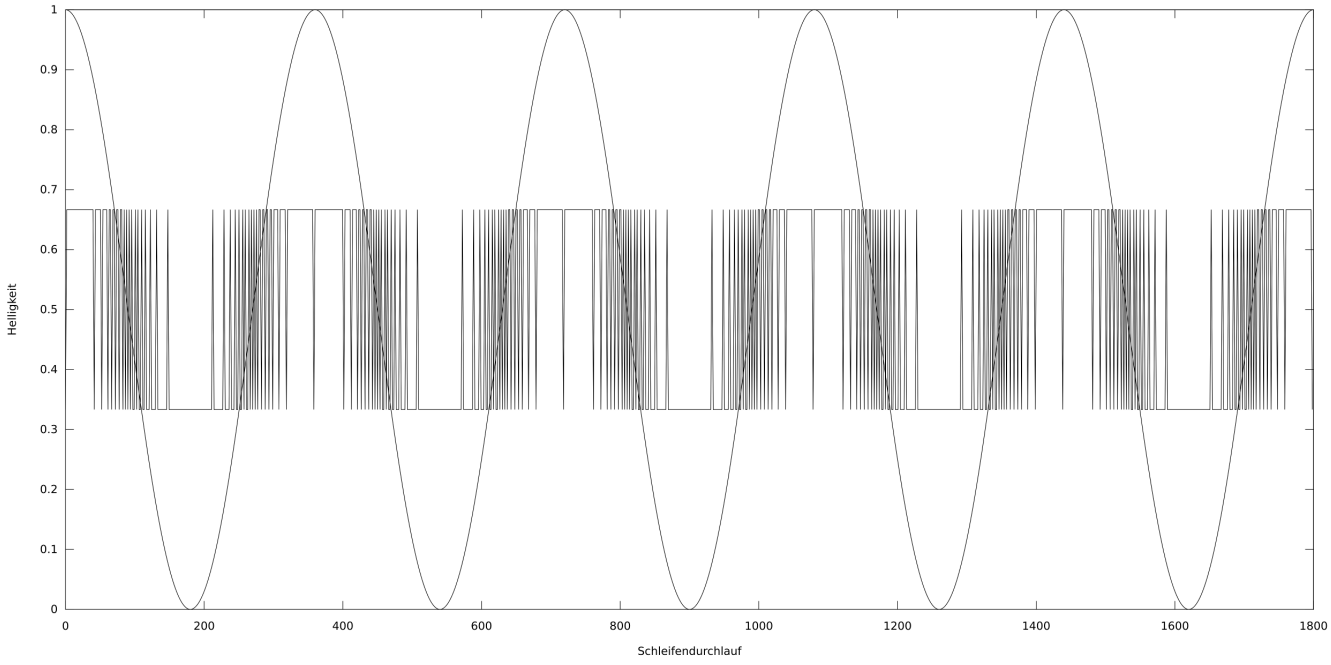


Abbildung 3: Das Prinzip der Sigma-Delta-Modulation in Liniendarstellung.

Listing II – RISC-V

```
## RISC-V - Icestick, FemtoRV32-Quark

  li x8, leds               # LED register
  li x10, 1                 # Set up initial x, y for Minsky circle algorithm
  slli x11, x10, 19

# ---------------------------------------------------------------------------
breathe_led: # Generate smooth breathing LED effect
# ---------------------------------------------------------------------------

    # Register usage:

    # x8  : Initialised with IO address for GPIO
    # x9  : Phase accumulator for sigma-delta modulator
    # x10 : Minsky circle alg x = sin(t)
    # x11 : Minsky circle alg y = cos(t)
    # x12 : Scratch
    # x13 : Scratch
                                # Minsky circle algorithm x, y = sin(t), cos(t)
    srai x12, x10, 17           # -dx = y >> 17
    sub  x11, x11, x12          #  x += dx
    srai x12, x11, 17           #  dy = x >> 17
    add  x10, x10, x12          #  y += dy

    srai x12, x11, 13           # -49 <= r4 <= 64   --> scaled cos(t)
    addi x12, x12, 167          # 118 <= r4 <= 231  --> scaled cos(t) with offset

    andi x13, x12, 7            # Simplified bitexp function.
    addi x13, x13, 8            #   Valid for inputs up to 231
    srli x12, x12, 3            #   Gives too small values above 231
    sll  x13, x13, x12          # Input in x12, output in x13

    add  x9, x9, x13            # Sigma-Delta phase accumulator
    sltu x13, x9, x13           # Sigma-Delta output on overflow
    sub  x13, zero, x13         # (0, 1) --> (0, -1)

   sw  x13, 0(x8)               # Set all LEDs at once
   j breathe_led
```

