Vierte Dimension

3/2023

Das Forth–Magazin

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

In dieser Ausgabe:

Titelbild Himbeeren und andere Früchtchen AUTOR mk

Quelle: Kollage aus Schnipseln aus dem Internet

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 1030
48481 Neuenkirchen
E–Mail: Secretary@forth-ev.de
Direktorium@forth-ev.de
Bankverbindung: Postbank Hamburg
BLZ 200 100 20
Kto 563 211 208
IBAN: DE60 2001 0020 0563 2112 08
BIC: PBNKDEFF

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00 € + Porto u. Verpackung

Liebe Leser,

die Himbeeren im Garten waren gut diesen Sommer, sie hatten reichlich Sonne. Und in der digitalen Welt gab es auch reichlich *Raspberry* für mich. Ein lustiger Name für so Platinchen. Früher war man technischer eingestellt bei der Namensgebung, *Motorola* wurde von den Gründern des Unternehmens, den Brüdern Paul V. und Joseph E. Galvin, im Jahr 1930 eingeführt, eine Kombination aus „Motor“ (für Motorfahrzeuge) und „ola“ (für Ton), was auf die Produktion von Autoradios hinweisen sollte. Die neue Open–Hardware–Szene kommt zwar auch wieder technischer daher, mit einem schlichten Zahnrad als Logo, aber ihr neues Betriebssystem heißt *Quark*, wie die fundamentalen Bestandteile der Materie. Und die sollen ja Geschmacksrichtungen haben, fast wie Himbeeren. Was genau sich der amerikanische Physiker Murray Gell–Mann 1963 dabei gedacht hat, die so zu taufen, weiß er selbst nicht mehr. Er habe es aus der Zeile „Drei Quarks für Muster Mark!“ in James Joyces Roman *Finnegans Wake* (Seite 383, in der Faber–und–Faber–Ausgabe des Buches von 1939) entnommen — behauptet ChatGPT. Ich ahne schon, *Quark* wird so bunt wie Forth.

Willem Ouverkerk zeigt, dass in *noForth T* die Anbindung der externen Bausteine mittels I²C geschafft ist. Ebenso Rafael Deliano, die Themopiles werden per I²C bedient. Rafael hat noch zwei weitere Beiträge beigesteuert. Der *springende Ball* zeigt physikalisches Zusammentreffen, digital und analog simuliert. Bauteil–Einsicht zu erzeugen, gefällt mir persönlich natürlich sehr, ich hoffe, euch auch; und dass das mit Forth so fein geht. Forth auf jede Werkbank!

Den Bastelspaß, einen Forth–Computer nur aus TTL–Chips zu machen, hat Matthias Koch sogleich ausprobiert und mit seinem *Mecrisp* Analyse–Tools dafür gemacht. Und Wolfgang Strauss nutzte die Gelegenheit der Sommerbekanntschaft zu Dennis Kuschel, ihn zu interviewen. Unsere Sommertreffen im Linuxhotel sind immer gut für die Kontaktpflege! Retrocomputing ist faszinierend, finde ich.

Und *Neo–Retrocomputing* ebenso. Ulrich Hoffmann führt uns in dieses Thema ein am Beispiel eines Open–Source–8–Bit–Mikrocomputers des 21. Jahrhunderts, auf einem Board in Open–Source–Hardware.

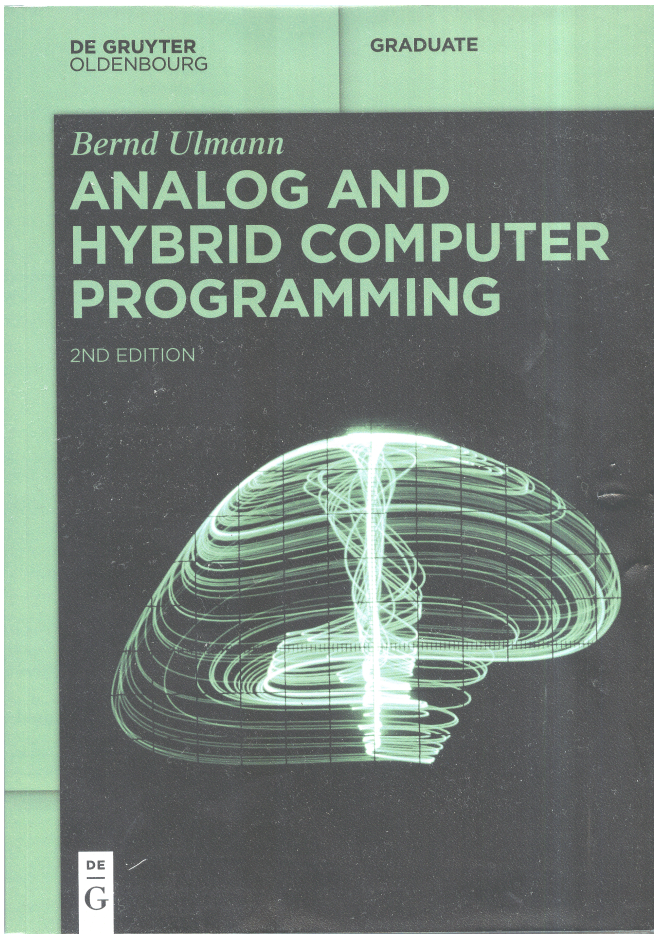
Wir bleiben verspielt. François Laagel löst Hexadokus mit Forth. Die besondere Kunst dabei ist das Debuggen des rekursiven Forthcodes. Sein Werkzeug dafür ist ein „cryptographic message digest algorithm“, zu Deutsch: „kryptografischer Nachrichten–Hash–Algorithmus“. Damit wird den Daten eine feste Zeichenfolge zugeordnet, ihr *Hash–Wert* oder *Digest*, welcher dann eine eindeutige Darstellung der ursprünglichen Nachricht ist, womit sich die Integrität der Daten feststellen lässt, sodass Fehler während der Bearbeitung des Stacks erkannt werden können.

Ist es nicht schön, zu sehen, was sich durch die Forth–Treffen Fantastisches ergeben hat?

Man sieht sich. Online auf der nächsten Tagung im Frühjahr, beim Sommertreffen leibhaftig und bei der EuroForth sowieso.

Euer Michael.

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
http://fossil.forth-ev.de/vd-2023-03



Analogrechner — Buchbesprechung

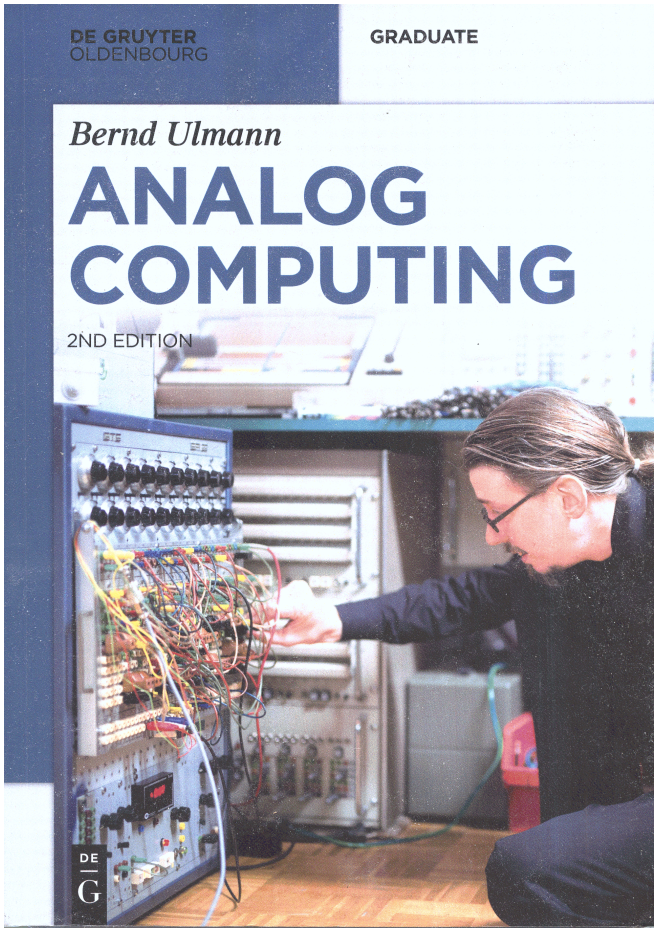
Neulich sind zwei Bücher erschienen, die für Forther sehr interessant sein könnten.

- Ulmann, Bernd; „Analog Computing“, 2. Aufl. De Gruyter Oldenbourg; 2022 (ca. 72 EUR)
- Ulmann, Bernd; „Analog and Hybrid Computer Programming“ 2. Aufl. De Gruyter Oldenbourg; 2023 (ca. 55 EUR)

Rafael Deliano hat sich beide genauer angesehen und schrieb dazu das Folgende.

> „Die allgemeine Retrowelle hat die Antiquitäten erreicht und der üppige Preis die Leser nicht abgeschreckt. Ein Nachteil ist, dass sich beide Bände inhaltlich deutlich überschneiden.
>
> Das Erfolgsrezept des Autors: Erstens, Illustrationen sind gut in den Text gemischt; d. h., Fotos der Hardware, Blockschaltbilder oder Auszüge von Schaltungen. Zweitens ist es ein fachkundiger Text der Themen, aber kurz abgehandelt. Nur sporadisch Formeln, lieber Fotos vom Testlauf einer Anwendung. Mit umfangreichem Quellenverzeichnis für den Leser, der sich in ein Thema tiefer einarbeiten will.
>
> Die Einschätzung des Autors über die rosige Zukunft neuer Analogrechner muss man nicht teilen. Rein wegen Unterhaltungswert kauft aber niemand Fachbücher. Es ist wohl eher so, dass die ‚offizielle Wahrheit‘, d. h., dass mehr Megabyte, mehr Gigahertz und mehr Komplexität in die Zukunft führen, nicht mehr geglaubt wird.
>
> Der Operationsverstärker als Standardbauteil der analogen Schaltungstechnik war ein Spinoff des Analogrechners. Die plumpe Annahme der 70er Jahre, dass ihn der digitale Signalprozessor (DSP) in den 80er Jahren verdrängen wird, ist nicht eingetreten. Der sequentielle DSP war zu langsam und Parallelrechner wurden nicht marktfähig. Die steigende Integrationsdichte bei fallenden Bauteilpreisen rechnete sich nur für Speicher, Standard–CPUs und einige ICs für Consumer–Anwendungen. Mit dem Nebeneffekt einer Marktkonzentration auf immer weniger Hersteller.
>
> Für alle, die nicht Massenware fertigen, ist dieser Trend unbefriedigend. Der Analogrechner kann dann als Ansatz erscheinen, wie man sich von der Ebene der Blockschaltbilder zurück zur linearen Schaltung bewegt.
>
> Diese traditionell minimalistische Lösung für ein Problem wird in vielen Fällen vom Entwicklungsaufwand und den Materialkosten her gesehen der markfähigere Ansatz bei kleinen Stückzahlen sein.“

Und so sind im vorliegendem Heft auch gleich zwei Beiträge von Rafael Deliano erschienen, die diesen seinen Ansatz zeigen. mk



HP–41C, The Forth Edition

Auf *Hackaday* schrieb Al Williams neulich[1] über die Auferstehung des Forth auf dem HP–41C Taschenrechner: „… arguably the best calculator ever made.“

> „The original code dates back to 1984, but some recent detective work by [Angel Margin] has the code running again. If you know about synthetic programming on the 41C and the oddities of its internal architecture, you can't help but be impressed … We wonder if the code would run on an emulated 41C?“ (Retrocomputing, Tagged forth, hp41c)

Worauf leaf antwortete: „See RpnCalc (Edward Falk) on Google Play.“

Garth kommentierte:

> „… HP–IL, along with the interface converters to IEEE–488, RS–232, parallel, and video (all of which I have), allowed these little machines to be interfaced to dozens of things at the same time. This is the whole reason I got into the 41 — and it's why I cannot get interested in a 42, 48, 50, or DM–anything. Most people's idea of I/O is very anemic, thinking only of things like mass storage and printer. I used the 41 and 71 for interfacing to lab instrumentation. My web page about how I got into the 41 and all the stuff I have for it is at `http://wilsonminesco.com/HP41intro.html`. I'm a strong advocate for Forth, and I use it almost exclusively on the workbench. It's the way my brain works …“

Die Retroszene lebt. mk

Advent of Computing

> „Welcome to Advent of Computing, the show that talks about the shocking, intriguing, and all too often relevant history of computing. A lot of little things we take for granted today have rich stories behind their creation, in each episode we will learn how older tech has lead to our modern world.“

Sean Haas schreibt dort seit 2019 über die Computergeschichte. Von Episode 1 (Lochstreifen) bis Episode 114 (LPG–30) reicht sein Blog inzwischen. Und die Episode 89 behandelt Forth:

> „What language has two stacks? What language is used on satellites and in home computers? What language deals in words? Why, Forth, of course! Forth is a highly unique language developed in the 60s by Chuck Moore. And when I say unique, I mean unique. Forth uses reverse polish notation for all operations, along with a dedicated data stack for passing parameters. But it's not just unique for the fun of it, Forth's design is highly deliberate. It offers a level of simplicity and power that's rarely seen in programming languages.“

Auch die anderen Episoden sind kurz und knackig. Angenehme Lektüre! mk

`https://adventofcomputing.libsyn.com/`

Natives Forth für Agon Light

Lennart Benschop hat ein Forth–System für den Agon–Computer entwickelt und auf GitHub unter dem Namen *agon–forth* veröffentlicht. Dieses Forth–System bietet zwei Hauptvarianten: *forth16* und *forth24*. Während forth16 eine 16–Bit–Version von Forth für den Agon darstellt, die im Z80–Modus läuft, ist forth24 eine 24–Bit–Version, die im eZ80–ADL–Modus ausgeführt wird und den gesamten verfügbaren RAM nutzen kann. Beide Versionen bieten derzeit denselben Funktionsumfang, wobei forth24 weiterentwickelt wird. Zusätzlich zu diesen Hauptversionen enthält das Repository eine Reihe von Beispielprogrammen. Einige dieser Beispiele, wie `tetris.4th`, sind allgemeine Forth–Programme, während andere, wie `serpent.4th` und `graphics.4th`, speziell für den Agon entwickelt wurden. Es ist bemerkenswert, dass `serpent.4th` ein Snake–ähnliches Spiel ist, das ursprünglich für den Olimex WPC im Juni 2023 eingereicht wurde. Lennart Benschop arbeitet aktiv an der Weiterentwicklung und Verbesserung dieses Forth–Systems für den Agon. Ulrich Hoffmann

`https://github.com/lennart-benschop/agon-forth`

ratfactor — eine Reise ins Forth

Dave Gauer schrieb: „I have finally finished the web page *‚Forth: The programming language that writes itself.‘* Which is now practically a small book about the evolution and meaning of the Forth programming language.“

Danke für die unterhaltsame Geschichte über „Charles H. Moore and the pursuit of simplicity.“ mk

`https://ratfactor.com/`

[1] (October 21, 2022)

Technische Temperaturmessung — Buchbesprechung

Dr. Ing. Frank Bernhard von der Technischen Universität Ilmenau hat 2004 ein umfassendes Werk zur *technischen Temperaturmessung* herausgegeben — *1465 Seiten* stark. Die 2. Auflage erschien 2014. Nun, fast 10 Jahre später, gibt es noch nichts Vergleichbares, soweit ich weiß. Wer in die Tiefe des Themas will, sollte da nachlesen. Ich hab die erste Auflage noch da, sprecht mich an.

Bernhard hatte den Anspruch, den aktuellen Entwicklungsstand der elektrischen, mechanischen, akustischen und optischen Messprinzipien und Sensoren, der entsprechenden Messgeräte und ihrer Einsatzbedingungen sowohl der berührenden als auch der Strahlungs–Temperaturmesstechnik, zu erfassen. Mit diesem Buch soll allen in Industrie und Labor tätigen Physikern und Chemikern, vor allem den Entwicklungsingenieuren sowie den Meterologen und den Studenten entsprechender Fachrichtungen, die Möglichkeit gegeben werden, sich in das Gebiet der technischen Temperaturmessung einzuarbeiten. Dazu ist das Buch übersichtlich gegliedert und mit zahlreichen Literaturhinweisen versehen. Es geht auf die theoretischen Grundlagen der Temperaturmesstechnik ein und behandelt die möglichen Fehlerquellen und ihre Beurteilung bei Messungen von Temperaturen, was im Zeitalter der hochauflösenden digitalen Messgeräte sowie der rechnergestützten Messwertverarbeitung als besonders wichtig erscheint.