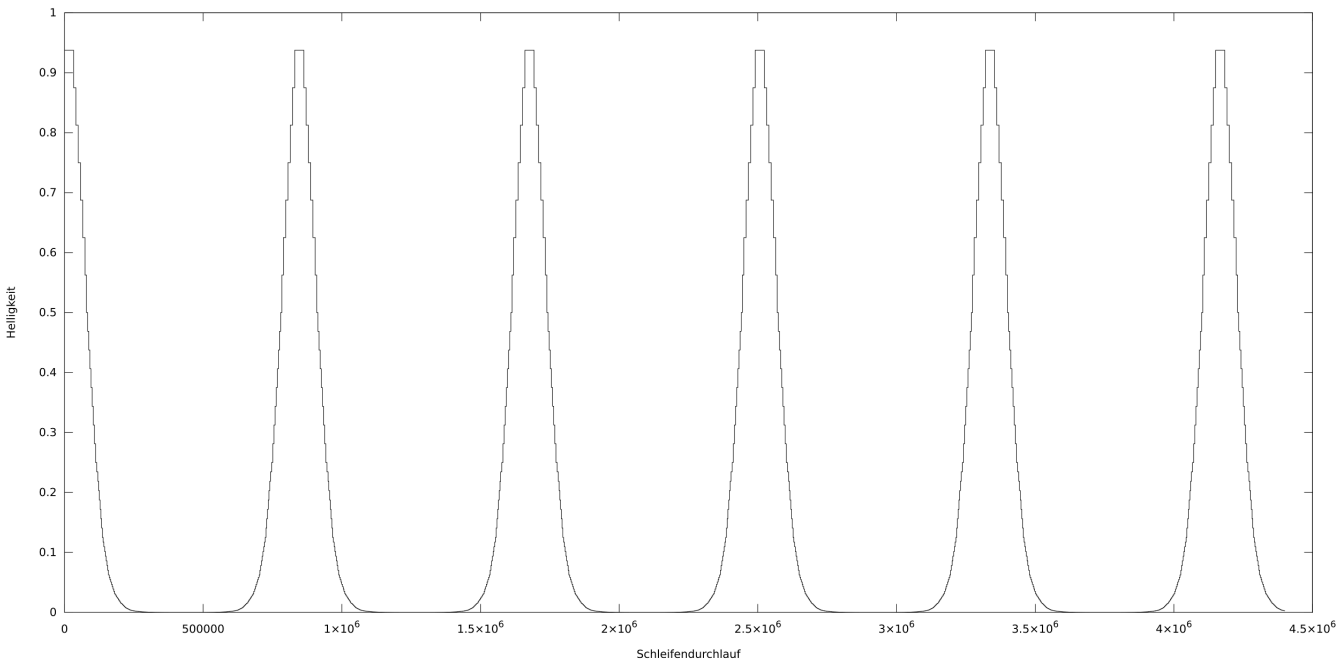


Abbildung 4: Die Helligkeitskurven der Blinkfolgen.

Listing III – Forth

```
1   ## Mecrisp
2
3   0 variable sdm-phase
4
5   : >sdm ( u -- )
6     sdm-phase @ ( u alt )
7     tuck +      ( alt neu )
8     dup sdm-phase !
9     u> led-out-gpio !
10  ;
11
12  : minskyblinker ( -- )
13
14    0
15    1 19 lshift \ Der Kreis beginne bei x=0, y=$00080000
16
17    begin
18
19      \ Einen Schritt auf dem Kreis gehen
20      swap over 17 arshift + swap \ x' = x + y  >>> 17
21           over 17 arshift -      \ y' = y - x' >>> 17
22
23      dup 13 arshift \ -49 <= TOS <=  64 --> Skalierter Cosinus
24      167 +          \ 118 <= TOS <= 231 --> Skalierter Cosinus mit Offset
25
26      dup 7 and 8 +  \ Vereinfachte Bitexp-Funktion,
27      swap 3 rshift  \   gültig für Eingabewerte
28      lshift         \   von 0 bis einschließlich 231
29
30      >sdm
31
32    key? until
33    2drop
34
35    0 led-out-gpio ! \ Leuchtdiode wieder ausschalten
36  ;
37
```

Swap und das Blinken

Tja, ich sitze immer noch bei Michael im Regal. Es ist schon ein halbes Jahr her, dass ich von Matthias Koch weitergegeben worden bin an den Boten. Damals wollte ich zur Forthtagung. Doch die hat nicht stattgefunden — Corona–Pandemie. Veranstaltungen, bei denen sich Menschen hätten versammeln können, sind untersagt gewesen. Und nun, im Herbst 2021, immer noch so. Auch in diesem Jahr wieder keine Jahrestagung.

Was bleibt mir übrig? Nachts schaue ich aus dem Fenster hinaus zu den blinkenden Sternen, und den Fixsternen auch. Beeindruckend dieses Firmament. Die Milchstraße ist von hier aus, im Ruhrgebiet, nicht zu sehen. Zu hell ist die Umgebung und der Dunst. Aber das Bild unserer Milchstraße steht gleich neben mir. Die Bezeichnung entstammt übrigens dem gleichbedeutenden altgriechischen (galaxías) und geht auf eine antike Sage zurück, wonach es sich dabei um die verspritzte Milch (gála) der Göttin Hera handelt, als diese Herakles stillen wollte.

Da hat sich das Weltbild doch schon sehr gewandelt seither — von anthropozentrisch über heliozentrisch zu astronomisch. Da, wo der Bleistift hinzeigt, ist unser Sonnensystem. Mit einem Durchmesser von 200.000 Lichtjahren ist unsere Galaxie eine der größeren. Und davon gibt es viele. *Hubbles Ultra–Deep–Field* in einem dreizehnmillionsten Teil unseres Himmels allein zeigte schon rund 10.000 Galaxien. Wir sind nicht allein! Also keine Sorge, das Leben an sich geht weiter. Swap

Heap und TmpHeap im VolksForth

Carsten Strotmann

VolksForth besitzt eine für Forth–Systeme ungewöhnliche Einrichtung: den Heap. Der Heap ist ein spezieller Speicherbereich für Daten des Dictionary. Die primäre Anwendung des Heap besteht in der Möglichkeit, kopflose Wort–Definitionen zu erzeugen.

Wird im VolksForth einer Wort–Definition ein Pipe–Symbol | vorangestellt, so wird der Kopf des Wortes (Block–Feld, Link–Feld, Name ...) im Speicherbereich des Heaps abgelegt, der Körper des neuen Wortes jedoch wie gewohnt im normalen Dictionary. Diese Wörter können im Folgenden ganz normal benutzt werden. Zu einem Zeitpunkt im Kompilierprozess kann der Heap gelöscht werden (mittels `clear`). Dabei werden die Header–Informationen der Forth–Wörter auf dem Heap aus der Suchkette des Dictionarys entfernt. Dabei bleiben die Körper der Wörter im Dictionary erhalten, nur die Header werden entfernt. Solche „headerless"–Words gibt es auch in anderen Forth–Systemen, jedoch werden diese dort separat erzeugt (z. B. mit `:name`) und der Programmierer muss mit den Execution–Token jonglieren. Bei VolksForth verwendet man Wörter mit Header auf dem Heap bis zum `clear` ganz normal mit ihrem Namen. Der Speicherplatz der Header dieser Wörter wird im resultierenden Programm eingespart, was speziell für Systeme mit geringem Programmspeicher nützlich ist. Die headerless Wörter funktionieren weiterhin innerhalb der bestehenden Definitionen, können aber nach dem Löschen des Heap nicht mehr gefunden und daher nicht in neue Definitionen eingebaut werden. Bei der Benutzung des Heaps kann der Programmierer lange und sprechende Wort–Namen für interne Funktionen benutzen, ohne das diese langen Namen sich negativ auf den freien Programmspeicher auswirken.

VolksForth benutzt intern beim Kompilieren des Kernels den Heap, viele kernel–internen Worte werden auf dem Heap definiert und die Header später verworfen.