Und er beginnt sein Werk geschichtsbewusst, zitiert im Vorwort einen wichtigen Vordenker:

> „Die Thermometer sind ohne Widerrede eine der hübschesten Erfndungen der modernen Physik, welche zugleich am meisten zu deren Fortschritten beigetragen haben. Sie haben uns eine große Anzahl interessanter Kenntnisse vermittelt, die ohne ihre Hilfe nicht erreichbar erschienen. In wie vielen Fällen hätten wir ohne Thermometer erfahren können, daß miteinander zusammengemischte Flüssigkeiten sich erwärmen?
>
> Wir wüßten ebensowenig, daß siedendes Wasser eine Temperatur hat, über welche hinaus das Wasser nicht erwärmt werden kann. Alle Physiker wissen, daß zahllose Experimente mit dem Thermometer in der Hand angestellt werden müssen. Und nicht sie allein bedürfen dieses Instrumentes; es ist nicht auf ihre Laboratorien beschränkt geblieben. Man liebt es sehr, das Thermometer zu beobachten, um die Temperaturen der Luft zu erfahren, namentlich, wenn die Kälte und die Wärme unbequem werden, benutzt man das Instrument.
>
> Weiß man einerseits, wie amüsant und nützlich dieses Instrument ist, so kennt man andererseits seine Unvollkommenheit. Der Gang fast aller Thermometer ist verschieden. So versteht man schließlich nur das Thermometer, daß man mehrere Jahre lang selbst verfolgt hat, jedes andere bleibt unverständlich.“ (Rene–Antoine Ferchault de Reaumur, *im Jahre 1730*; [0–1])

Wie wahr doch sein letzter Satz ist!

Rene–Antoine Ferchault de Reaumur war ein französischer Natur– und Materialforscher mit einem weiten Interessen– und Arbeitsgebiet (wie fast alle Forth–Enthusiasten). :) So beschäftigte er sich unter anderem mit der Entstehung der Schalen der Schalentiere, der Temperaturmessung (Réaumur–Skala) sowie auch mit der Herstellung von Stahl, Glas und Papier. Große Beiträge leistete er besonders zur Entomologie (Insektenkunde). Er wird als einer der Wegbereiter der modernen Thermodynamik angesehen. [Wikipedia] mk

Was ist Wärme?

> „Wärme ist eine Form der Energieübertragung zwischen Substanzen mit unterschiedlichen Temperaturen. Sie tritt aufgrund der kinetischen Energie der Atome und Moleküle in einem Material auf. Wenn ein Material wärmer ist als ein anderes, haben seine Teilchen eine höhere durchschnittliche Bewegungsenergie.
>
> Wärme kann auf verschiedene Arten übertragen werden:
>
> Wärmeleitung: Diese tritt auf, wenn benachbarte Teilchen Energie durch direkten Kontakt weitergeben. Metalle sind oft gute Wärmeleiter.

> Konvektion: Dies ist der Prozess, bei dem wärmere Fluide (Flüssigkeiten oder Gase) aufsteigen und durch kühlere Fluide ersetzt werden, wodurch ein ständiger Wärmetransport entsteht. Das ist der Grund, warum warme Luft aufsteigt.
>
> Strahlung: Hierbei handelt es sich um die Übertragung von Wärmeenergie durch elektromagnetische Wellen. Die Sonne überträgt Wärme auf die Erde durch Strahlung, auch Infrarotstrahlung genannt.
>
> Die Einheit für Wärme ist *Joule* (J), jedoch wird oft auch die Einheit *Kalorie* (cal) verwendet, wobei 1 kcal ungefähr 4,18 J entspricht.
>
> Wärme ist eine wichtige Energieform, die in zahlreichen Aspekten des täglichen Lebens eine Rolle spielt, von der Temperaturregelung in Gebäuden bis zur Energieerzeugung in Kraftwerken.“ [Quelle: Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT]

Ist das nicht nett? So hat man doch im Handumdrehen die noch fehlende halbe Seite Text im Heft herbeigezaubert. mk

Fehler

> „Begeht man Fehler, weil man etwas versucht, das man bis zu diesem Fehler noch nicht konnte, dann sind das durchaus ehrbare Fehler. Das Gegenteil besteht in Fehlervermeidungsstrategien. Sich nichts zu trauen, weil man nichts falsch machen will, ist ein schwerer Fehler. Damit entschuldige ich meine schlechten Sachen nicht, ich halte schlechte Werke für durch nichts gerechtfertigt. Sie sind nur unumgänglich.“

Schrieb Bernd Zeller, Satiriker, in „Die Technik von Komik und Satire, Lehrbuch zum professionellen Gebrauch in Journalismus und Kreativem Schreiben“ neulich. mk

Temperatur, thermisch

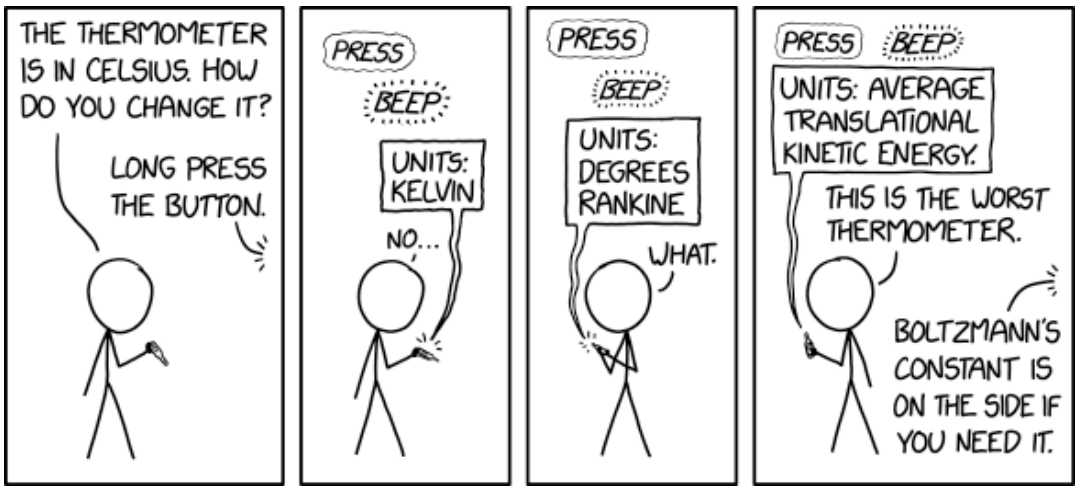
Da wir ja in Forth Worte benutzen, hat es mich nicht in Ruhe gelassen, auch diesem Wort nachzuspüren.

Herkunft: Ableitung vom gebundenen Lexem *therm–*, Entlehnung von altgriechisch *thermos* (Deutsch: warm, heiß, siedend, glühend).

Forscht man in der englischsprachigen Ethymologie, wirds spannender.

> **gwher–*
>
> Proto–Indo–European root meaning „to heat, warm.“
>
> It forms all or part of: brand; brandish; brandy; brimstone; brindled; forceps; Fornax; fornicate; fornication; fornix; furnace; hypothermia; thermal; thermo–; Thermopylae; Thermos.
>
> It is the hypothetical source of/evidence for its existence is provided by: Sanskrit *gharmah* „heat;“ Old Persian *Garmapada–*, name of the fourth month, corresponding to June/July, from *garma-* „heat;“ Hittite *war–* „to burn;“ Armenian *jerm* „warm;“ Greek *thermos* „warm;“ Latin *formus* „warm,“ *fornax* „oven;“ Old Irish *fogeir* „heated;“ Old English *bærnan* „to kindle.“ … *temperature* (n.) mid–15c., „fact of being tempered, proper proportion;“ 1530s, „character or nature of a substance,“ from Latin *temperatura* „a tempering, moderation,“ from *temperatus*, past participle of temperare „to mix in due proportion, modify, blend; restrain oneself“ (see *temper* (v.)). Sense of „degree of heat or cold“ first recorded 1670 (Boyle), from Latin *temperatura*, used in this sense by Galileo. Meaning „fever, high temperature“ is attested from 1898. `https://www.etymonline.com/search?q=Temperature`

Hättest du gedacht, dass Tempera–Farben–Malerei und schottischer Whiskey sprachlich so eng verwandt sind? mk



Quelle: `https://xkcd.com`

I²C Master & Slave on RP2040

Willem Ouwerkerk

The RP2040 has two hardware I²C modules on board. To explore the internals of the chip I did build a simple but solid I²C master and I²C slave example.

For the master I used the factorisation as designed for Project Forth Works, this means that all P.F.W. drivers can be used with this code. The code was written for noForth T, but with minor modifications it should run on any RP2040 Forth system.

The I²C Master

In Table 1 you see the most important P.F.W. I²C master primitives:

`I2C-ON`	Activate the I²C driver
`{I2C-WRITE`	Open I²C to write +n bytes
`{I2C-READ`	Open I²C to read +n bytes
`I2C}`	Close I²C read or write operation
`BUS!`	Send byte via the I²C bus
`BUS@`	Read byte via I²C bus
`DEVICE!`	Set I²C device address
`{DEVICE-OK?}`	Leave true when a device is in use and accessible

Table 1: I²C master primitives

To activate the I²C hardware as master we have to do this:

- Enable I²C on the chosen GPIO pins
- Activate those inputs with pull–up resistors and 4 mA output drivers
- Choose the I²C hardware unit (`I2C0` in this example)
- Disable the I²C hardware
- Set the high and low clock duration (200 kHz in this example)
- Set the spike suppression (100 ns in this example)
- Initialise a 7–bit master, fast speed & slave off
- Finally enable the I²C hardware again

```
hex  0 value ‘I2C            \ Hold I2C base register
: I2C-ON        ( -- )
    03 0C gpio! 03 0D gpio!  \ I2C0 on GPIO12 & GPIO13
    4A 0C pads! 4A 0D pads!  \ Set GPIO12=SDA & GPIO13=SCL with pull up
    40044000 to ‘i2c         \ Use I2C0 register set ( I2C0_BASE )
    1      ‘i2c 6C + **bic   \ Disable I2C
    dm 240 ‘i2c 1C + !       \ Set high & low clock period (~200kHz)
    dm 294 ‘i2c 20 + !
    dm 12  ‘i2c A0 + !       \ Spike suppressing 100ns (50ns for high speed)
    0065   ‘i2c  !           \ 7-bit master, fast speed, restart & slave off
    1      ‘i2c 6C + **bis ; \ Enable I2C
```

Using the I²C Status Registers

The example below shows the result of a primitive I²C code runtime test, while observing the two I²C status registers after a write action is done. More experiments like this were done to find out all the tests for the master code.

ICIS I²C Interrupt Statusregister (offset 0x2C)

ICS I²C Statusregister (offset 0x70)

```
@)blink     ( Two write actions to a PCF8574 chip )

ICIS-ICS    ( 1 )
 10  27     - 27 = I2C bus active
 10  27
 10   6     -  6 = I2C bus in rest ( ready )
 10   6
ICIS-ICS    ( 2 )
 10  27     = 27 = I2C bus active
 10  27
 10   6     =  6 = I2C bus in rest ( ready )
 10   6
```

Resulting I²C Master Code

Applying these test conditions, we need only 3 of the 42 I²C registers and end up with the code below.

$10 IC_DATA_CMD — Data buffer and command register

$2C IC_INTR_STAT — Interrupt status register

$70 IC_STATUS — Status Register

```
40044000 value ‘I2C     \ I2C0_BASE     I2C register pointer
0        value SUM      \ Count of bytes to transmit or receive
: BUS?          ( -- )
    10 us  ‘i2c 70 +  @ 2 = ?abort ; \ Abort on not connected bus
: DEVICE!       ( dev -- )
    1 ‘i2c 6C + **bic           \ Disable I2C
    7F and 400 or  ‘i2c cell+ ! \ Set TARget address & start condition
    1 ‘i2c 6C + **bis ;         \ Enable I2C
: DATA!         ( +n -- )       \ Send data +n
    -1 +to sum  sum 0= 200 and  \ Decr. byte count, last add stop
    or  ‘i2c 10 + ! ;           \ condition to last byte & send
: {I2C-WRITE    ( +n -- )
    to sum  begin  ‘i2c 70 + @ 6 = until ; \ Bus free? (6)
: {I2C-READ     ( +n -- )       {i2c-write ;
: I2C}          ( -- ) ; immediate  \ Dummy I2C ending
: BUS!          ( b -- )
    FF and  data!                   \ Send data byte b
    begin   bus?  ‘i2c 70 + @       \ Read status register (6/27)
            6  sum if  21 +  then   \ Bus ready- or busy-status
    = until ;                       \ Ok
: BUS@          ( -- b )
    100 data!                       \ Send dummy byte
    begin   ‘i2c 2C + @ 50 = ?abort \ Abort on invalid read (50)
            bus?  ‘i2c 70 + @       \ Read status register (E/2F)
            0E  sum if  21 +  then  \ Bus ready or busy
    = until                         \ Wait until data is received
    ‘i2c 10 + @  FF and ;           \ Read returned data b
```

We use a fourth I²C register to reset the write hardware when an I²C device was not available. A read on that register is enough to reset the write error.

$54 IC_CLR_TX_ABRT — Clear TX abort flag by reading

```
: {DEVICE-OK?}  ( -- f )        \ Leave true when device matches
    1 {i2c-read  100 data!      \ Start dummy read data with stop condition
    begin
        ‘i2c 2C + @  dup 14 =   \ Device present & ready (14)?
        swap 50 =               \ Device not present or busy (50)?
    or until                    \ Wait for response
    true  ‘i2c dup 2C + @ 40 and \ Device not present (40)?
```

```
    if    >r  0=  r>  44 +       \ Yes, change flag & correct address
    then  10 + @ drop ;          \ Dummy read on data or abort register
```

An I²C master example, on I²C address $20, a port with 8 LEDs. On I²C address $21, a port with 8 switches, `INPUT` reads those switches, `>LEDS` activates the LEDs and `BLINK` flashes the LEDs once.

```
\ PCF8574 example
: >PCF8574   ( b dev -- )  device!  1 {i2c-write  bus!} ;
: PCF8574>   ( dev -- b )  device!  1 {i2c-read   bus@} ;

i2c-on
: >LEDS      ( b -- )      invert 21 >pcf8574 ; \ Write byte to PCF8574
: INPUT      ( -- b )      20 pcf8574>  FF xor ; \ Read byte from PCF8574
: BLINK      ( -- )        true >leds 100 ms  false >leds 100 ms ;
```

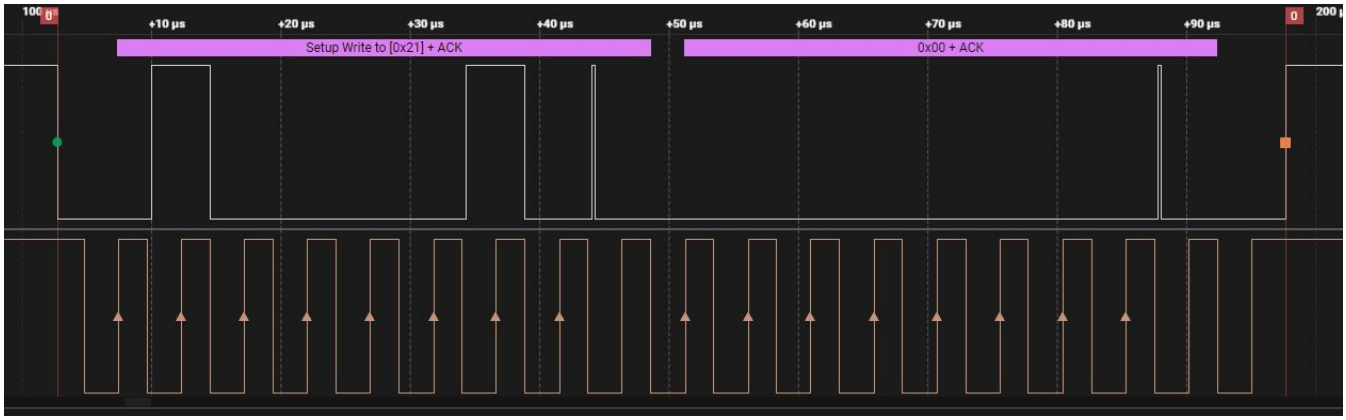


Figure 1: A PCF8574 write action

The I²C Slave

Table 2 shows the discovered I²C slave primitives until now:

`DEVICE-ON`	Activate the I²C slave driver
`RESET-FLAGS`	Clear I²C read request & TX abort flags
`I2C-READ?`	Is there an I²C request for reading?
`I2C-WRITE?`	Is there an I²C request for writing?
`I2C-DATA`	Leave I²C data register

Table 2: I²C slave primitives

To activate the I²C slave hardware we have to do a few things:

- Enable I²C on the chosen GPIO pins
- Activate inputs with pull–up resistors and 4 mA output drivers
- Choose the I²C hardware unit (`I2C1` in this example)
- Disable the I²C hardware
- Set own slave address
- Initialise 7–bit address, fast speed & slave on
- Finally enable the I²C hardware again

```
hex  0 value 'I2C            \ Hold I2C base register
: I2C-ON        ( mode my-addr -- )
    03 0E gpio! 03 0F gpio!  \ I2C1 on GPIO14 & GPIO15
    4A 0E pads! 4A 0F pads!  \ Set GPIO14=SDA & GPIO15=SCL with pull up
    40048000 to 'i2c         \ Use I2C0 register set ( I2C0_BASE )
    1      'i2c 6C + **bic   \ Disable I2C
```

```
          'i2c 8 +  !       \ Set SAR (slave) address
          'i2c  !           \ Set I2C mode for this device
    1     'i2c 6C + **bis ; \ Enable I2C
```

I²C Slave Code

Only a few primitives have to be defined!

```
: RESET-FLAGS   ( -- )
    'i2c 50 +               \ Clear interrupt registers base
    @+ drop  @ drop ;       \ Clear read request & TX abort

: I2C-READ?     ( -- 0|x )  20 'i2c 2C +  bit** ; \ I2C read request?
: I2C-WRITE?    ( -- 0|x )   8 'i2c 70 +  bit** ; \ I2C write request?
: I2C-DATA      ( -- a )    'i2c 10 + ;           \ I2C data register
```

Two I²C Slave Examples

The first slave emulates a PCF8574 like device, it uses the bootkey on the Pico board as an input. The incoming data is printed in rows on the terminal screen.

```
: BOOTKEY?  ( -- f )
    2000 4001800C **bis  10 us  \ QSPI pin-SS is input (OEOVER bitfield)
    2 D0000008 bit** 0=         \ Read boot key on QSPI pin-SS
    3000 4001800C **bic ;       \ QSPI pin-SS peripheral function again

: IO-SLAVE      ( -- )          \ Send & receive I2C data on MY address
    84 30 device-on  ." on " cr \ I2C slave on addr. 30 active
    begin
        i2c-read? if                    \ Read request from me?
            bootkey? FF and  i2c-data ! \ Yes , send bootkey status
            reset-flags                 \ Clear interrupts
        then
        i2c-write? if                   \ Write request to me?
            i2c-data @  FF and  3 .r    \ Yes, read data & show it
        then
    key? until  65 30 device-on         \ I2C slave off
    cr ." Slave off " ;
```

This is the master code for addressing the slave:

```
: >PCF8574      ( b dev -- )    device!  1 {i2c-write  bus!} ;
: PCF8574>      ( dev -- b )    device!  1 {i2c-read   bus@} ;

: COUNTER       ( -- )      \ Send counter to I2C slave
    cr  i2c-on  0  begin    \ Start master
        30 pcf8574> if      \ Ask data from slave (bootkey)
            dup .  dup 30 >pcf8574  1+ \ Send & increase counter to slave
        else  ." ."  then  20 ms \ Print dot when bootkey was not pressed!
    key? until  drop ;      \ Ready
```

The second slave emulates a memory like device, it uses a byte–addressed buffer in RAM on the slave board for data storage. Data can be read and written to that RAM buffer.

```
hex create RAM  100 allot   \ Data buffer
0 value MEM                 \ Pointer

: MEM-SLAVE ( -- )                  \ Store & read I2C data on device address
    84 30 device-on  ." on " cr     \ I2C memory slave on address 30 active
    begin
        i2c-read? if                        \ Read request from me?
            mem c@ i2c-data !               \ Yes, read buffer & send data
```

```
            incr mem  reset-flags           \ Increase addr. & clear interrupts
        then
        i2c-write? if                        \ Write request to me?
            i2c-data @ FF and  ram + to mem \ Yes, set memory buffer address
            begin   i2c-write?
                    i2c-read?  or until      \ Read or write request received?
            i2c-write? if                    \ Is it another writing request to me?
                i2c-data @  mem c!           \ Yes, fetch data & store in buffer
            then
        then
     key? until  65 30 device-on             \ I2C slave off
    cr ." Slave off " ;
```

This is the master code for the memory slave, these are Forth like primitives that read & write the memory buffer over the I²C bus. The used addresses ‘ma’ are byte wide.

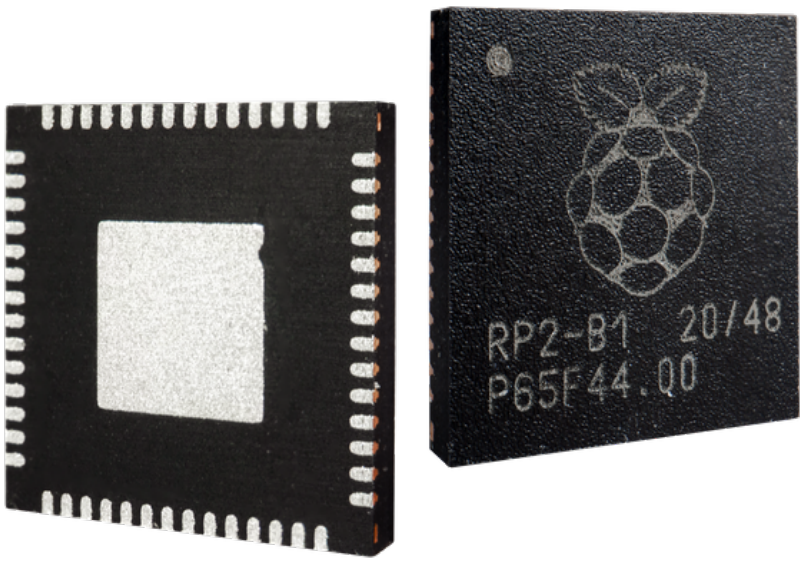
```
: {MADDR ( ma +n -- )    \ Address memory buffer
    30 device!  {i2c-write  bus! ;

\ Byte wide fetch and store in a memory buffer
: NMC@   ( -- b )        1 {i2c-read  bus@ i2c} ; \ Read next byte from memory
: MC@    ( ma -- b )     1 {maddr i2c}  nmc@ ;    \ Read byte from address
: MC!    ( b ma -- )     2 {maddr  bus! i2c} ;    \ Store byte at address
: MC@+   ( ma -- ma+ x ) dup 1+  swap mc@ ;       \ Buffer version of COUNT
: MFILL  ( ea u b -- )   rot rot for  2dup mc!  1+  next  2drop ;

: MDMP   ( ma -- )  \ Dump memory buffer from byte address ‘ma’
    hex  i2c-on  begin
        cr  dup 4 u.r ." : "
        dup   10 for  mc@+ 2 .r space  next  ch | emit  \ Show hex
        drop  10 for  mc@+ pchar emit  next             \ Show Ascii
    key bl <> until  drop ;
```

Resource & Early Access Link

The first official release of noForth T will be ready in early October 2023. In the meantime, you can download the package here: `https://www.dropbox.com/scl/fi/vy9nlcceqqpyheay1lve1/noForth-T-230919.zip?rlkey=d8b0fhv336x26e7jm52m4c7qo&dl=1`



Thermopile MLX90614

Rafael Deliano

Berührungslose DC–Temperatursensoren sind handelsüblich geworden. Deren Empfindlichkeit ist nicht so hoch wie die der pyroelektrischen[1] (AC)–Sensoren. Da man aber keinen mechanischen Chopper benötigt, sind Thermopiles deutlich einfacher anwendbar.

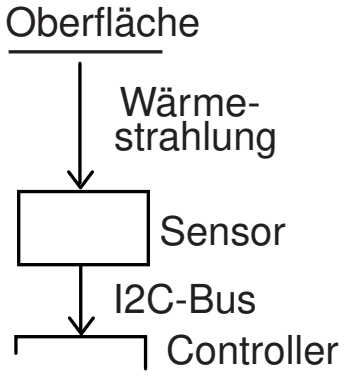


Abbildung 1: Blockschaltbild

Bei Thermopiles werden in Dünnfilmtechnik auf einem Chip eine große Zahl von Thermoelementen in Serie geschaltet (Abb. 2). Das erhöht die Spannung des Signals, aber auch den Innenwiderstand. Typischerweise befindet sich im IC ein weiterer Sensor für die Chiptemperatur.

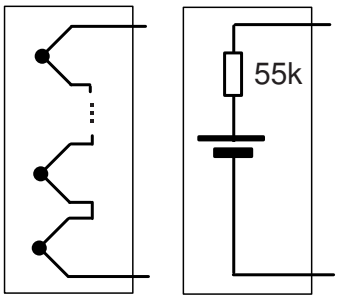


Abbildung 2: Thermopile und Ersatzschaltung

Im alten MLX90247 war nur der Sensor in der Dose. Der wurde dann ergänzt mit Baugruppen wie MLX90601[2], die auf der Leiterplatte noch ein IC zur Ansteuerung über SPI enthielt. Inzwischen sind beide Chips, Sensor und Bus–Interface, im Gehäuse montiert, die Schnittstelle ist nun ein I2C–Bus. Die Beschaltung wurde damit denkbar einfach (Abb. 3).

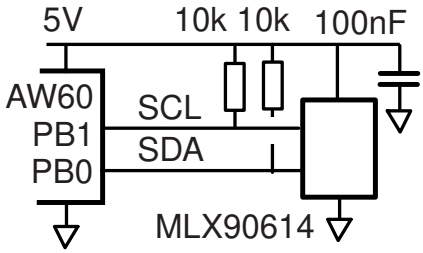


Abbildung 3: Schaltung

Für die Beschaffung bietet sich der Händler in Bulgarien auf *ebay.de* an. Der Hersteller fertigt dort, der Anbieter sitzt an der Quelle. Es gibt wahlweise Typen mit 5V– oder 3,3V–Versorgung. Nur Letztere haben den Sleep–Modus, somit eine Möglichkeit, die Eigenerwärmung zu reduzieren. Probleme sind neben Drift durch Aufheizen besonders das Rauschen. Saubere Versorgungsspannung hilft, das Schalten der digitalen Pins ist schädlich. Der Zugriffe auf das IC auf ein Minimum beschränken wäre sinnvoll, ist aber nicht immer praktikabel.

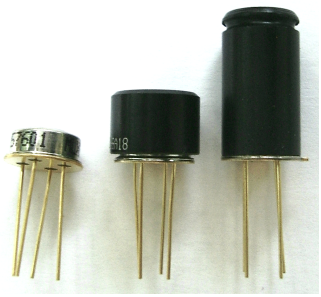


Abbildung 4: Gehäuseformen

Die nützlichen Bauformen haben oben schwarze Kappen, die als Blenden das Sichtfeld auf 10° bzw. 35° begrenzen (Abb. 4). Der Nachteil ist, dass die schwer zu reinigen sind. Es empfiehlt sich, bei Lagerung Kappen aufzustecken. Da man fürs Breadboarding Pins aus IC–Sockeln verwendet, in die man die Sensoren steckt, entfällt das Risiko der Überhitzung beim Einlöten.

SMBus

Gegenüber SPI benötigt I²C weniger Pins. Für den Sensor im Transistorgehäuse ein entscheidender Vorteil. I²C hängt sich aber bei Funktionsstörung gerne mal auf. Der bidirektionale SDA–Pin ist anfällig für Störungen durch die Kabelkapazität. Dies, und Scherereien mit der von Philips geforderten Lizenzierung führten dazu, dass es an kompatiblen Varianten mit anderem Namen nie mangelte.

Der von Intel 1995 initiierte *SMBus* ist eine davon [1] [2] [3]. Durch Einführung von Timeouts und Begrenzung der Taktrate wurde die Funktionssicherheit verbessert. Der Takt an SCL hat nun eine Ober– und eine Untergrenze (Abb. 5). Auf dem AW60 mit 4,19 MHz erwies sich eine Mischung aus Assembler und Forth als ausreichend schnell. Interrupts während des Zugriffs muss man aber sperren! Die Prüfung mit dem Oszilloskop ist empfehlenswert.

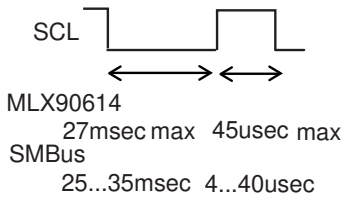


Abbildung 5: Timing

Ein weiterer Pferdefuß ist das üppige Protokoll (Abb. 9), das eine CRC8–Prüfsumme mitschleppt.

Man kann mehrere ICs an einem Bus verdrahten, dann ist aber die Slave–Adresse 5Ah im EEPROM zu ändern.

[1] Pyroelektrizität; Eigenschaft einiger piezoelektrischer Halbleiterkristalle. Dabei führt eine Temperaturänderung delta–T zu einer messbaren Änderung der elektrischen Spannung. PIR–Sensoren reagieren nicht wie andere Temperatursensoren auf ein bestimmtes zeitlich konstantes Temperaturniveau, sondern nur auf die Veränderung der Temperatur. (Quelle: Wikipedia)

[2] IR thermometer modules, which perform signal conditioning, linearisation and ambient temperature compensation.

Es existiert deshalb auch eine global gültige Adresse 00h.

Das IC ist nach Initialisierung der Portpins nicht sofort betriebsbereit — entweder man muss Wartezeit einfügen oder explizit das Flagbit `INIT?` pollen.

Memory–Map

Je 32 Datenworte im EEPROM und RAM, beide mit je 16 Bit Wortbreite, sind nur spärlich belegt (Tab. 1, 2). Nur das EEPROM kann von außen geschrieben werden. Dafür sind explizites Löschen und Wartepausen nötig. Und die Änderung wird erst nach einem Power–Up wirksam. Für Breadboards wäre also eine schaltbare Versorgungsspannung angenehm. Man kann aber meist mit den Defaults leben.

Signalverarbeitung

Auf dem zweiten Chip ist neben Analogschaltung und A/D–Wandler auch ein Controller integriert, der die Filter realisiert (Abb. 6, 10). Die Abtastrate ist undefiniert, sie hängt von den Einstellungen ab; d. h., die Schleife wird durch längere FIR–Filter langsamer abgearbeitet.

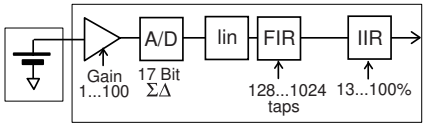


Abbildung 6: Signalfluss

Der Default für `Gain` ist `*100` und sollte für die meisten Anwendungen passen (Abb. 7). Nur bei sehr hohen Temperaturen muss man reduzieren. Der A/D–Wandler hat zwar üppige Dynamik, aber seine unteren Bits rauschen. Die Linearisierung biegt die Kennlinie des Thermoelements gerade und ist fix. Sie wird vor den Filtern ausgeführt (Abb. 6). Die Kalibrierung hat der Hersteller per EEPROM vorgenommen.

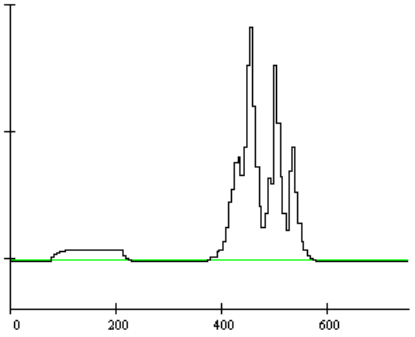


Abbildung 7: Testlauf, 5 cm Abstand: T_{amb} 28°C = 14900; Hand 37°C = 15300; Lötkolben = 25000.

Beim FIR–Filter, einem simplen Averager, kann die Anzahl der Taps gewählt werden (Tab. 3) [4]. Es soll Spikes eliminieren. Die Rechenleistung des Schrumpf–DSPs hat absehbar nicht für ein Median–Filter gereicht. In manchen Anwendungen (z. B. Fieberthermometer) ist die Ansprechzeit weniger wichtig als ein stabiler, rauschfreier Messwert. Die dafür nötigen langen Zeitkonstanten liefert das optionale 1–pol–IIR–Filter (Abb. 8). Sie sind nach Zeitkonstante für das Testsignal Sprung aufgelistet. (Tab. 4).

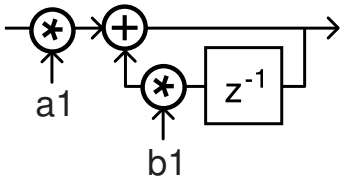


Abbildung 8: IIR–Filter

Anwendung

Der Traum des Herstellers sind Fieberthermometer und damit Consumer–Stückzahlen. Also schwaches Signal, bei dem man mit den Filtern das Rauschen reduzieren muss.