Programmcode auf dem Heap

Wenn der Dictionary–Zeiger (Dictionary–Pointer `dp`) auf den Heap „umgebogen" wird, so werden alle neuen Definitionen innerhalb des Heap–Speichers kompiliert, und zwar Header *und* Programmcode. Sinnvoll ist dies bei Erweiterungen des Forth–Compilers, welche nicht in das resultierende Programm aufgenommen werden. Ein gängiges Beispiel ist der Forth–Assembler im VolksForth: wird der Assembler benutzt, um Code–Definitionen zu erstellen, so ist es selten gewünscht, das der Assembler ein Bestandteil der resultierenden Applikation wird. Der Assembler wird daher komplett auf den Heap geladen, benutzt und danach wieder aus dem Heap und damit aus dem Speicher entfernt. Die mit dem Assembler erstellten Code–Worte verbleiben in der Anwendung, der Assembler jedoch nicht.

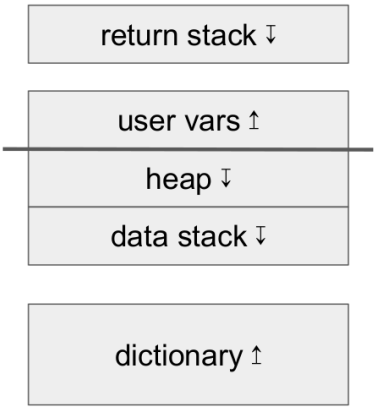


Abbildung 1: Lage des „Heap" im VolksForth

TmpHeap

Die klassische Implementation des Heaps im VolksForth erlaubte keine Granularität beim Löschen des Heaps. Entweder der Heap war in Benutzung, oder er wurde gelöscht und damit waren alle Wort–Definitionen auf dem Heap verschwunden.

Bei der Portierung des VolksForths und des in VolksForth geschriebenen C–Compilers für Commodore 6502–Rechner auf den neuen Commander X16–Rechner wurde Philip Zembrod der Programmspeicher im Commander X16 knapp (der Commander X16 hat gegenüber dem C64 oder dem C16 weniger zusammenhängendes freies RAM zur Verfügung). Der Platz auf dem VolksForth Heap reichte für die Übersetzung des C–Compilers nicht aus.

Die Lösung ist ein 2. Heap–Speicher, TmpHeap (temporärer Heap) genannt. Dieser TmpHeap kann optional zu dem normalen Heap hinzugeladen werden. Dabei kann, je nach Situation des freien Speichers im Forth–System, dieser TmpHeap einen eigenen Speicherbereich belegen, oder aber auf dem regulären Heap liegen. Es ist auch möglich, einen Dummy–TmpHeap zu laden, bei dem die TmpHeap–Forth–Wörter nur Platzhalter sind und keine spezielle Funktion ausführen.

Der TmpHeap kann unabhängig vom klassischen VolksForth–Heap benutzt und auch gelöscht werden. Die internen Worte eines Forth–Moduls werden z. B. auf dem TmpHeap gelegt, während die Header der Schnittstellen–Worte des Moduls auf dem normalen Heap verbleiben. Nach dem Laden des Moduls kann der TmpHeap gelöscht, und damit Speicher für ein weiteres Modul freigegeben werden. Die Wörter auf dem normalen Heap bleiben bis

zum Ende des Kompilier–Durchgangs bestehen und werden nur zum Abschluss gelöscht, so dass diese nicht in der fertigen Applikation Speicher verbrauchen. Mit diesem zweistufigen Konzept lassen sich größere Programme in Systemen mit geringem Speicher laden.

Wird einer neuen Forth–Definition ein doppeltes Pipe–Symbol vorangestellt, so wird der Header dieser Forth–Definition auf dem TmpHeap abgelegt:

```
|| : tmp-heap-hallo ." Hallo vom TmpHeap" cr ;
```

Sollen gleich mehrere Forth–Definitionen auf dem TmpHeap landen, oder sogar ein komplettes Quellcode–Modul, so kann die Benutztung des TmpHeap mit dem Wort `||on` an– und mit dem Wort `||off` abgeschaltet werden.

Inkompatible Änderungen

Um den TmpHeap zu implementieren, wurde eine inkompatible Änderung am VolksForth–Kern vorgenommen: Die Variable `?head` hatte bisher gesteuert, ob und wieviele folgende Wort–Header auf den Heap kompiliert werden sollen. Bei der Kompilierung mehrerer Wörter war dieses Schema jedoch fehleranfällig, da eine unbedachte Änderung der Anzahl der Wörter im Quellcode dazu führen kann, das die falschen Wörter auf dem Heap landen (oder nicht landen).

In der neuen Implementation von Philip wurde `?head` durch die Variable `?heapmove-xt` ersetzt. Ist der Wert von `?heapmove-xt` gleich Null, so wird der Wort–Kopf einer neuen Definition normal in das Haupt–Dictionary gespeichert. Alternativ kann in dieser Variablen der Execution–Token eines Wortes gespeichert werden, welches den Header einer neuen Wort–Definition verschiebt. Und dies können derzeit die Wörter `heapmove` und `heapmove1x` sein, welche den Header des neuen Wortes auf den Heap verschieben (`heapmove1x` verschiebt nur einen Header und setzt danach `?heapmove-xt` wieder auf den Wert „Null“). Oder aber die neuen Wörter des TmpHeap (`tmp-heapmove` und `tmp-heapmove1x`), welche den Header auf den TmpHeap verschieben. Durch die Implementation per Execution–Token kann dieser Mechanismus in Zukunft für andere Funktionen benutzt werden, welche den Header eines neuen Wortes manipulieren.

Status VolksForth 3.9.3

Der neue TmpHeap ist derzeit im VolksForth für die Commodore 8–Bit–Maschinen (C64, C16, Plus4, Commander X16) implementiert. Dabei gibt es drei unterschiedliche Implementationen, welche auch als Vorlagen für die Implementationen auf andere Plattformen benutzt werden können:

- `notmpheap.fth` — Eine Dummy–Implementation, dabei werden alle TmpHeap–Daten auf dem normalen Heap gespeichert und können nicht unabhängig vom Heap gelöscht werden. Diese Implementation hat keinerlei Overhead und kann auf Systemen benutzt werden, bei denen alle Definitionen für den TmpHeap in den normalen Heap passen.
- `tmpheap.fth` — In dieser Implementation wird der TmpHeap innerhalb des Speicherbereiches des Heap reserviert. Der TmpHeap kann unabhängig von Heap gelöscht werden.
- `x16tmpheap.fth` — Implementierung des TmpHeap für den Commander X16–Rechner. Aufgrund des fragmentierten Speichers beim Commander X16 wird der TmpHeap innerhalb eines 8–K–Speicherbereiches implementiert, welcher ansonsten nicht von VolksForth benutzt wird.