Prinzipbedingt liefert der Sensor keine °C direkt. Die ergeben sich über den Faktor *Emissivity*, der im EEPROM vorbelegt ist, dort von 0,1 … 1,0 einstellbar wäre [5], aber nur eine leidlich gut definierte Eigenschaft des Messobjekts ist. Das Verfahren bietet sich nicht für Präzisionsmessungen an.

Unproblematischer sind industrielle Anwendungen, wie z. B. Vergleichsmessungen an heißen Heizwendeln.

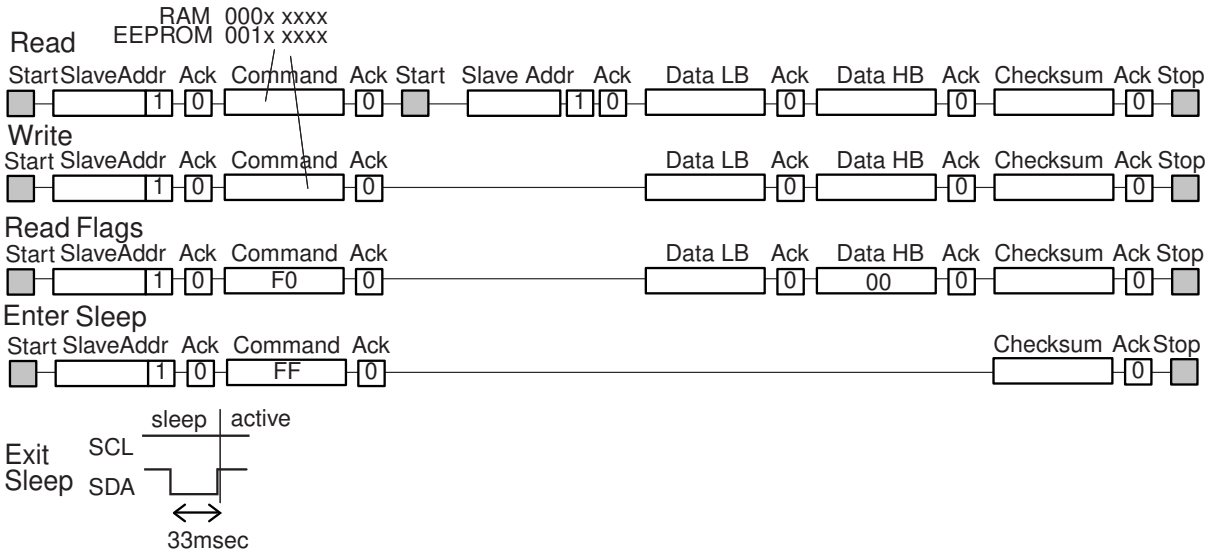


Abbildung 9: Protokoll

Tabellen

00	TO_{min}
01	TO_{max}
02	PWMCTRL (not used)
03	T_a range
04	Emissivity
05	Config 1
...	
0E	SMBus addr (= 5Ah)
...	
1C	ID (write prot.)
1D	ID
1E	ID
1F	ID

Tabelle 1: EEPROM

...	
04	raw data channel 1
05	raw data channel 2 (not available)
06	T_{amb}
07	T_{obj1}
08	T_{obj2} (not available)
...	

Tabelle 2: RAM

Taps	
128	
256	
512	
1024	default

Tabelle 3: FIR–Filter

	a1	b1	
100%	1,0	0,0	disable
80%	0,8	0,2	
67%	0,666	0,333	
57%	0,571	0,428	
50%	0,5	0,5	default
25%	0,25	0,25 0,75	
17%	0,1666	0,833	
13%	0,125	0,875	

Tabelle 4: IIR–Filter

Quellen

[1] Maxim „Comparing the I²C Bus to the SMBus“ AN476
[2] Analog Devices „Designing with SMBus 2.0“
[3] MLX90614 SMBus Communication Application Note
[4] MLX90614 On Chip Digital Signal Filters Application Note
[5] MLX90614 Changing Emissivity Unlocking Key Application Note

Und noch der Blick ins Innere

Entfernt man den Deckel der „Blechdose“, sieht man die Chips im Inneren (Abb. 10). Man erkennt an der Paste unter dem Thermopile, warum Löten heikel sein kann.

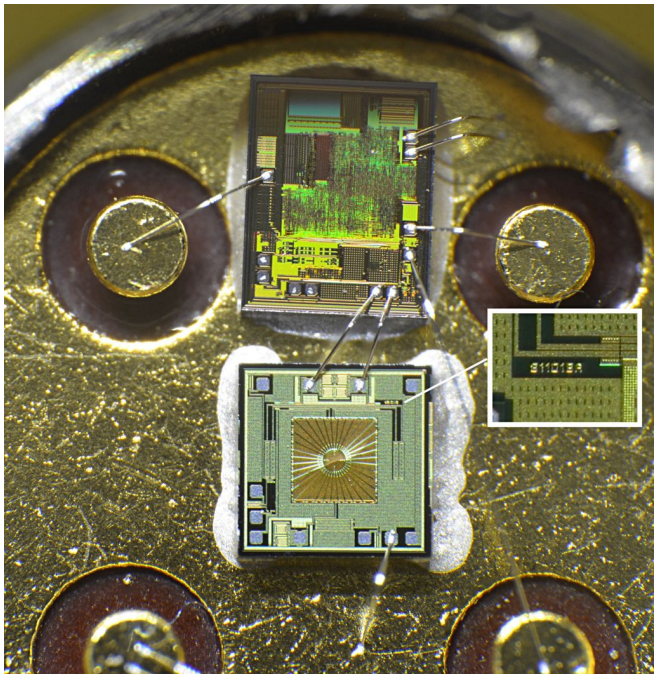


Abbildung 10: Abgedeckelte „Blechdose“ mit Thermopile MLX811018A in der Mitte und darüber dem Steuer–IC.

Listing[3]

```
1   <| HEX  \ MLX90614
2
3   PB 0 DCONSTANT SDA     \ in 0
4   DPB 0 DCONSTANT SDA-DIR
5   PB 1 DCONSTANT SCL     \ out 1
6
7   :CODE INIT       \ (  ---  )
8   PB  B% 00000010 #. MOV,
9   DPB B% 11111110 #. MOV, RTS, CODE;
10
11  :CODE SCL=0
12  SCL MBC, NOP, NOP, NOP, NOP, NOP, RTS, CODE;
13  :CODE SCL=1
14  SCL MBS, NOP, NOP, NOP, NOP, NOP, RTS, CODE;
15  :CODE SDA=0
16  SDA MBC, SDA-DIR  MBS, RTS, CODE;
17  :CODE SDA=1
18  SDA-DIR  MBC, NOP, NOP, NOP, NOP, NOP, RTS, CODE;
19  :CODE SDA?
20  SDA-DIR  MBC, NOP, NOP, NOP, NOP, NOP,
21  1 $ SDA  BBC, 1 $: RTS, CODE;
22
23  :CODE STOP         \ (  ---  )
24  ' SCL=0 JSR, ' SDA=0 JSR,
25  ' SCL=1 JSR, ' SDA=1 JMP, CODE;
26
27  :CODE START        \ (  ---  )
28  ' SDA=1 JSR, ' SCL=1 JSR,
29  ' SDA=0 JSR, ' SCL=0 JMP, CODE;
30
31  :CODE TC! \ ( UC1 --- Flag ) Flag =1 Error
32            \ in/out SCL=0, SDA=0
33                 1 ,X LDA,
```

[3] nanoFORTH GP32. Für die Non–Standard–Words `kann das` Handbuch (PDF) bei Bedarf vom Autor angefordert werden.

```
34  XSAVE                STX,
35                 08 #. LDX,
36  1 $:                A. ASL,
37  2 $                    BCC,
38        ' SDA=1          JSR,
39  2 $: ' SCL=1          JSR,
40        ' SCL=0          JSR,
41        ' SDA=0          JSR,
42                         DEX,
43  1 $                    BNE,
44        ' SDA=1          JSR,
45        ' SCL=1          JSR,
46                      A. CLR,
47        ' SDA?           JSR,
48  3 $                    BCC,
49                      A. COM,
50  3 $: ' SCL=0          JSR,
51        ' SDA=0          JSR,
52  XSAVE                LDX,
53                  1 ,X STA,
54                         RTS,
55  CODE;
56
57  :CODE TC@ \ ( Flag --- UC1 ) Flag=1 with ACK
58            \                  Flag = 0 no ACK
59            \ in/out SCL = 0, SDA = 0
60                  1 ,X LDA,
61  XSAVE                STX,
62                         PHA, \ flag to stack
63        ' SDA=1          JSR,
64                 08 #. LDX,
65  1 $: ' SCL=1          JSR,
66        ' SDA?           JSR, \ bit to carry
67                      A. ROL,
68        ' SCL=0          JSR,
69                         DEX,
70  1 $                    BNE,
71                   N     STA,
72                         PLA, \ flag from stack
73                      A. TST,
74  3 $                    BEQ,
75        ' SDA=0          JSR,
76  3 $: ' SCL=1          JSR,
77        ' SCL=0          JSR,
78        ' SDA=0          JSR,
79  XSAVE                LDX,
80                   N     LDA,
81                   1 ,X STA,
82                         RTS,
83  CODE;
84
85  2 ZVARIABLE CRC%
86
87  : CRC      \ ( CRC --- CRC' )
88  7 0 DO DUP 8000 AND
89  IF 1<SHIFT 0700 XOR ELSE 1<SHIFT THEN
90  LOOP ;
91
92  : CRCi    \ ( UC1 --- UC1 ) init
93  DUP 8<SHIFT CRC% ! ;
94
95  : CRC+     \ ( UC2 --- UC2 ) add
96  DUP CRC% @ OR CRC CRC% ! ;
97
98  : CRCe    \ (       --- CRC ) end
99  CRC% @ CRC 8SHIFT> ;
100
101 \ B4 CRCi DROP
102 \ FF CRC+ DROP
103 \    CRCE       -> E8
104
105 5A 1<SHIFT CONSTANT SA \ default slave addr
106 \ 0 CONSTANT SA        \ ROM slave address
107
108 : TR@ \ ( adr --- UN1 ) adr=00...1F read RAM
109 START SA 00 OR CRCi TC! 99 ?ERROR
110                CRC+ TC! 99 ?ERROR
111 START SA 01 OR CRC+ TC! 99 ?ERROR
112                   1 TC@ CRC+
113                   1 TC@ CRC+ 8<SHIFT OR
114                   1 TC@ CRCe = 88 /?ERROR
115 STOP ;
116
117 : TEE@ \ ( adr --- UN1 ) read EEPROM
118 START SA 00 OR CRCi TC! 99 ?ERROR
119          20 OR CRC+ TC! 99 ?ERROR
120 START SA 01 OR CRC+ TC! 99 ?ERROR
121                   1 TC@ CRC+
122                   1 TC@ CRC+ 8<SHIFT OR
123                   1 TC@ CRCe = 88 /?ERROR
124 STOP ;
125
126 : FLAGS@    \ ( --- UN1 ) read Flags
127 START SA 00 OR CRCi TC! 99 ?ERROR
128             F0 CRC+ TC! 99 ?ERROR
129                   1 TC@ CRC+
130                   1 TC@ CRC+ 8<SHIFT OR
131                   1 TC@ CRCe = 88 /?ERROR
132 STOP ;
133
134 : EEBUSY?   FLAGS@ 80 AND ; \ normally 0
135 : EE_DEAD?  FLAGS@ 20 AND ; \ normally 0
136 : INIT?     FLAGS@ 10 AND ; \ set when ready
137
138 : =SLEEP  \ (  ---  ) switch off ; only 3,3V
139 START SA 00 OR CRCi TC! 99 ?ERROR
140             FF CRC+ TC! 99 ?ERROR
141                CRCe TC! 99 ?ERROR
142 STOP ;
143
144 : =ACTIVE  \ (  ---  ) switch on ; only 3,3V
145 STOP SDA=0 D% 33 MSEC SDA=1 ;
146
147 : CR" 7 AND LNOT IF CR THEN ; \ ( UC1 ---  )
148
149 : TDUMP     \ (  --- ) dump memory
150 CR ." RAM"
151 1F 0 DO I CR" I CH. I TR@  NH. LOOP
152 CR ." EEPROM"
153 1F 0 DO I CR" I CH. I TEE@ NH. LOOP CR ;
154
155 \ init 50 msec tdump
156
157 : (TEE!)  \ ( UN1 adr --- )
158           \ adr = 00...1F  write EEPROM
159 START SA 00 OR CRCi TC! 99 ?ERROR
160 B% 00100000 OR CRC+ TC! 99 ?ERROR
161     DUP FF AND CRC+ TC! 99 ?ERROR
162        8SHIFT> CRC+ TC! 99 ?ERROR
163                CRCe TC! DROP
164 STOP ;
165
166 : TEE!    \ ( UN1 adr --- ) adr = 00...1F
167 0 OVER (TEE!) D% 10 MSEC     \ erase
168        (TEE!) D% 10 MSEC ;   \ write
169
170 |>
171
```

Ein Forth–Computer ganz ohne Prozessor: My4TH — ausprobiert und disassembliert

Matthias Koch

Wer exotische Architekturen zu schätzen weiß und dazu noch Forth in allen Geschmacksrichtungen mag, dem sei ein Bastelspaß der ganz besonderen Art empfohlen: Dennis Kuschel *aus Bremen hat den My4TH entwickelt, einen Computer, der nur aus Speicher und 16 TTL–Chips zusammengesetzt ist.*

Für mich war es eine schöne Überraschung, auf einmal einen Bausatz dafür in meinem Briefkasten zu entdecken. Vielen Dank an Carsten Fulde vom *Computermuseum Visselhövede*[1], der meine Vorlieben ganz genau richtig eingeschätzt hat!

Das Zusammenlöten der gut durchdachten Platine gestaltete sich problemlos. Da ich Sockel für sämtliche Chips eingebaut hatte, konnte ich das Resultat auch gut mit einem Durchgangsprüfer auf versehentliche Lötbrücken hin untersuchen. Das frisch gelötete Kunstwerk erwachte sogleich blinkend zum Leben, als ich, wie in der Bauanleitung [1] beschrieben, eine Drahtbrücke und eine Leuchtdiode hineinsteckte und es anschließend über ein USB–Kabel mit 5 Volt versorgte, um einen eingebauten Selbsttest zu aktivieren.

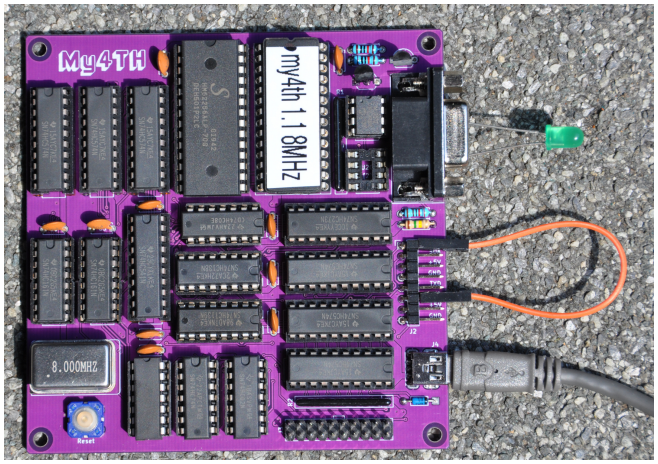


Abbildung 1: My4TH — gesockelt aufgebaut

Höchste Zeit, den My4TH ans Terminal anzustöpseln und mit schneckenschnellen 4800 Baud ins retrofuturistische Forth–Abenteuer einzutauchen! Das geht sowohl an einem echten seriellen Port als auch über eine Pinleiste mit einem USB–Seriell–Kabel, welches jedoch nicht unbedingt zur Stromversorgung herangezogen werden sollte, da der My4TH doch ein bisschen mehr Strom benötigt, als einer meiner USB–Seriell–Wandler vertragen hat.

Ein Druck auf den Reset–Taster, und schon werden die Mutigen an der Tastatur begrüßt:

```
EEPROMs: 24LC512,none (64 blocks)
Welcome to My4TH v1.1 / 8 MHz
```

[1] https://computermuseum-visselhoevede.de/

Das Forth ist standardkonform und bringt sogar einen kleinen eigenen Editor mit, womit unvergessliche Programme im I²C–EEPROM abgelegt werden können. Der Inhalt meiner ersten beiden Blöcke, die mit `0 edit` und `1 edit` angezeigt werden, ist als *Screendump* am Ende des Artikels wiedergegeben.