Die anderen VolksForth–Implementationen befinden sich derzeit noch auf der Versionsnummer 3.9.0 oder 3.9.1 und werden den TmpHeap beim Versionssprung auf 3.9.3 bekommen (Hilfe bei der Aktualisierung dieser Versionen ist immer gerne willkommen).

Glossary

- `| ( -- )` Lädt bei der Ausführung `?HEADMOVE-XT` mit dem Execution–Token, welcher dafür sorgt, dass der nächste erzeugte Name nicht im normalen Dictionary–Speicher angelegt wird, sondern auf dem Heap.
- `|| ( -- )` Lädt bei der Ausführung `?HEADMOVE-XT` mit dem Execution–Token, welcher dafür sorgt, dass der nächste erzeugte Name nicht im normalen Dictionary–Speicher angelegt wird, sondern auf dem TmpHeap.
- `||on ( -- )` Alle neu definierten Namen werden bis zur Ausführung von `||off` auf dem TmpHeap abgelegt.
- `||off ( -- )` Löscht die Variable `?HEAPMOVE-XT`; alle nachfolgenden Definitionen werden in dem normalen Dictionary abgelegt.
- `clear ( -- )` Löscht alle Namen und Worte auf dem Heap und auf dem TmpHeap, so dass vorher mit `|` definierte Worte nicht mehr in neuen Definitionen benutzt werden können. Auf diese Weise kann der Geltungsbereich von Forth–Wörtern eingeschränkt werden.
- `tmp-clear ( -- )` Löscht alle Namen und Worte auf dem TmpHeap, so dass vorher mit `||` definierte Worte nicht mehr in neuen Definitionen benutzt werden können.
- `heap ( -- addr )` Liefert den Anfang des freien Heap–Speichers (Heap–Pointer), wird z. B. von `hallot` geändert.
- `hallot ( n -- )` Reserviert `n` Bytes auf dem Heap (oder gibt bei einem negativen `n` Heap–Speicher frei). Dabei wird der Stack verschoben, sowie der Anfang des Heaps.

- `tmp-hallot ( n -- )` Reserviert `n` Bytes auf dem TmpHeap (oder gibt bei einem negativen `n` Speicher auf dem TmpHeap frei).
- `heap? ( addr -- flag )` Liefert ein `TRUE`–Flag, wenn `addr` ein Byte im Heap adressiert, ansonsten `FALSE`.
- `halign ( -- )` Passt bei Architekturen mit Speicher–Alignment–Anforderungen den Heap–Zeiger an, so dass die nächste Zuteilung von Heap–Speicher an einer geraden (aligned) Speicheradresse beginnt.
- `tmpheap[ ( -- addr )` User–Variable mit dem Zeiger auf den Anfang des TmpHeaps.
- `tmpheap> ( -- addr )` User–Variable mit dem Zeiger auf den nächsten freien Speicherplatz im TmpHeap.
- `]tmpheap ( -- addr )` User–Variable mit dem Zeiger auf das Ende des TmpHeap.

Links

Alles in einem Repository: `https://github.com/forth-ev/VolksForth/tree/master/6502/C64`

Der Commander X16

In Carstens Artikel wird ein neuer Retro–Computer mit dem Namen Commander X16 erwähnt. Das hat meine Neugierde geweckt. Warum braucht die Welt einen weiteren 8–Bit–Rechner und wie ist der Stand des Projekts?

Der Traum

David Murray (YouTube–Kanal: The 8–Bit Guy) wird häufig gefragt, welchen Retro–Computer er empfiehlt, um in das Hobby einzusteigen. Mit der Antwort tut er sich dann schwer, da alte Originalgeräte inzwischen rar werden und dementsprechend kostspielig sind. Mit der Zuverlässigkeit alter Systeme ist es auch nicht gut bestellt. Man weiß nie, wann der geliebte Veteran ausfällt. Die Ersatzteilversorgung gestaltet sich zunehmend schwieriger.

Also dachte David darüber nach, was ihn fasziniert an den 8–Bit–Maschinen und wie sein zu bauender „Traumcomputer“ ausgestattet sein sollte.

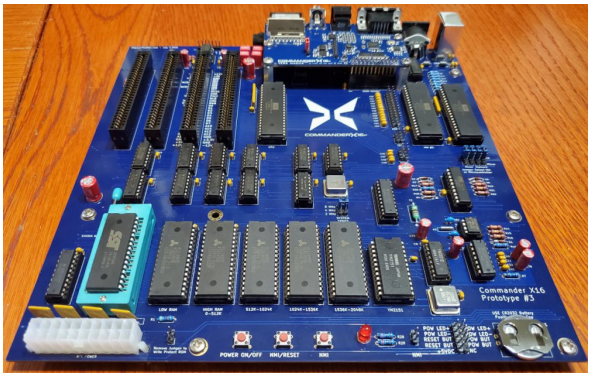
Faszinierend: +++ Direktzugriff auf die Hardware mit `peek` und `poke` (Forth: `@ !`) und sofortige Auswirkung (Zeichen erscheint, Ton erklingt) +++ Eine Person kann alle Komponenten (Hardware/Software) verstehen +++ Nach Einschalten sofort einsatzbereit +++ Keine Updates +++ Beschränkt, deshalb wird Kreativität gefördert +++ Komplettes Benutzerhandbuch ist dabei +++

Frustrierend: +++ Lange Ladezeiten von Kassette und Diskette +++ Tastaturen sind oft billig und haben kein Standard–Layout +++ Nicht für produktives Arbeiten geeignet (nur 40 Zeichen/Zeile) +++

Davids Designvorgaben: +++ Moderne, handelsübliche Bauteile +++ VGA– oder HDMI–Video +++ „Richtige“ 6502–CPU +++ Keine FPGAs oder MCUs +++ Hat Commodore Basic eingebaut +++ Vernünftige Tastatur +++ Datenspeicherung auf SD–Karte +++ Preiswert: 50 – 100 USD +++

Die Realität

Erstaunlich, wie schnell einen die Wirklichkeit einholt. Schnell wurde klar, dass es ohne FPGAs nicht gehen würde, da es keine Video– und Audio–Chips mit 8–Bit–Datenbus mehr gibt (schon gar nicht im DIP–Gehäuse).



Im Bild des aktuellen Prototypen ist oben eine kleine Huckepack–Platine zu erahnen, die mittels FPGA VGA–Video und ein mehrstimmiges Audiosignal erzeugt.

Stand des Projektes

Das Entwicklerteam kämpft noch gegen hartnäckige Fehler in den SD–Karten– und PS/2–Tastatur–Routinen. Wer den X16 kaufen will, muss sich also noch in Geduld üben. Der Vertrieb des X16 soll als Bausatz erfolgen (200 – 300 USD). Handgelötete Fertiggeräte kosten 100 – 200 USD extra. Es gibt übrigens auch einen X8 (reine FPGA–Version mit stark reduzierten Features) als Prototypen. Ob der weiterentwickelt wird, ist gerade Gegenstand der Diskussion. So ein Gerät könnte dann auch den anfangs angepeilten Preis von unter 100 USD erreichen.

Schlussbemerkung

Wie schwer kann es schon sein, einen 8–Bit–Kleinrechner zu entwickeln und zu vertreiben? Sehr schwer! Das Projekt läuft jetzt seit 2,5 Jahren. Vieles ist schon geschafft, aber es ist noch genug zu tun. David hat fähige Mitstreiter an seiner Seite und eine treue Fangemeinde. Das hilft ungemein. Ich wünsche ihm Erfolg.

Ist der Rechner etwas für mich? Wenn mich beim Start anstatt des drögen „Ready.“ ein VolksForth mit „ok.“ anlachen würde, gäbe es kein Halten mehr: „Shut up and take my money!“ :-) Wolfgang Strauß

VolksForth auf dem Commander X16

Philip Zembrod

Der Commander X16 ist ein neuer, in Entwicklung befindlicher Commodore–inspirierter und teilweise –kompatibler 65C02–Heimcomputer einer Gruppe von Enthusiasten um David Murray[1]*, der in der Retro–Szene derzeit einige Anziehungskraft entwickelt und eine aktive Community inspiriert. Entsprechend groß war meine Motivation, sowohl mit VolksForth als auch mit meinem in VolksForth implementierten kleinen C–Compiler* cc64 *auf dem X16 mitzuspielen. Aufbauend auf den 3 Varianten C64, C16–32k und C16–64k, die VolksForth schon unterstützte, und mit bequemen make–Regeln sowohl für den Targetcompiler als auch für Tests sollten die Schwierigkeiten nicht allzu groß sein.*

Portierung von VolksForth

Das Erbgut des *X16* stammt vom *VC20* und *C64*. Sein erweitertes Basic baut auf *Basic 2.0* auf, und sein „Kernal“[2] auf dem des C64. Der naheliegende Ausgangspunkt für den VolksForth–Port war damit die C64–Variante. Die Speicheraufteilung des X16:

$0000—$9EFF Hier liegt RAM, gefolgt von 256 Bytes I/O–Bereich mit zwei 6522 VIAs, einem Custom–Videochip und einem Soundchip.

$A000—$BFFF Hier liegen 512 KB oder 2 MB RAM in 8–KB–Bänken.

$C000—$FFFF Dort liegen 128 KB ROM in 16–KB–Bänken.

Ein wesentlicher Unterschied zu C64 und C16–64k: Weder I/O–Bereich noch ROM sind zugunsten von RAM ausblendbar; damit sind an zusammenhängendem Arbeitsspeicher tatsächlich nur knapp 40 KB RAM verfügbar, ab dem Basic–Start bei $0801 sogar nur knapp 38–KB. Beim *cc64* wurde das tatsächlich zum Problem, doch dazu später mehr.

Der erste Portierungsschritt war, den X16 als neues Target im Makefile anzulegen und die systemspezifischen Sourcen vom C64 zu kopieren. Als Nächstes mussten gut zwei Handvoll systemspezifische C64–Adressen durch die entsprechenden X16–Adressen ersetzt werden — dank der Kernal–Genealogie gab es für alle eine 1:1–Entsprechung. Viele sind dabei aus der *Zeropage* in den $0200er–Bereich gewandert; der X16 hält (erfreulicherweise) die halbe Zeropage für Anwenderprogramme frei, im Gegensatz zu den nur 8 freien Bytes beim C64.[3] Interessant war auch der Tastaturpuffer, der beim X16 im banked RAM bei $A000 liegt.

Dann mussten die Code–Worte zum Auslesen der Tastatur, zum An– und Abschalten des Cursors und die Systeminitialisierung angepasst werden. Die größte Schwierigkeit war dabei tatsächlich der Cursor, und halb damit verbunden, die Konsolen–Bildschirmausgabe. Ganz zufrieden bin ich mit dem aktuellen Stand nicht; es gibt noch einen Bug, der teilweise beim Backspace einen inversen Cursor stehen lässt. Aber bisher sehe ich nur undokumentierte Kernal–Einsprünge oder Duplizieren von Kernal–Code als mögliche Fixes; mal sehen, ob sich in der Diskussion mit der Community etwas ergibt.

Bemerkenswert fand ich, daß ich den systemspezifischen Code, den sich C64 und C16 teilen (den CBM–spezifischen Code sozusagen), bei der gesamten Portierung praktisch nicht anfassen musste. Dieser Teil ist offensichtlich sauber über die Kernal–API implementiert, die auch der X16 unterstützt. Sehr schön.

Der letzte Schritt bis zum ersten Release war dann, den Emulator *x16emu* in die Testscripts und ins Makefile einzubinden. Der Targetcompiler für alle (mittlerweile) 4 Varianten sowie die Tests für C64 und C16 laufen im Emulator *VICE*[4]. Es stellte sich heraus, dass x16emu ein SD–Card–Image zur vollen Laufwerksemulation benötigt, während VICE sehr bequem CBM–Laufwerke auf Host–Verzeichnisse mappen kann. Es waren also zusätzliche Kopierschritte in und aus SD–Card–Images nötig. Aber ansonsten funktionierten alle Ansätze zum Fernsteuern von VICE auch mit x16emu, und die Tests liefen tatsächlich unverändert, allerdings ohne die Block–Tests; Blockzugriff habe ich beim X16 bisher noch nicht implementiert, da ich fast ausschließlich mit Streamfiles arbeite.