Wenn in der ersten Zeile von `Block 0` eine Blocknummer steht, so wird jener Block, hier `1`, automatisch beim Start geladen und sorgt für eine frische Begrüßungsmeldung:

```
EEPROMs: 24LC512,none (64 blocks)
Press any key to stop autostart
Now executing screen 1
ok
ok
ok
Guten Tag lieber Computer!
Ich habe Dich am 4. Juni 2023 zusammengebaut.
ok
```

Wer möchte, kann eigene Erweiterungen an die Platine anschließen und über die noch freien Bits im Eingaberegister und im Ausgaberegister ansteuern. Die Leitungen `IN0` und `OUT0` sind für das Terminal zuständig, an `IN1` und `OUT1 (SDA)` sowie `OUT2 (SCL)` ist das I²C–EEPROM angeschlossen, alle anderen Leitungen sind frei für eigene Ideen. Dafür stehen die Wörter `WOUT`, `ROUT` und `RINP` zur Verfügung, und auch weitere I²C–Bausteine werden von Haus aus unterstützt. Einfach im Glossar nachschlagen! Noch allerdings habe ich keine Erweiterung für die Platine gebaut und überlege noch, welche Aufgabe ich diesem ganz besonderen Computer geben möchte.

Bis hierhin fühlt sich das Forth ganz normal an, aber bei einem Blick hinter die Kulissen wird es richtig spannend.

Da die arithmetische Einheit nur *1 Bit breit* ist und aus einem einzigen NOR–Gatter besteht, ist das Platinchen eher von der gemächlichen Sorte. Um die Suche im Dictionary ein wenig zu beschleunigen, gibt es gleich acht verlinkte Ketten durch das Dictionary, wobei die untersten drei Bits der Länge des Namens entscheiden, welcher der acht Fäden durchsucht wird. Pfiffig.

Aber wie ist das Forth aufgebaut? Was hält die kleine Welt im Innersten zusammen? Um das stilvoll aus Forth heraus zu ergründen, habe ich einen Disassembler für den My4TH geschrieben [2]. Für das Hochladen

von Quelltextdateien gibt es übrigens ein ganz praktisches Linux–Werkzeug gleich vom Schöpfer, welches das Handshake übernimmt und nach dem Laden ein Terminal präsentiert:

```
my4th exec /dev/ttyUSB0 file insight.fs -t
```

Eins meiner Lieblingsbeispiele beim Erkunden von neuen Compilern ist die Umwandlung von Binärzahlen zum Gray–Code:

```
: gray ( u -- x ) dup 1 rshift xor ; ok
see gray
8F01 : 1E PHL
8F02 : 19 JSR 5982 --> dup
8F03 : 82
8F04 : 59
8F05 : 19 JSR 44EC --> push 1
8F06 : EC
8F07 : 44
8F08 : 19 JSR 700B --> rshift
8F09 : 0B
8F0A : 70
8F0B : 19 JSR 5A0A --> xor
8F0C : 0A
8F0D : 5A
8F0E : 1F RTS
ok
```

Es handelt sich um ein Forth mit *Subroutine–Threaded Code* (STC). Schauen wir mal in die aufgerufenen Primitiven:

```
see dup
5982 : 1E PHL
5983 : 19 JSR 450E
5984 : 0E
5985 : 45
5986 : 19 JSR 44DB
5987 : DB
5988 : 44
5989 : 18 JMP 44D7
598A : D7
598B : 44
ok
see rshift
700B : 1E PHL
700C : 19 JSR 4559
700D : 59
700E : 45
700F : 15 TST r22
7010 : 16
7011 : 16 JPF 44D7
7012 : D7
7013 : 44
7014 : 0C ROR r21
7015 : 15
7016 : 0C ROR r20
7017 : 14
7018 : 0A DEC r22
7019 : 16
701A : 18 JMP 700F
701B : 0F
701C : 70
ok
see xor
5A0A : 1E PHL
5A0B : 19 JSR 453A
5A0C : 3A
5A0D : 45
5A0E : 04 LDA r20
5A0F : 14
5A10 : 0F XOR r22
5A11 : 16
5A12 : 06 STA r20
5A13 : 14
5A14 : 04 LDA r21
5A15 : 15
5A16 : 0F XOR r23
5A17 : 17
5A18 : 06 STA r21
5A19 : 15
5A1A : 18 JMP 44D7
5A1B : D7
5A1C : 44
ok
```

Die Instruktionen der Akkumulatormaschine sind wie beim altbekannten 6502 ein Byte lang, und können um eine Konstante, eine Adresse oder ein Register in der Zeropage — die ab $8000 im RAM–Speicher liegt — erweitert werden. Hier ein kleiner Auszug aus dem Disassembler mit sämtlichen Instruktionen:

```
00 of ." RST" endof
01 of ." LD " disasm-reg disasm-imm endof
02 of ." LD " disasm-reg disasm-reg endof
03 of ." LDA" disasm-imm endof
04 of ." LDA" disasm-reg endof
05 of ." LAP" disasm-reg endof
06 of ." STA" disasm-reg endof
07 of ." SAP" disasm-reg endof
\ ---
09 of ." INC" disasm-reg endof
0A of ." DEC" disasm-reg endof
0B of ." ROL" disasm-reg endof
0C of ." ROR" disasm-reg endof
0D of ." AND" disasm-reg endof
0E of ." OR " disasm-reg endof
0F of ." XOR" disasm-reg endof
10 of ." ADD" disasm-reg endof
11 of ." SUB" disasm-reg endof
\ ---
13 of ." CMP" disasm-imm endof
14 of ." CMP" disasm-reg endof
15 of ." TST" disasm-reg endof
16 of ." JPF" disasm-addr-cont endof
17 of ." JNF" disasm-addr-cont endof
18 of ." JMP" disasm-addr-name endof
19 of ." JSR" disasm-addr-name endof
1A of ." RET" endof
1B of ." PSH" disasm-reg endof
```

```
1C of ." POP" disasm-reg endof
1D of ." IN" endof
1E of ." PHL" endof
1F of ." RTS" endof
20 of ." JLP" disasm-addr-cont endof
21 of ." RWL" disasm-reg endof
22 of ." SEC" endof
23 of ." CLC" endof
24 of ." AD " disasm-reg endof
25 of ." SU " disasm-reg endof
26 of ." OUT" endof
\
```

Der Trick in der Implementierung dieser Instruktionen auf der spartanischen Logik besteht darin, dass die Instruktions–Bytes einfach nur der High–Teil einer Microcode–Adresse im EPROM sind. Der Microcode von `ROR` (Instruktion `$0C`, ROtate Right) ist also im EPROM in dem Bereich von `$0C00` bis `$0CFF` zu finden. Dieser „8–Bit–Prozessor“ ist im Prinzip eine virtuelle Maschine, die auf der 1–Bit–TTL–Logik läuft. Mit einem anderen Microcode wären also auch ganz andere Befehle möglich ... Leider kann die Logik keinen Microcode im RAM ausführen, was für Experimente so richtig klasse wäre. Es gibt allerdings einen Simulator, in dem Microcode entwickelt werden kann, ohne ganz viele EPROMs brennen und geknickte Pins riskieren zu müssen. Mein erster Gedanke war: Wäre es vielleicht möglich, wie bei klassischen Stackmaschinen gleich die nötigen Forth–Primitiven im Microcode unterzubringen? Doch ein tiefer Blick in die Implementierung des Microcodes zeigt deutlich: Hut ab! Mit Dennis Kuschel war ein Genie am Werk.

PS: Wer sich übrigens für diskret aufgebaute Computer ganz ohne Microcode–Tricks begeistert, dem möchte ich noch einen Blick auf den *Q2* empfehlen [3].

Links

[1] `http://mynor.org/my4th.htm`
[2] `https://github.com/Mecrisp/my4th`
[3] `https://joewing.net/projects/q2/`

```
-----------------------------[Screen 000]-----------------------------
01| 001                                                               |
02|                                                                   |
03|                                                                   |
04|                                                                   |
...                                                                ...
10|                                                                   |
11|                                                                   |
12|                                                                   |
13|                                                                   |
14|                                                                   |
15|                                                                   |
16|                                                                   |
------------------------------------------------------------------------
Enter line number to edit, or enter:
S = save, Q = quit (lose changes), W = save and quit, C = clear screen
>
-----------------------------[Screen 001]-----------------------------
01| : Hallo ." Guten Tag lieber Computer!" cr ;                       |
02| : Kunstwerk ." Ich habe Dich am 4. Juni 2023 zusammengebaut." ;   |
03| : run Hallo Kunstwerk cr ;                                        |
04| run                                                               |
05|                                                                   |
06|                                                                   |
07|                                                                   |
08|                                                                   |
09|                                                                   |
10|                                                                   |
11|                                                                   |
12|                                                                   |
13|                                                                   |
14|                                                                   |
15|                                                                   |
16|                                                                   |
------------------------------------------------------------------------
Enter line number to edit, or enter:
S = save, Q = quit (lose changes), W = save and quit, C = clear screen
>
```

Interview mit Dennis Kuschel, dem Entwickler von My4TH

Wolfgang Strauß

Auf dem Sommertreffen unseres Vereins hatte ich das Vergnügen, Dennis Kuschel *kennenzulernen. Er ist der Entwickler einer erstaunlichen Serie von CPU–losen Computern. Seine neuesten Erfindungen, den My4TH Forth–Computer und das Forth Deck, konnten die Teilnehmer live in Augenschein nehmen und auch ausprobieren. Mich haben seine Projekte so in den Bann gezogen, dass ich mehr über die Geschichte dahinter erfahren wollte. Ich habe Dennis um ein Interview gebeten. Hier ist es.*

Das folgende Interview hat am 10.08.2023 stattgefunden. Es geht darin primär um das „Warum“ und nicht so sehr um das „Wie“. Wer an technischen Details und der genauen Funktionsweise der Geräte interessiert ist, dem seien die Links am Ende dieses Artikels empfohlen. Und natürlich der Artikel von Matthias Koch in diesem Heft.

W: Hallo Dennis.

D: Hallo Wolfgang.

W: Ich schlage vor, wir legen sofort los.

D: Ich bin bereit.

W: Wie bist du zur Elektronik gekommen? Ein Aha–Erlebnis in der Jugend und danach kommt man einfach nicht mehr davon los?

D: Bei mir war das anders. Es ist mir sozusagen in die Wiege gelegt worden. Schon als 4–jähriger habe ich angefangen, Häuser zu malen — komplett mit der elektrischen Ausstattung: Beleuchtung, Lichtschalter, Steckdosen, Verteilerkästen. Das hat irgendwann ein Nachbar gesehen und gesagt: „Das wird bestimmt mal ein Ingenieur, ein Elektroniker.“ Dann stand das einfach fest, das war gesetzt, und so ist es dann auch gekommen.

Als ich in der 1. Klasse war, hatte ich einen Klassenkameraden, der mir ein defektes Transistorradio zeigte. Das wollte ich unbedingt haben. Ich habe dann das Radio gegen Süßigkeiten eingetauscht. Zu Hause habe ich es auseinandergenommen, an den Schräubchen gestellt und irgendwann spielte es dann wieder. Später habe ich bei meinem Vater in der Grabbelkiste ein bisschen Zwillingsleitung gefunden, den Lautsprecher ausgebaut und über das Kabel wieder angeschlossen. Dann spielte das, auf der einen Seite des Wohnzimmers das Radio und auf der anderen Seite der Lautsprecher. Meine Eltern waren erstmal platt. Was macht der da? Woher kann der das?

Wir hatten in Bremen einen Elektronikladen (Völkner). In der 2. Klasse habe ich in der Bücherei „Was ist Was“ ausgeliehen über Radios. Aha, so kann man Radios bauen. Danach noch andere Bücher ausgeliehen über Bauteile. Dann bin ich meinem Vater auf die Nerven gegangen: „Ich brauche Transistoren, Widerstände, Leuchtdioden ...“ Mein Vater wusste überhaupt nicht, was das ist. Vater hat einen Kumpel gefragt, der hat Völkner empfohlen, und dann hat mein Vater in dem Laden tütenweise Bauteile aus der Restekiste gekauft. Ein paar Lüsterklemmen noch dazu; das Ganze habe ich dann zu Weihnachten bekommen. Da war ich 8 Jahre alt oder so.

W: Ich hätte jetzt gedacht, der typische Einstieg wäre ein Experimentierbaukasten mit Anleitungsheft gewesen. Aber du hast wirklich eine Stufe tiefer angefangen ...

D: Genau. Dann habe ich in der Bücherei geschaut und andere Elektronikbücher gefunden. Ein Buch mit einer Wechselblinkerschaltung. Die habe ich mit meinen Bauteilen mittels Lüsterklemmen zusammengeschraubt. Auf einmal blinkte es. Oh, funktioniert. Noch keine Ahnung gehabt von Widerstandswerten und Farbcodes. Immer geschaut, ist die LED zu hell, zu dunkel, ah, jetzt genau richtig.

Dann habe ich angefangen, Sachen auseinanderzunehmen. Mein Vater hat immer Elektroschrott aus der Firma mitgebracht. Defekte Telefonanlagen, Drucker ... Da hatte ich einen großen Vorrat an Bauteilen und habe mich dann so reingefuchst.

W: Zwischenfrage: Hast du immer mit Lüsterklemmen gearbeitet? Wann gab es einen Lötkolben?

D: Irgendwann hat mich mein Vater mit zu Völkner genommen. Da gab es dann eine Streifenrasterplatine und damit war dann auch der Lötkolben fällig. Da war ich aber 10, als das losging. Meine Kinder habe ich jetzt auch löten lassen, da waren sie 5.

W: Wann hast du deinen ersten Rechner bekommen?

D: Wir hatten in der Schule eine Computer–AG. Der Lehrer hat seinen Schülern auf einem C64 Logo beigebracht. Das fand ich spannend, aber ich hatte keinen Computer daheim. Meine Freunde hatten teilweise einen, ich nicht. Mit 15 gabs dann einen C64 zu Weihnachten. Erst liefen auf dem Rechner Spiele, dann habe ich selber in BASIC programmiert. BASIC war aber langsam. Aus der Zeitschrift „64er“ habe ich einen Maschinensprachemonitor abgetippt. Ein halbes Jahr später programmierte ich in Maschinensprache. Ich habe auch 2 Programme an eine Computerzeitschrift verkauft. Da war ich noch keine 16.

Das war die Zeit, wo ein Klassenkamerad auf mich zukam, ob ich ihm einen Lichtcomputer bauen könnte. Ich hatte einen Vorrat an defekten C64 und habe aus den

Chips einen Steuerrechner zusammengeschustert. Ich habe mir auch einen EPROM–Emulator gebaut als C64–Einsteckmodul. Damit konnte ich dann den Selbstbaucomputer laufen lassen, ohne für jede Änderung ein EPROM zu löschen und zu brennen. Es wurde ein autakes Gerät mit 3 Pulten. Da war ich 17 Jahre alt.

W: Du hast einen Computer auf Lochrasterplatinen zusammengebaut?

D: Erst auf Lochraster. Dann aber sehr schnell auch selber Platinen geätzt mit Eisen(III)–chlorid. So haben wir dann den Lichtcomputer zusammengebaut. Das war eine spannende Zeit.

W: Du hast also immer alles selbst aus Einzelteilen gebaut. Keine Bausätze, vorgefertigte Module oder ähnliches?

D: Genau. Ich hatte aber irgendwann auch einen Kosmos Elektronik–Kasten. Den habe ich richtig ausgelutscht. Dann war für mich klar: Ich muss studieren. Elektronik. An der Fachhochschule. Ich hätte auch zur Uni gehen können. Mein Vater sagte: „Machste halt 'ne Ausbildung zum Physikalisch–technischen Assistenten, kurzer Dienstweg.“ Also Fachhochschulreife, dann in die FH rein, in Bremen die Hochschule für Technik, das war easy und auch langweilig. Aber ich hatte nachher mein Diplom, das war das Wichtigste. Ich habe viel Freizeit gehabt. Meine Diplomarbeit ging über Netzwerke und die Anbindung an einen DSP von TI zur Videoübertragung. Mein Professor hat die Diplomarbeit durchgelesen und gesagt, er verstehe kein Wort, das würde seinen Horizont übersteigen. Da aber alles super zu funktionieren scheint, könne er mir guten Gewissens eine 1 geben.

W: Au weia!

D: Das war ne schöne Zeit. Also, ich bin halt low–level. Ich will immer wissen, wie es funktioniert. Deshalb kommt es für mich auch nicht in Frage, mir ein Smartphone zu kaufen. Das Ding ist mir einfach zu high–level. Ich brauche die Herausforderung. Einfach ist nichts für mich.

W: Ja, genau. Erst immer in den Keller gehen und dann von da aus die Dinge aufbauen.