Das Release auf der X16–Webseite wurde positiv aufgenommen, und aus der Diskussion entstand eine Zusammenarbeit mit dem Autor eines Editors, mit dem Ergebnis, daß dieser Editor inzwischen in der letzten freien ROM–Bank des X16 platziert und von VolksForth aus gerufen werden kann.

Portierung von cc64

Nachdem die Basis „VolksForth“ lief, wollte ich natürlich auch den darauf aufbauenden cc64 auf den X16 bringen.

[1] „The 8–Bit Guy“

[2] CBM–Sprech für *Kernel*, eigentlich eher BIOS.

[3] Die ersten 256 Byte im Speicher (entspricht einer Page), also der Bereich von 0 bis 255 bzw. $0000-$00FF, wird auch Zeropage (Page 0 bzw. Seite 0) genannt. Bei den Prozessoren der 6502–Familie kommt der Zeropage eine besondere Bedeutung zu, da zur Adressierung einer Speicherzelle nur ein Byte gebraucht wird (das Highbyte ist definitionsgemäß 0). Befehle mit Zeropage–Adressierung sind daher besonders kompakt und schnell. Außerdem müssen die Pointer für die indirekte Adressierung in der Zeropage liegen (ausgenommen beim indirekten JMP–Befehl). Somit kann die Zeropage als eine Art externe Registererweiterung angesehen werden.

[4] VICE ist ein freier plattform–unabhängiger Emulator für verschiedene 8–Bit–Computersysteme von Commodore.

cc64 ist ein Small–C–Compiler, den ich auf dem C64 in Forth geschrieben und vor kurzem auf den C16 portiert habe, und der 6502–Code erzeugt. Er ist auch das Projekt, das mich überhaupt zu *Forth* gebracht hat.

Da der Compiler selbst praktisch keinen C64– bzw. C16–spezifischen Code enthält, hätte die X16–Portierung einfach sein können. War sie aber nicht, und zwar wegen der bereits erwähnten Grenze von knapp 38 KB zusammenhängendem RAM. Zum Vergleich: Auf dem C64 stehen bis zu 50 KB und auf dem C16–64k sogar bis zu 59 KB zur Verfügung, wobei jeweils das ROM ausgeschaltet und das parallele RAM genutzt wird. Beim X16 dagegen ist bei $9F00 Schluss, da liegt der I/O–Bereich, der nicht abschaltbar ist. Die cc64–Binaries sind 32 KB lang. Dazu kommen die Header fast aller internen Worte, die mit `|` auf dem Heap[5] angelegt werden, und der transiente 6502–Assembler, der auch auf dem Heap lebt. Es erstaunte also wenig, als es beim ersten Bauversuch mitten im cc64–Parser hieß: „dictionary full“. Eine schnelle Messung zeigte, daß es selbst mit den 8 KB einer X16–RAM–Bank als zusätzlichem Heap nicht reichen würde. Was also tun?

Die „große“ Option wäre, einzelne Codestrecken, z. B. einzelne Module, in verschiedene RAM–Bänke zu kompilieren und mit Bank–Switching zu rufen. Das würde große Mengen RAM erschließen, wäre aber mit dem Aufwand verbunden, den Code in geeignete, voneinander möglichst unabhängige 8–KB–Pakete zu gruppieren, und falls Cross–Bank–Aufrufe nicht zu vermeiden wären, diese über einen Zwischensprung außerhalb der Bänke zu führen, oder evtl. die Next–Routine und ggf. den Returnstack um Bank–Handling zu erweitern. Eine interessante Option, mächtig, aber teuer, und vielleicht geht es ja auch noch anders.

Was, wenn ich Heap–Speicher mehrfach nutzen könnte? Viele der internen Worte werden nur innerhalb eines Moduls gerufen; wenn das Modul fertig geladen ist, werden nur noch die Header seines Interface gebraucht, der Rest könnte weg. Ich bräuchte also einen zweistufigen Heap, wo ich einen Teil immer wieder entsorgen und den Speicher wiederverwerten könnte, während der andere Teil des Heaps bis zum Ende des Ladens erhalten bleiben muss.

Damit war die Idee des `TmpHeap` geboren, den Carsten Strotmann zusammen mit dem grundlegenden Heap–Mechanismus für Header in seinem Artikel in diesem Heft beschreibt. Die größte Hürde war, die innere Mechanik des existierenden Heap–Codes im VolksForth zu verstehen; im Zuge dessen sind auch einige hoffentlich nützliche Kommentare in die VolksForth–Sourcen geflossen. Danach stellte sich die Implementierung als relativ einfach heraus, und es war auch einfach, beim X16 den `TmpHeap` in eine RAM–Bank zu legen, was 8 KB zusätzlichen Speicher beim Laden der cc64–Sourcen bedeutete.

Der letzte interessante Schritt war, die internen und externen Worte der Module von cc64 zu identifizieren und entsprechend zu markieren, und zu entscheiden, wann und wie oft der `TmpHeap` während des Baus geleert wird. Das bedeutete dann auch, sich anzuschauen, wie die verschiedenen Module voneinander abhängen und wie breit die Schnittstellen dazwischen sind. Normalerweise betrachte ich Abhängigkeiten eines Forth–Projektes mit einem mentalen Schicht–Modell: Jedes Modul wird auf die vorherigen draufgeladen, und die Reihenfolge der Schichten spielt keine große Rolle, solange nur jedes Modul oberhalb derer liegt, von denen es abhängt, egal ob es viele oder wenige Worte davon nutzt.

Aber plötzlich bot sich ein anderes Bild: Module gruppierten sich zu einem Abhängigkeitsgrafen, teils mit vielen Schnittstellenworten eng gekoppelt, teils mit wenigen eher lose, und es bot sich die Möglichkeit, Gruppen von eng gekoppelten Modulen zu definieren, und den `TmpHeap` nicht an Modul– sondern an Modulgruppengrenzen abzuräumen. Dadurch konnten die vielen Header der breiten Schnittstellen innerhalb einer Modulgruppe auch auf den `TmpHeap` wandern und brauchten nicht bis zum Ende des Baus auf dem Heap zu liegen — mehr wiederverwendbare Speicher. Andererseits — je kleiner der `TmpHeap` und je häufiger er geleert wird, desto höher die Mehrfachverwendung des dafür reservierten Speichers. Im Falle von cc64 stellte sich ein `TmpHeap` von 8 KB, eine volle Speicherbank des X16, als Sweetspot heraus, groß genug für alle internen Wortheader der größten Modulgruppe, bestehend aus Parser, Codegenerator und transientem 6502–Assembler, klein genug, um auf dem C64 noch auf den regulären Heap zu passen — und deutlich größer, als ich zunächst erwartet hätte. Experimentalinformatik halt. :-) Auf dem C16–64k mit seinem reichlichen Speicher von $1001—$FC00 kommt cc64 übrigens weiterhin ganz ohne `TmpHeap` aus.

Ich war froh, dass mit geeigneter Modulgruppenstruktur und 8–KB–TmpHeap der Bau des cc64 auf dem X16 durchlief. Es war nicht selbstverständlich, dass der `TmpHeap` genug Speicher sparen konnte, aber es reichte — knapp, wie sich später zeigte.

Die cc64—Laufzeitbibliothek benötigte nur geringfügige Anpassungen. Die make–Regeln und –Skripte um den x16emu herum mussten aus der VolksForth–Codebasis in die des cc64 übertragen und die make–Regeln zum Testen von einer 2x2–Matrix (cc64–Host– und –Target–Plattformen C64 und C16) auf eine 3x3–Matrix (C64, C16, X16) erweitert werden, was nochmal eines Makefile–Refactorings bedurfte, sonst wäre es arg unhandlich geworden. Von Seiten des cc64 liefen die Tests auf dem X16 auf Anhieb. Danach wurde noch die Dokumentation aktualisiert, ein kleines Tool zum Patchen des Zeichensatzes wurde für den X16 angepasst, und ein ärgerlicher Bug beim Anzeigen von Verzeichnissen wollte gefunden und gefixt werden, den ich zunächst auf den veränderten Kernal des X16 schob, der aber letztlich daher kam, dass ich während des cc64–Baus `SAVE` benutzt hatte, während noch die Protokollierung aller Ausgaben in eine Logdatei aktiv war.

[5] „Heap“ ist kein C–Style–Heap. Es ist eigentlich ein nach unten wachsendes temporäres Dictionary, so wie der Data–Stack, wohingegen das Dictionary selbst aufwärts wächst — Siehe Carsten Strotmanns Artikel in diesem Heft.

Danach stand dann dem Release des cc64 v0.8 auf dem X16 nichts mehr im Wege, und es wurde, wie schon das VolksForth–Release, positiv aufgenommen.

Wie knapp der verbleibende Speicher war bzw. ist, wurde bei den Arbeiten an den Versionen v0.9 und v0.10 deutlich, welche die schlimmsten Flaschenhälse bezüglich Übersetzungsgeschwindigkeit sowie ein paar Parser– und Codegen–Probleme beheben und ein kleines Feature hinzufügen. Nicht mehr als wenige hundert Bytes zusätzlicher Code, aber für den X16 schon wieder zu viel. Zwei weitere Tricks um den `TmpHeap` herum konnten mich noch einmal retten. Zum einen fiel mir auf, dass der Code für den `TmpHeap` selbst auf dem Heap leben kann, genau wie der Code des transienten 6502–Assemblers. So braucht der TmpHeap–Code nicht Teil der Turnkey–Anwendung zu sein, in diesem Fall des cc64. Und zum anderen wurde mir klar, dass es bei der letzten Modulgruppe vor dem `SAVE` funktional keinen Unterschied macht, ob ein Wortheader auf dem normalen `Heap` oder dem `TmpHeap` liegt — beide Heaps werden beim `SAVE` geleert. Und da in dieser Modulgruppe noch viel TmpHeap–Platz frei war, konnte ich dort pauschal sowohl interne als auch Interface–Header auf den `TmpHeap` legen und so etwas normalen `Heap` sparen, der sich den Platz ja mit dem Dictionary teilt.

Für den zusätzlichen Code der Version v0.10 reicht der so gesparte Platz noch, aber um weitere Feature–Arbeit am cc64–Compiler auch für den X16 zu ermöglichen, werde ich wohl an der weiter oben angedeuteten aufwendigeren Nutzung mehrerer Speicherbänke des X16 nicht herumkommen, so dass nach dem v0.10–Release mein Fokus vom cc64 wohl erstmal wieder auf VolksForth wechseln wird und ich mir Möglichkeiten zum Bank–Switching mit Forth überlegen muss.

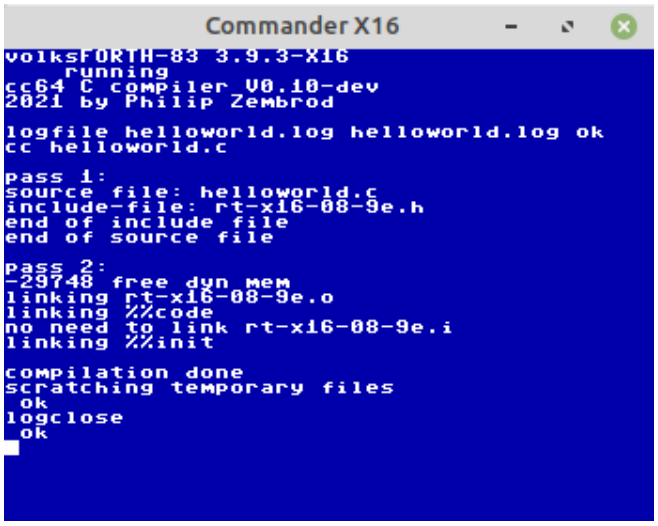


Abbildung 1: Screenshot — cc64 kompiliert helloworld.c

Links[6]

VolksForth

`https://github.com/forth-ev/VolksForth/tree/master/6502/C64`

`https://www.commanderx16.com/forum/index.php?/files/file/84-volksforth-x16/`

`https://www.commanderx16.com/forum/index.php?/topic/686-new-community-dev-tool-uploaded-volksforth-x16/`

X16Edit

Der erwähnte Editor, den VolksForth jetzt rufen kann:

`https://www.commanderx16.com/forum/index.php?/files/file/68-x16-edit-a-text-editor/`

cc64

`https://github.com/pzembrod/cc64`

`https://www.commanderx16.com/forum/index.php?/files/file/121-cc64-x16/`

`https://www.commanderx16.com/forum/index.php?/topic/935-new-community-dev-tool-uploaded-cc64-x16/`

Commander X16

`https://commanderx16.com/`

X16–DevTools

Liste mit cc64, VolksForth und anderen Entwicklungswerkzeugen:

`https://www.commanderx16.com/forum/index.php?/files/category/9-dev-tools/&sortby=file_downloads&sortdirection=desc`

Eventuell auch noch nützlich

Mein Talk bei der SVFIG über den `heap` und den `tmpheap` und den Einsatz für *cc64*:

`https://www.youtube.com/watch?v=MmL3J-pc6Vk`

Anmerkung

Ich habe (noch) kein Board. Es existieren, glaube ich, auch noch keine, außer den ersten 3 Prototypen. So ist Forth sogar auf dem X16 schon da, wenn das neue Board kommt. Ich entwickle übrigens unter Linux Mint.