D: Stimmt. Aber wo wir gerade bei meinem Studium waren. Der Grundstein für My4TH wurde ja in meinem Studium gelegt. Das ist ja aus meinem Spaßprojekt entstanden, der MyCPU. MyCPU hat ne eigene Website. Während meines Studiums war ich unterfordert und mir deshalb oft langweilig. Wir hatten gerade die Vorlesung über Digitaltechnik. Interessant. Ist ein nettes Hilfsmittel für mich, aber wie kann ich das anwenden? Ich will aus diskreten Bauteilen einen Taschenrechner bauen. Löten wollte ich nicht. Ich habe dann in C einen Prozessor–Simulator geschrieben und hatte ziemlich schnell einen Prozessor am Laufen mit einer Kommandozeile. Dann habe ich Platinen geätzt, um das Ganze mal aufzubauen. Dann ist das gewachsen. Mein bester Freund und Kommilitone sagte damals, dass ich MyCPU ins Internet stellen solle. Ich habe dann auf `https://www.mikrocontroller.net` eine Unterseite bekommen. So ging das los. Als ich dann berufstätig war, habe ich in meiner Freizeit immer weitergebaut.

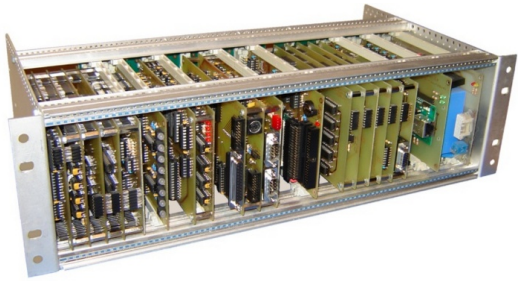


Abbildung 1: MyCPU, ganz links im Rack zu erahnen

W: Ok, das war also die MyCPU, geboren aus der Langeweile im Studium. Wie ging es dann weiter?

D: Vor ungefähr 4 Jahren hatte ich wieder Langeweile, bin in so ein Loch gefallen. MyCPU? Machste da noch ein bisschen weiter? Kannste. Aber eigentlich ist das doch kalter Kaffee, man müsste das ganze Ding mal wegschmeißen und komplett neu machen. Heutzutage würde ich das alles anders machen. Willst du das Ganze nochmal so aufziehen, so komplex? Nee, das dauert zu lange. Ich brauche was Einfaches, was auf eine Platine passt. Eines Abends war ich in einem seltsamen Rauschzustand, ich dachte, ich schwebe gleich, mir war so schwindelig. Ich habe dann in meinem Kopf den gesamten Schaltplan abgebildet, den ich mir so vorgestellt hatte und den auch durchsimuliert in meinem Kopf. Da war ich irgendwie wohl 2 Stunden in Trance. Ich habe dann angefangen, das ganze Ding aufzuschreiben und habe dann gemerkt: Mensch, das kann ja sogar funktionieren. Dann wieder eine Simulation geschrieben auf Gatterebene und es funktionierte. Wow! Das war dann der MyNOR Computer mit einem NOR–Gatter als zentralem Schaltelement. Wenn ich schon was Neues entwickele und es dann auch im Internet veröffentliche, dann muss es auch was Besonderes sein. Das Problem war nämlich, nach der MyCPU kamen viele Nachahmer, die haben ähnliche Projekte rausgebracht.

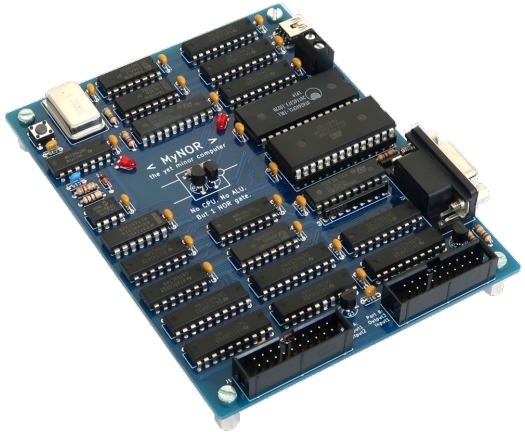


Abbildung 2: Der MyNOR. Man beachte das diskrete NOR–Gatter aus 2 Transistoren

W: Sag mal, warum hast du ein NOR–Gatter genommen und keinen NAND?

D: Ganz einfach. Für ein NOR brauche ich nur 2 Transistoren parallel und einen Pull–Up. Ein NAND wäre auch gegangen, ein NOR war für mich logischer.

So, dann habe ich überlegt: Wenn du im Internet gehört werden willst, musst du was Besonderes machen, ein Kunstprojekt. Eine schöne Platine, wo du das NOR–Gatter in Szene setzt.

W: Das NOR–Gatter ist auch bestimmt der Schaltungsteil, der die Geschwindigkeit vorgibt?

D: Nein, dachte ich auch erst. Das diskrete NOR–Gatter läuft mit bis zu 10 MHz ohne Probleme. Meine Vorgabe war, ich wollte nur Bauteile nehmen, die noch produziert werden. Deshalb konnte ich auch die 74181–ALU nicht nehmen. Außerdem liebe ich Minimalismus. Da bietet sich das NOR–Gatter an.

W: Ok, ich habe verstanden. Dir wird schnell langweilig. Eine Fertig–ALU zu verwenden, kann jeder. Da fehlt die Herausforderung.

D: Ja, genau.

So, dann habe ich MyNOR im Internet veröffentlicht und irgendwann hat das jemand gepostet auf `https://hackaday.com`. Da habe ich einiges an Feedback bekommen. Ein gewisser Marc hat mich gefragt, ob ich mal daran gedacht hätte, Forth auf MyNOR zu portieren. Ich habe mir das angeschaut und war gleich Feuer und Flamme. Ich habe Brodies „Starting Forth“ gelesen und auf dieser Grundlage gleich ein Forth für MyNOR entwickelt, ohne mir je ein existierendes Forth angesehen oder auszuprobiert zu haben. Ich habe eine erste Version dann an Marc geschickt und der war hin und weg. Er berichtete, seine Testprogramme würden laufen. Marc war eine große Hilfe, denn ich hatte ja keine Programme zum Testen. Nach ein paar Debug–Runden habe ich die Forth Version auf meiner Website veröffentlicht und hatte dann die ersten Anwender. Durch das Feedback wurde das Forth dann immer stabiler und anwenderfreundlicher. Es war nun auch für Einsteiger gut verwendbar. Es ist kein Wissen nötig, wie die Maschine intern funktioniert.

W: Du hast für MyNOR auch Erweiterungskarten entwickelt?

D: Ja, genau. Unter anderem ein Display/Keyboard–Modul, um einen richtigen Taschenrechner zu verwirklichen. Den habe ich dann tatsächlich auch lange Zeit auf der Arbeit benutzt.

W: Echt nerdig. Wie ging es dann weiter mit der Entwicklung?

D: Marc hat gerne mit dem Forth auf MyNOR gearbeitet, aber sich beschwert, dass es so wenig RAM gibt und die Kiste so langsam ist. 2 Jahre später war mir dann wieder langweilig und ich habe überlegt, wie man das System schneller machen kann und mehr Speicher haben kann. Aber es musste ja auch wieder was Besonderes sein! Ein bisschen kleiner vielleicht, etwas höher getaktet und natürlich mehr Speicher. Daraufhin habe ich ein System entworfen, direkt mit Forth im EPROM. Bei MyNOR musste das Forth nachgeladen werden. So ist dann My4TH entstanden als dedizierter Forth–Computer.

W: Phantastisch. Aber bevor wir das vergessen: Zwischen MyNOR und My4TH gab es auch noch eine Transistorversion von MyNOR?

D: Stimmt. Die heißt TraNOR und ist die diskrete Version aus Einzeltransistoren von MyNOR. Ich dachte, andere haben auch schon solche Geräte gebaut und MyNOR ist so einfach, da müsste sich etwas machen lassen. So ist dann TraNOR entstanden. Es ist tatsächlich eine 1:1 Transistorkopie von MyNOR. Man kann das EPROM aus einem MyNOR–Computer ziehen und es läuft auf dem TraNOR.

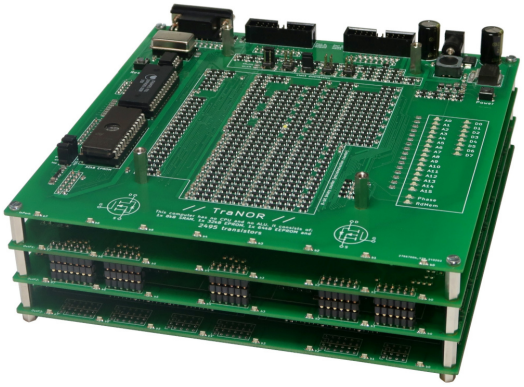


Abbildung 3: Der TraNOR: 2495 Transistoren, verteilt auf 4 Platinen

D: My4TH ist aus dem „MyNOR mit Forth“ entstanden. Ab da habe ich die ganze Entwicklung weitergetrieben auf My4TH. My4TH hat auch ein paar zusätzliche CPU–Instruktionen, um den Forth–Kern zu beschleunigen, denn es gibt so ein paar Operationen, die sind ein bisschen langsam gewesen. Dazugekommen ist z. B. ein 16–Bit–Shift, um die Mathematik–Routinen zu beschleunigen. Bei der Addition habe ich ein wenig geschummelt. My4TH benutzt eine Look–Up–Tabelle. Das spart ca. 20% CPU–Zeit. Bei 8 MHz Taktrate sind so immerhin 6300 8–Bit–Additionen pro Sekunde möglich.

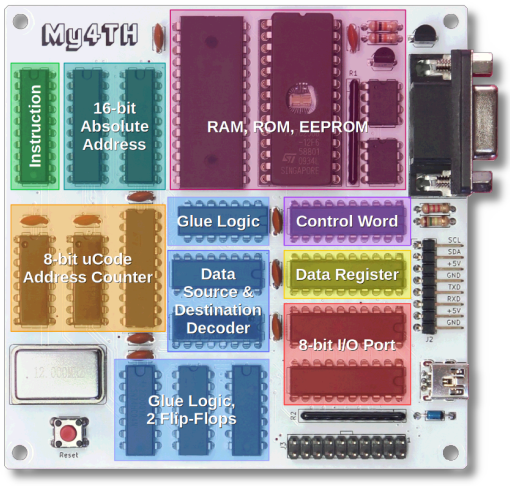


Abbildung 4: Die My4TH Platine. Hier mit den Funktionsblöcken als Overlay

W: Ich finde immer wieder diese Detailtreue in deiner Arbeit. Sei es hier bei der Optimierung des Zeitverhaltens oder auch bei der Platine, die du für den My4TH entwickelt hast. Die ist wirklich ein Kunstwerk. Ich habe mir beim Löten der Platine mehr Mühe gegeben als üblich und auch nach Fertigstellung die Reste des Flussmittels entfernt. Es fühlte sich einfach nicht richtig an, die auf einer so schön gestalteten Platine zu lassen.

D: Das hast du gut erkannt. Die Platine ist ein wichtiger Bestandteil des Projektes. Sie muss gut aussehen und sie muss funktionieren. Ich habe besonderes Augenmerk auf die Beschriftung gelegt. Das ganze Ding muss nachbausicher sein. Da hilft eine gute Beschriftung ungemein und reduziert die Supportanfragen. Möglichst viel Informationen auf die Platine, damit man auch die Funktionsblöcke erkennt, wenn man das Teil in der Hand hält. Da hat man dann auch einen ganz anderen Bezug dazu.

W: Erzähle doch etwas über das Forth Deck. Wie kam das zustande?

D: Es war ein Traum, den ich schon ziemlich am Anfang hatte, noch bevor ich mit dem Layout der My4TH–Platine begonnen hatte. Ich hatte wieder vor, das Gleiche zu machen, wie bei MyNOR. Eine Display– und Tastaturplatine als Erweiterung. Im selben Formfaktor, also 10x10 cm^2. Dann kam das Missgeschick mit dem Display. Das passte nicht. Außerdem hatte ich nur so ein HexPad vorgesehen, wie für den Taschenrechner. Das ist für Forth nicht gut. Also neuer Versuch, aber dieses Mal richtig. Schönes Display und eine vollständige Tastatur, so klein wie möglich. Und portabel, also mit Batterie und Gehäuse. Herausgekommen ist dann das „Forth Deck mini“ mit kleinen Knackfrosch–Tasten. Im Hinterkopf habe ich immer noch, eine größere Version zu haben, die dann normale Tasten hat. Aber die mache nicht ich, da habe ich den Marc angestachelt. Mal sehen, wie sich das entwickelt.

Abbildung 5: Das Forth Deck mini. Ein kleiner, eigenständiger Forth–Computer

W: Das Forth Deck ist eine schöne Anwendung; da kann man My4TH in Aktion sehen, mit Display–Ansteuerung und Tastaturabfrage. Wir konnten das Gerät ja auf dem Forth–Sommertreffen bestaunen und „bespielen“. Es funktioniert gut und fühlt sich wertig an. Wie die Basic–Taschencomputer von Casio oder Sharp aus den 80ern, nur halt mit Forth drauf.

D: Ja, und mit den enthaltenen Fließkommaroutinen kann man das Gerät als einen vollwertigen Taschenrechner verwenden, natürlich mit UPN.

W: Zum Abschluss habe ich noch ein paar kleine Fragen.

D: Los geht's.

W: Bist du sicherlich schon oft gefragt worden: Warum hast du das nicht einfach mit einem FPGA gemacht?

D: Ja, die Frage kommt immer mal wieder auf mich zu. Meine Antwort: FPGA kann jeder. Nein, im Ernst: Mit FPGAs habe ich in meinem Beruf zu tun, das ist langweilig.

W: Der komplette Rechner passt bestimmt in ein FPGA mit 1000 LUTs rein, oder?

D: Machst du Witze? In das kleinste FPGA, welches wir benutzen (so 1300 LUTs), bekomme ich den My4TH locker 10 Mal rein, wenn nicht noch öfter.

W: Da könntest du dir dann einen Rechencluster auf einem Chip definieren.

D: Daran habe ich tatsächlich auch schon gedacht.

W: Nächste Frage: Was machst du beruflich?

D: Ich arbeite bei einem deutschen Hersteller für Industriekameras. Da bin ich hauptsächlich für die Hardware zuständig, designe die Kameraplatinen und die Controller.

W: Letzte Frage: Welche Pläne hast du?

D: Ich habe tatsächlich ein paar Ideen. Die Richtung ist klar: Es wird immer kleiner. Aber mehr möchte ich im Moment noch nicht verraten. Es wird etwas sein, was noch weniger Bauteile benötigt. Und es wird natürlich Forth können.

W: Es bleibt also spannend. Vielen Dank für die Einblicke, die du uns gewährt hast.

D: Es war mir eine Freude.

Links

MyCPU–Website: `http://mycpu.eu`
MyNOR–Website: `http://mynor.org/mynor.htm`
TraNOR–Website: `http://mynor.org/tranor.htm`
My4TH–Website: `http://mynor.org/my4th.htm`
Forth Deck mini: `http://mynor.org/my4th_forthdeck.htm`

volksForth auf dem Agon Light 2

Ulrich Hoffmann

In den letzten Jahren erleben Retrocomputer eine Renaissance. Während einige Enthusiasten die nostalgischen Maschinen der Vergangenheit wieder zum Leben erwecken, gibt es eine wachsende Gemeinschaft, die sich dem Neo–Retrocomputing widmet. Diese Bewegung kombiniert den Charme und die Ästhetik alter Computer mit der Leistungsfähigkeit und den Möglichkeiten moderner Technologie. Der Agon Light™ *und der* Agon Light 2 *sind solche Neo–Retrocomputer, die die Brücke zwischen Vergangenheit und Gegenwart schlagen. Lässt sich darauf auch Forth, speziell* volksForth, *einsetzen? Ja — dank einer CP/M–2.2–Portierung läuft auch volksForth auf echter Hardware und noch dazu flotter als je zuvor.*

Der Agon Light Neo–Retro–Computer

In der Welt der Retrocomputer gibt es viele Strömungen. Es gibt die Sammler, die alte Computer liebevoll restaurieren und am Laufen halten, es gibt die Retro–Gamer, die per Emulation alte Konsolen wiederbeleben und so auch alte Spiele nach wie vor erlebbar machen. Und es gibt die Fans des *Neo–Retrocomputings,* die mit heutiger Technik des 21. Jarhunderts Computer im Stil der alten Zeit bauen und darauf alte Software meist viel schneller laufen lassen als ursprünglich.