[6] In der PDF–Ausgabe des Heftes könnt ihr die Links anklicken. `https://wiki.forth-ev.de/doku.php/vd-archiv`

Forth–Gruppen regional

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

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

München **Bernd Paysan**
Tel.: (0 89) – 41 15 46 53
`bernd@net2o.de`
Treffen: Jeden 4. Donnerstag im Monat um 19:00 in der Pizzeria La Capannina, Weitlstr. 142, 80995 München (Feldmochinger Anger).

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

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

Dienste der Forth–Gesellschaft

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

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

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

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

Spezielle Fachgebiete

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

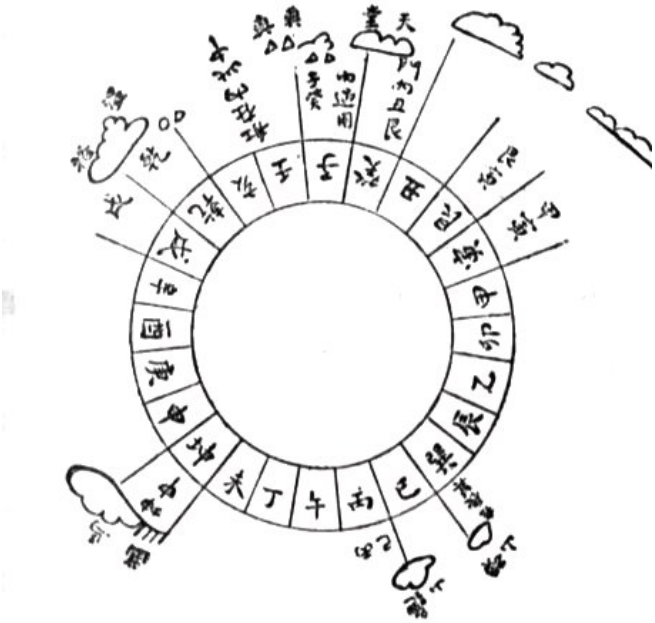
Termine

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

Montags ab 20:30 Uhr
Forth lernen
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–Konferenz „Rom“

10. – 12. September 2021 — war online

Du warst nicht dabei?

> „on air . . . these session are streamed live on twitch, recorded and shared on youtube for future generations to enjoy and to rewatch.“
>
> (Quelle: `https://euro.theforth.net/`)

Neben der Möglichkeit, sich in den *chatrooms* zu verabreden, gab es Vorträge von (in dieser Reihenfolge):

- Ulrich Hoffmann: Taming the IoT — Forth's Role in the Internet of Things.
- Nick Nelson: A simple Forth wrapper for a Linux logging library.
- M. Anton Ertl: Practical Considerations in a Static Stack Checker.
- Philip Zembrod: Where does my app spend its time? A small Forth profiler.
- Leon Wagner: Forth Satellite Antenna Control.
- Klaus Schleisiek: microCore progress and The Linguistics of Forth.
- M. Anton Ertl: Moving Bytes.
- Gerald Wodni: fput: standardized sockets and directories.
- Bill Ragsdale: A Two–dimensional Data Strucure for Forth.
- Bob Armstrong: CoSy Becomes usable by Ordinary People.
- null: Virtual Excursion: Non Forth Hobbies.
- Peter Knaggs: Using Test Driven Development to build a new Forth interpreter.
- Bernd Paysan: net2o update.
- Krishna Myneni: Simulation of Einstein–Podolsky–Rosen (EPR) Experiment in Forth.
- Andrii Pylypets: new to Forth. null: Lightning Talks.
- Brad Rodriguez: The case for `<BUILDS`.

Abends zum „Dinner“, dann jeweils der *Open Chat*.

Schnelles Internet, Firefox–Browser und Headset sowie eine gute Beleuchtung waren Voraussetzung für die Teilnahme. Die web–basierte Chat–Applikation *Mattermost* hat funktioniert. Und die Video–Konferenz selbst lief über *BigBlueButton*, aufgesetzt auf eigenem Server von Gerald Wodni.

Im Vorfeld, 7. – 9. September, war wieder das Forth–Standard–Meeting.

Links

Proceedings: `http://www.euroforth.org/ef21/papers/`

Videos: `https://wiki.forth-ev.de/doku.php/events:ef2021:start`

CfP: `http://www.euroforth.org/ef21/cfp.html`

Jahrestagung der Forth–Gesellschaft und Mitgliederversammlung

6. – 7. November 2021 — wird online sein

Im Heft 4d2021–01 hatten wir unsere Mitglieder dazu aufgerufen, ihre Meinung mitzuteilen, in welcher Form nun das Jahrestreffen gewünscht wird. Mitglieder, welche an der Abstimmung teilgenommen haben, wünschten sich mehrheitlich eine *Online–Tagung mit Mitgliederversammlung*. Die Auszählung hatten wir im Heft 4d2021–02 an dieser Stelle publiziert.

Das Direktorium hat inzwischen die Online–Tagung und MV vorbereitet. Details zur Wahl des Direktoriums müssen noch geklärt werden.

Das Direktorium hat neben Vorträgen auch Räume für Diskussionen und lockere Gespräche vorbereitet. Wenn ihr Ideen und Vorschläge habt, aber auch Anträge zur Tagesordnung der Mitgliederversammlung, dann sendet diese bitte *nun* an `direktorium@forth-ev.de` oder per Brief an die Büroadresse der Forth–Gesellschaft.

Wir freuen uns auf rege Rückmeldungen.

Gleich anmelden!

Das Anmeldeformular zur Tagung wird *ab dem 01.10.2021* unter `https://tagung.forth-ev.de` online sein, zusammen mit näheren Informationen zur erforderlichen Technik und den Abläufen.

Euer Direktorium

Ulrich Hoffmann, Bernd Paysan, Carsten Strotmann