Einer dieser Neo–Retrocompurer ist der *Agon Light™*, den sein Erfinder Bernardo Kastrup (TheByteAttic) als „Black–Belt der 8 Bit“ bezeichnet. Es handelt sich um einen brandneuen, vollständig Open–Source 8–Bit–Mikrocomputer des 21. Jahrhunderts. Wie ein klassischer Homecomputer ist auch er ein eigenständiges Gerät und wird direkt an einen VGA–Bildschirm und eine PS/2–Tastatur angeschlossen, ohne dass ein Host–PC benötigt wird. Er ist nach dem Einschalten sofort einsatzfähig und kann in der Grundausstattung in BBC–BASIC progammiert werden. Als Massenspeicher für Programme und Daten fungiert eine µSD–Karte. An der Peripherie bietet der Agon Light™ einen Steueranschluss mit SPI, I²C und mehr als zwanzig GPIO–Leitungen.

Als Prozessor arbeit im Agon Light ein moderner ez80–Prozessor. Dieser Prozessor ist eine verbesserte Version des klassischen Z80, der bis zu 16 MB Speicher adressieren kann und mit 20 MHz läuft. Zusätzlich wird ein ESP32–PICO–D4 als Sound– und Grafik–Coprozessor mit fester Firmware verwendet.

Die Firmware des Agon hat den Namen *Quark™* und besteht aus den beiden Teilen *MOS* (Machine Operating System) und *VDP* (Video Display Processor). MOS ähnelt dabei einem Monitor–Programm ähnlich dem des BBC–Micro–Computers von Acorn. VDP ist die Firmware des als Display–Prozessor arbeitenden ESP32 und ist eine Variation der FABGL–Bibliothek von Fabrizio Di Vittorio, die auch in der ESP32forthStation [3] zum Einsatz kommt.

Der Agon Light™ ist als Open–Hardware–Produkt konzipiert. Das bedeutet, dass alle Details seines Hardware–Designs und alle Spezifikationen öffentlich zugänglich sind und von jedermann eingesehen, modifiziert und weiterverbreitet werden können.

Olimex Agon Light 2

Die bulgarische Hardware–Firma *Olimex* hat sich dem Open–Hardware–Design des Agon Light™ angenommen und das kompatible Board *Agon Light 2* [2] (Abb. 1) entwickelt, das in einigen Parametern vom Originaldesign abweicht, aber zusätzlich den Olimex–UEXT–Konnektor besitzt, an den sich einfach Module der Olimex–Sensor/Aktor–Serie anschließen lassen. So kann beispielsweise ein ESP8266–WLAN–Modul verwendet werden, um den Agon Light 2 netzwerkfähig zu machen.

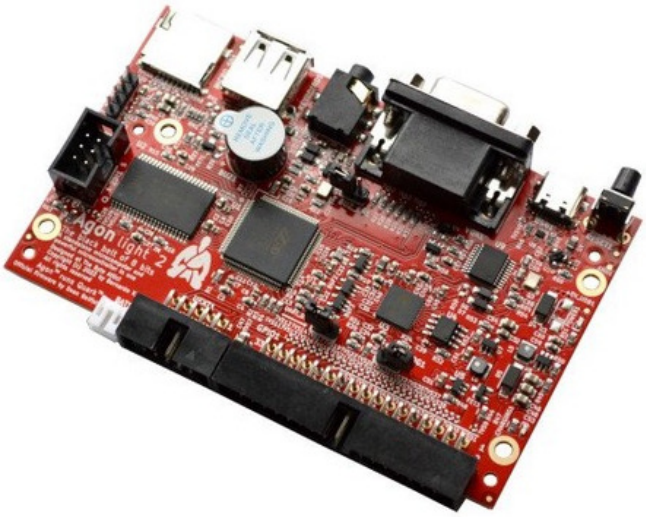


Abbildung 1: Das Agon Light 2 Board

Der ez80 im Agon Light 2 ist mit 128 kB Flash, 8 kB internem und 512 kB externem SRAM augestattet. In der Standard–Ausstattung unterstützt der Agon Light 2 VGA–Grafik mit 320x200 und 64 Farben, 512x384 und 640x480 mit 16 Farben, und 1024x768 mit 2 Farben.

CP/M auf dem Agon Light 2

Wenn denn der Agon Light schon eine Z80–CPU verwendet, dann kann man doch darauf auch CP/M–80 laufen lassen, oder? Das hat sich auch Alexander Sharikhin (nihirash auf GitHub) gedacht und es auf den Agon angepasst.

Seine Portierung von CP/M auf den Agon Light 2 steht auf GitHub [4] zur Verfügung. Sie ermöglicht es, CP/M 2.2 zu nutzen, ohne dass eine Neuprogrammierung des VDP

erforderlich ist. Das System startet direkt aus dem MOS als Anwendung und verwendet die Dateisystemtreiber von MOS, um mit Laufwerksabbildern zu arbeiten. Um CP/M auf dem Agon Light zu nutzen, erstellt man einfach ein Verzeichnis auf der SD–Karte des Agon, legt die Datei `cpm.bin` und die Laufwerksabbilder, die als `cpma.dsk`, `cpmb.dsk` usw. benannt sind, in dasselbe Verzeichnis. Diese Abbilder repräsentieren CP/M–Laufwerke von `A:` bis `P:`. Nach dem Laden und Ausführen der Binärdatei befindet sich der Benutzer in der CP/M–Umgebung. Die CP/M–Laufwerksabbilder lassen sich auf einem PC mit Hilfe der *cpmtools* erstellen und bearbeiten (Format `nihirash`). So läuft Wordstar und Turbo–Pascal auf dem Agon Light 2.

volksForth auf dem Agon Light 2

Nun — wenn CP/M–Programme laufen, dann auch volksForth–83, das seinerzeit 1986 für CP/M 2.2 und CP/M 3.0 entwicklelt wurde und auch auf dem Schneider CPC unter CP/M lief. Gesagt, getan. Die Daten aus dem volksForth–revival–Projekt [5] mit cpmtools in ein Laufwerksabbild kopiert und auf der SD–Karte abgelegt. Nach Starten des CP/M ins richtige Verzeichnis gewechselt lässt sich volksForth unter seinem dortigen Namen `volks4th` starten (Abb. 2).

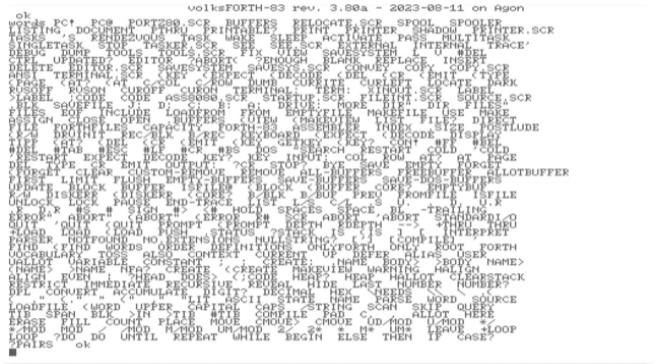


Abbildung 2: volksForth auf dem Agon Light 2

volksForth kommt mit einem Full–Screen–Editor (Abb. 3)[1], der sich auf Wordstar–Art mit Hilfe von `Ctrl-E S D X` steuern lässt. Das besondere des volksForth–Editors sind der Zeichen–Stack — push `Ctrl-J`, pop `Ctrl-K`, copy `Ctrl-L` — und der Zeilen–Stack — push `Ctrl-I`, pop `Ctrl-O`, copy `Ctrl-P`. Auf ihm werden einzelne Zeichen bzw. Zeilen aus dem Screen aufgenommen und an anderer Stelle wieder abgelegt. So lässt sich etwa durch `Ctrl-P Ctrl-O` die aktuelle Zeile duplizieren, indem sie auf den Zeilenstack kopiert — `Ctrl-P` — und dann wieder im Screen eingefügt — `Ctrl-O` — wird. Das ist die *Stack–Variante* des klassichen *Copy&Paste*.

Um auf die Agon Light 2 Peripherie zuzugreifen, stehen in volksForth `PC@` und `PC!` in einer 8080– und einer Z80–Variante bereit. Auf ihnen basierend können moderne Bit–Operationen, wie die von noForth [6] — `*BIS *BIC *BIC BIT* ...` — einfach realisiert werden. Die Welt der Hardware steht damit auch volksForth auf dem Agon offen.

Fazit

volksForth auf dem Agon Light 2 fühlt sich Dank des 20–MHz–Ausführungstaktes angenehm flott an, etwa 5–mal so schnell wie ehemals. Da volksForth als komplettes Entwicklungssystem mit Editor, Debugger, File–System–Zugriff und Quellcodeverwaltung konzipiert wurde, eignet sich der Agon Light mit volksForth hervorragend als Stand–Alone–System zur ablenkungsfreien Entwicklung.

Links

[1] Agon Light: `https://www.thebyteattic.com/p/agon.html`
[2] Olimex Agon Light 2: `https://www.olimex.com/Products/Retro-Computers/AgonLight2/open-source-hardware`
[3] ESP32forthStation: `https://github.com/uho/ESP32forthStation`
[4] CP/M auf dem Agon Light 2: `https://github.com/nihirash/Agon-CPM2.2`
[5] volksForth Projekt der Forth Gesellschaft: `https://github.com/forth-ev/VolksForth`
[6] noForth `https://home.hccnet.nl/anij/nof/noforth.html`

Abbildung 3: volksForth Screen–Editor

[1] Im Original hell leuchtende Schrift auf dunklem Monitorhintergrund; hier invertiert, weil es soherum auf hellem Papier besser zu drucken ist.

Springender Ball

Rafael Deliano

Den „Bouncing Ball“ hat wohl jeder schon mal gesehen. Er ist vom Physikunterricht zum Computer gewandert. Wenn man den Weg weiter zurückverfolgt, kommt man zur praktischen Anwendung.

Die moderne Reinkarnation ist die Grafikdemo. Der „Boing Ball“ (Abb. 1) wurde 1984 auf dem Prototypen des Amiga während der CES[1] vorgeführt. Er blieb seitdem eng mit diesem Computer verbunden.

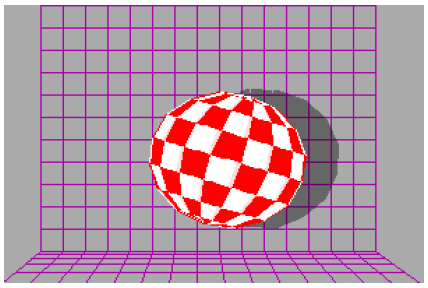


Abbildung 1: „Boing Ball“ des Amiga, 1984

Vorläufer ist die idealisierte physikalische Simulation eines Gummiballs, der von der Tischkante auf den harten Boden fällt, wieder hochspringt und in langsam abklingenden Sprüngen zur Ruhe kommt (Abb. 3). Das ist als anschaulicher Versuch im Physikuntericht vorführbar. Die Mathematik ist dafür soweit simplifiziert, dass sie auch in Handrechnung zu bewältigen ist.

Stoß

In einem weiteren Schritt zurück verlässt man die Demos. Das mechanische Verhalten bei einem Endanschlag hat praktische Bedeutung für die Industrie. Ein Beispiel sind Güterwaggons der Eisenbahn, die beim Rangieren aufeinanderprallen. So ein Stoß muss minimiert werden, sonst fallen Kosten für Transportschäden an.

Computer

Dieses Problem bei den Eisenbahnwaggons wurde schon 1955 auf einem Analogrechner untersucht [1] und tauchte als „Billiard Ball“ 1962 im ersten Computer–Handbuch [2] wieder auf. Die Terminologie dazu ist uneinheitlich, wurde in [3] unter dem Sammelbegriff „Impact Phenomena“ aufgeführt. Diese Schaltungen sind vereinfacht, benötigten aber typischerweise Relais.

Die in echten Anwendungen verwendeten Gummifedern sind nichtlinear [4], damit steigt die Komplexität der Simulation weiter. Das ist aber auch der Grund, warum man von Anfang an am Computer arbeitete.

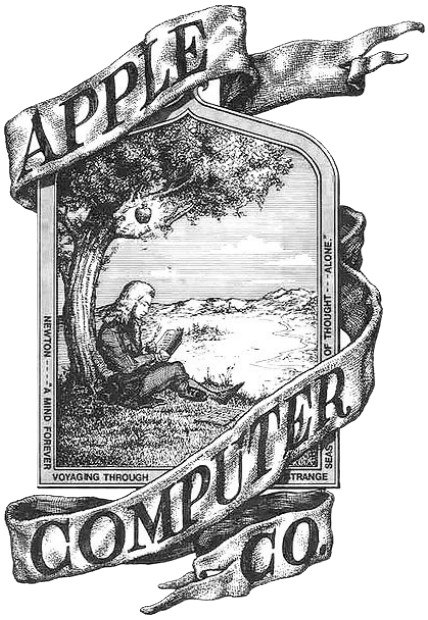


Abbildung 2: Cover Manual Apple I

Simulation

Die Theorie von Newton ist ehedem aus unelastischer Kollision mit einem Apfel hervorgegangen (Abb. 2). Ihr typischer doppelter Integrator führt von der Beschleunigung über Geschwindigkeit zur Position x des Balls (Abb. 3). Die Flugphase ist so direkt darstellbar. An den Punkten des Aufpralls ist der Verlauf aber nicht kontinuierlich. Dort müssen neue Startwerte geladen werden. Im Plot wird darauf verzichtet, mit einer halben Parabel anzufangen (Abb. 5), das vereinfacht das Programm (Abb. 4 , Listing). Beim Aufprall muß die Amplitude `FORCE` schrittweise verkleinert werden. Bei kleinerer Sprunghöhe wird der Ball auch schneller wieder landen. Die Zeitachse wird gestaucht, indem die Skalierung des Integrators `STEP` erhöht wird. Ein Verhältnis beider Konstanten 0,66 zu 1,50 (d. h. $1/1{,}5 = 0{,}66$) würde eine lineare Verkleinerung der Parabeln ergeben. Realistischer ist ein durchhängender Verlauf, hier 0,50 statt 0,66.

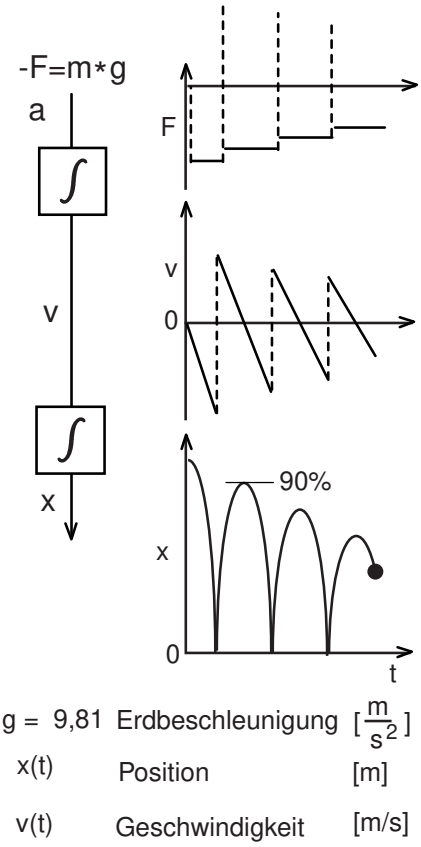


Abbildung 3: Blockschaltbild der Fallbewegung (linke Bildhälfte) und Darstellung der Parameter mit der Zeit (rechte Bildhälfte)

[1] Consumer Electronics Show. Alljährlich in Las Vegas, USA.

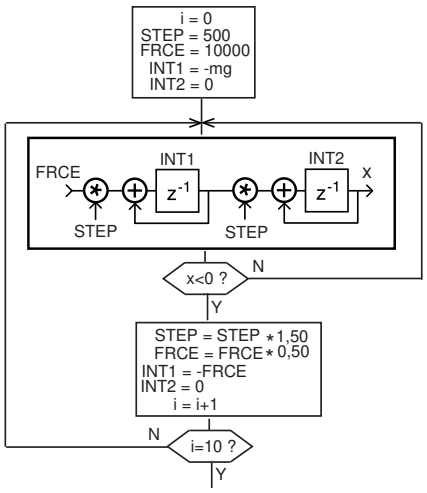


Abbildung 4: Flussdiagramm der Fallbewegung

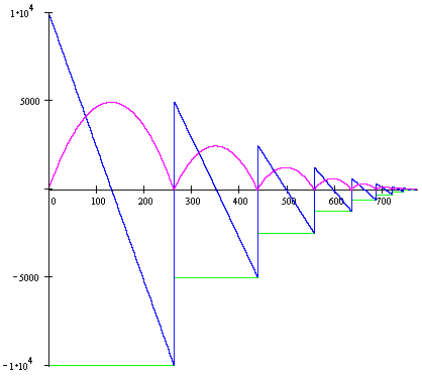


Abbildung 5: Plot

Listing zur Simulation[2]

```
1   <| \ Bounce
2
3   2 ZVARIABLE FRCE    2 ZVARIABLE STEP
4   2 ZVARIABLE INT1    2 ZVARIABLE INT2
5
6   : *0,66   \ ( N1 --- N2 ) always "-"
7     NEGATE D% 43254 U* SWAP DROP NEGATE ;
8
9   : *1,50    \ ( N1 --- N2 ) always "+"
10    DUP 1SHIFT> + ;
11
12  : (*s)
13    STEP @ U* SWAP DROP ;
14
15  : *s   \ ( N1 --- N2 )
16    DUP 8000 AND
17    IF NEGATE (*s) NEGATE
18    ELSE (*s) THEN ;
19
```

```
20  : BOUNCE  \ (  ---  )
21    BEGIN CR
22    FRCE @                DUP SND. \ print force
23    *s INT1 @ + DUP INT1 ! DUP SND. \ print v
24    *s INT2 @ + DUP INT2 ! DUP SND. \ print x
25                         STEP @ ND. \ print step
26    8000 AND                        \ X<0 ?
27    UNTIL
28    STEP @ *1,50     STEP ! \ increment step
29    FRCE @ *0,66 DUP FRCE ! \ decrement force
30               NEGATE INT1 !
31    0                INT2 !
32  ;
33
34  : RUN    \ (  ---  )
35                 D% 500 STEP ! \ load step
36    D% 10000 NEGATE DUP FRCE ! \ load force
37                 NEGATE INT1 ! \ load integrator 1
38                      0 INT2 ! \ load integrator 2
39    9 0 DO BOUNCE LOOP CR ;
40  |>
```

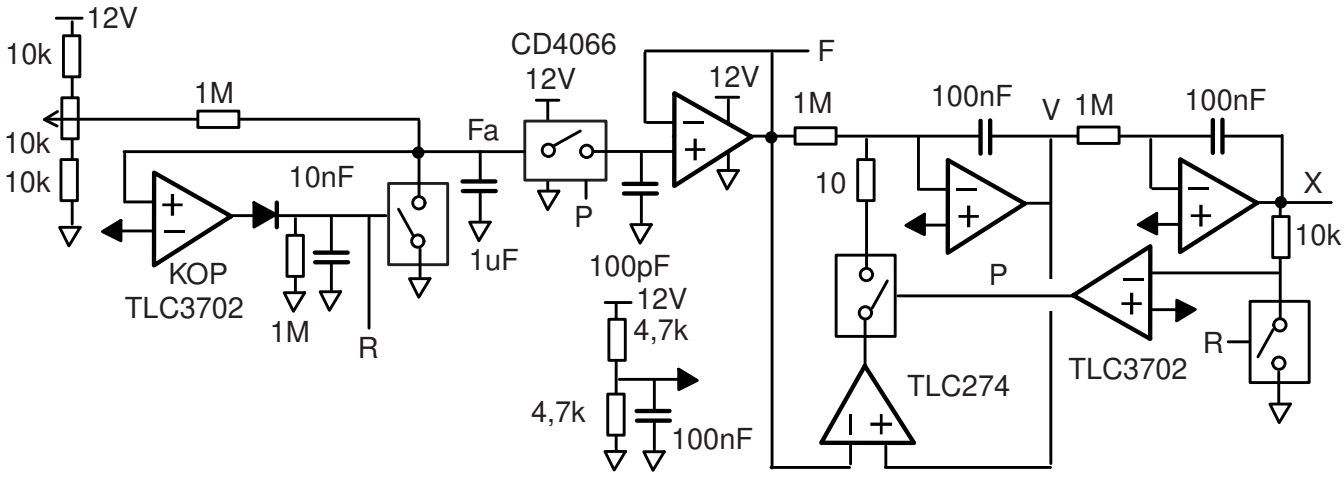


Abbildung 6: Analogschaltung „Bouncing Ball“

[2] nanoFORTH GP32. Für die Non–Standard–Words `zvariable d% 1shift> snd. nd. kann das` Handbuch (pdf) bei Bedarf vom Autor angefordert werden.

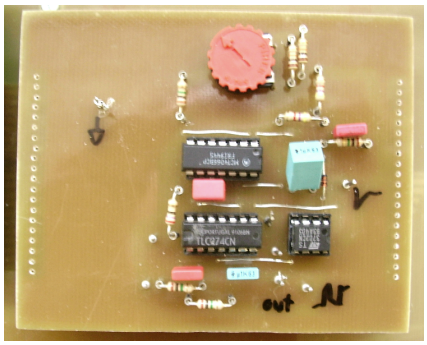


Abbildung 7: Analoger „Bouncing Ball“ als Breadbord

Analogschaltung zur Untersuchung sehr schnellen Prellens

Die Platine nach Abb. 6 und 7 hat keine unmittelbare Anwendung. Ich bevorzuge für ein Verfahren eine Softwarelösung und eine analoge Hardwarelösung parallel zu haben, weil letztere auch in Echtzeit sehr schnell ablaufen kann. Die Simulation auf einem 8 Bit–Controller ist dafür meist zu langsam.

Hier zeigte sich aber, dass die Analogschaltung eher unattraktiv kompliziert und unflexibel ist. D. h., man konnte nicht einfach mal die Zeitkonstanten auf 1 sec pro Hüpfer wie beim richtigen Gummiball bekommen. Das ist für Demos aber erwünscht, weil man da auch einen Lautsprecher anschließen kann, der sein „tock, tock, tock . . .“ beim Aufpall macht, also akustisch „realistisch“ abklingt.

Dennoch: Da man inzwischen die Relais durch Analogschalter ersetzen kann (CD4066)[3], bleibt der Umfang überschaubar (Abb. 7). Für das V–Signal ist zu berücksichtigen, dass die Integratoren invertieren (Schaltung Abb. 6). Die sinkende Sprunghöhe wird hier simpel durch ein RC–Glied erzeugt, R = 1 M, C = 1 µF. Wenn das Ausgangssignal `X` die Null unterschreitet, pulst der Komparator `P`. Dadurch wird ein neuer Wert `F` in die Sample&Hold–Schaltung eingelesen und gleichzeitig der erste Komparator durch eine Schleife mit kurzer Zeitkonstante gezwungen, diesen Wert sofort als Ausgangswert zu übernehmen.

Abb. 8 und Abb. 9 zeigen den geplanten abklingenden Signalverlauf X in Abhängigkeit vom steigenden Fa–Pegel und die Abb. 10 das Ergebnis der verwirklichten Schaltung.

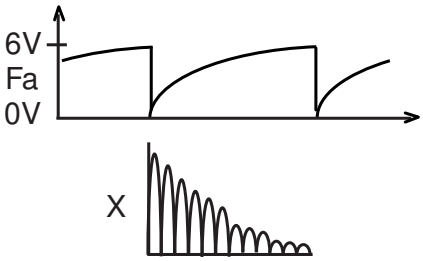


Abbildung 8: Analoges Dekrement

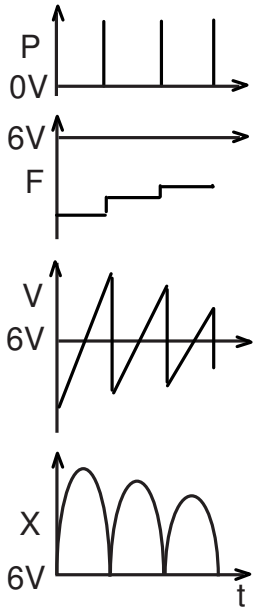


Abbildung 9: Analoge Erzeugung der Parabeln. Wiedergabe des Zeitverlaufs an den Messstellen P, F, V und X der Schaltung

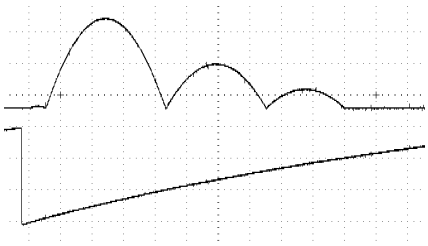


Abbildung 10: Tatsächliche Signale; oben: X, unten Fa

Quellen

[1] Philbrick; „A PALIMPSEST on the Electronic Art“, 1955

[2] Huskey, Korn; „Computer Handbook“, McGraw-Hill 1962

[3] Hausner; „Analog and Analog/Hybrid Computer Programming“, Prentice-Hall 1971

[4] Shigley; „Simulation of Mechanical Systems. An Introduction“, McGraw-Hill 1967

[5] Bernd Ulmann, Analog Computing, De Gruyter 2022.

Und schließlich noch, nur mal so zum Spaß . . .

> „Apr 26, 2009 . . . A rather sophisticated simulation of a bouncing ball in a box. This simulation is completely analog and has been performed on a historic Telefunken analog computer RA 742. (Bernd Ullmann)“
> `https://www.youtube.com/watch?v=jTf7AFdIS2Y`

Bei ihm findet man natürlich auch den „Lunar Lander“, d. h., alle klassischen Anwendungen gabs meist auch auf Analogrechnern.

[3] The CD4066BC is a quad bilateral switch intended for the transmission or multiplexing of analog or digital signals. (Quelle: Datenblatt, Fairchild)

Prellen an Relais untersuchen

Rafael Deliano

In praktischen Anwendungen laufen Simulation und Tests an Prototypen parallel. Die Messungen müssen die Parameter für die Rechnermodelle gewinnen. Bei großer Mechanik ist es naheliegend, Beschleunigungssensoren zu befestigen. Kniffeliger sind Miniatursysteme. Hier als Beispiel die Blattfedern in Kamm–Relais (Abb. 1 und 2). Um die zu beobachten, wurde eine Reflexlichtschranke benutzt.

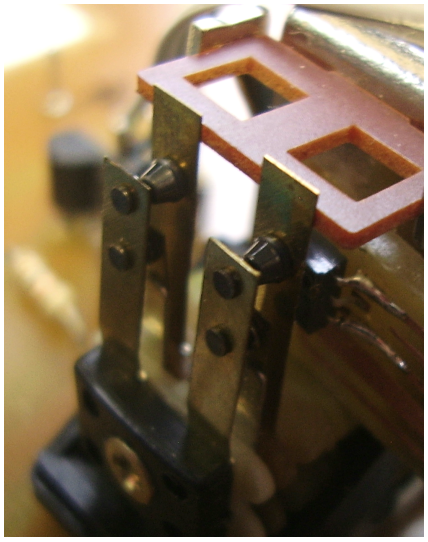


Abbildung 1: Blattfedern eines Kammrelais

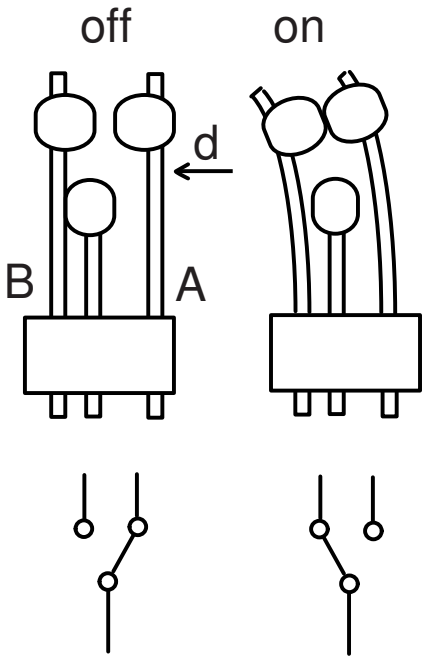


Abbildung 2: Bewegung der Blattfedern (oben) und zugehörige Schalterstellungen, schematisch (unten). Beachte: Die Kontakte sind *nicht* starr, sonden nachgiebig — s. Anmerkung

Zur Reflexlichtschranke

Diese arbeitet berührungslos und ist klein (Abb. 3 und Abb. 4). Metall reflektiert Licht gut. Man muss etwa 1 mm Mindestabstand einhalten und hat von da dann mehrere Millimeter Arbeitsbereich. Bewegt sich das Objekt im Bereich von etwa 1 mm hin und her, ist das Signal in etwa linear (Abb. 5).

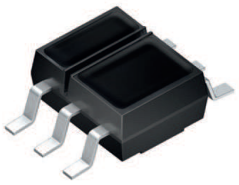


Abbildung 3: Reflex–Lichtschranke SFH9202

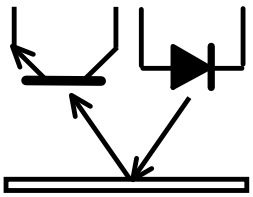


Abbildung 4: Funktionsweise einer Reflex–Lichtschranke

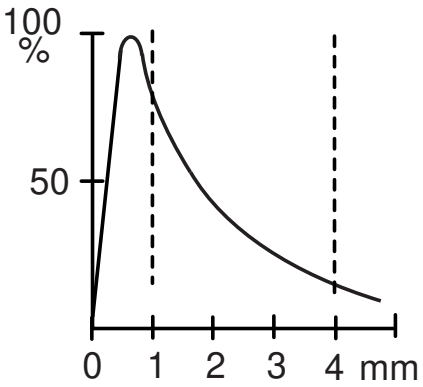


Abbildung 5: Kennlinie der Reflex–Lichtschranke

Auf dem Breadboard ist der Fototransistor als Kaskode[1] beschaltet, um die Geschwindigkeit zu verbessern (s. Schaltung, Abb. 7). Im Messbetrieb muss das Relais, welches in Abb. 9 frei stehend zu sehen ist, mit schwarzer ESD–Folie abgedeckt werden, um das Umgebungslicht auszublenden. Und um das zu verifizieren, ist die LED schaltbar ausgeführt, abgeschaltet sollte das Ausgangssignal dann 0 V sein. Die Ansteuerung des Relais kann über den D/A–Wandler auch rampenförmig erfolgen, um eine Schaltschwelle zu simulieren. Von dieser Option wurde aber noch kein Gebrauch gemacht.

Testaufbau und Beobachtung am Oszilloskop

Für die Analyse des Blattfederverhaltens wurde eine Testschaltung entworfen und aufgebaut (Abb. 7 und Abb. 9). Die Blattfeder wird mittels einer *Reflexlichtschranke* beobachtet.

Einen ersten Eindruck vom mechanischen Verhalten so einer Blattfeder liefert dann das Oszilloskop, man sieht die Bewegung und die abklingende Resonanzfrequenz (Abb. 6).

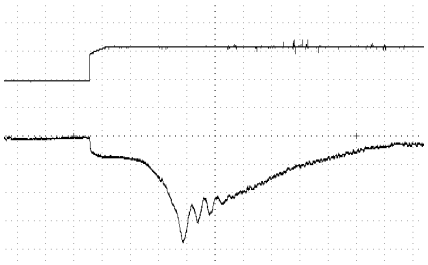


Abbildung 6: Oszilloskop–Bild: Ansteuerung der Spule (oben), Signal der Reflexlichtschranke (unten). Auflösung horizontal 2 msec/div, vertikal 500 mV/div

Man beachte, dass ins Messsignal der 7,2–Hz–Hoch– und der 5,8–kHz–Tiefpass des Verstärkers eingehen. Diese beiden Filter werden gegen

[1] `https://de.wikipedia.org/wiki/Kaskode` Der klassische Miller–Killer.

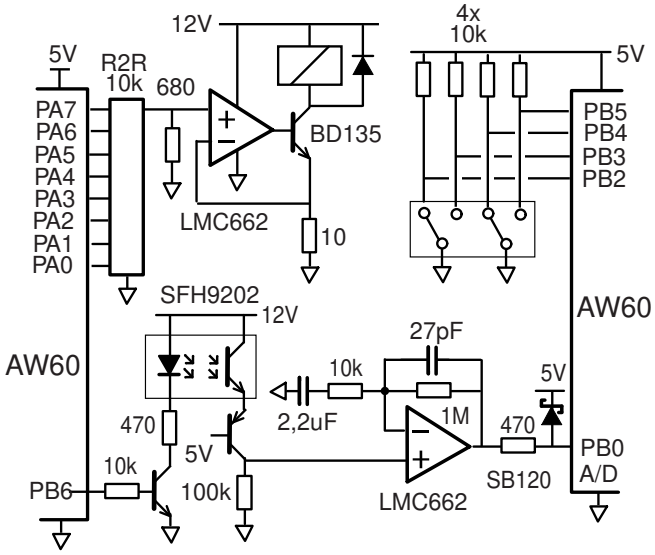


Abbildung 7: Schaltung für die Analyse der Blattfederbewegung in einem Relais. PA0 – 7: Treiber zum Schalten des Relais. PB2 – 5: Lesen der geschalteten Leitungen. PB0: Reflex lesen, A/D–Wandler mit Vorverstärker

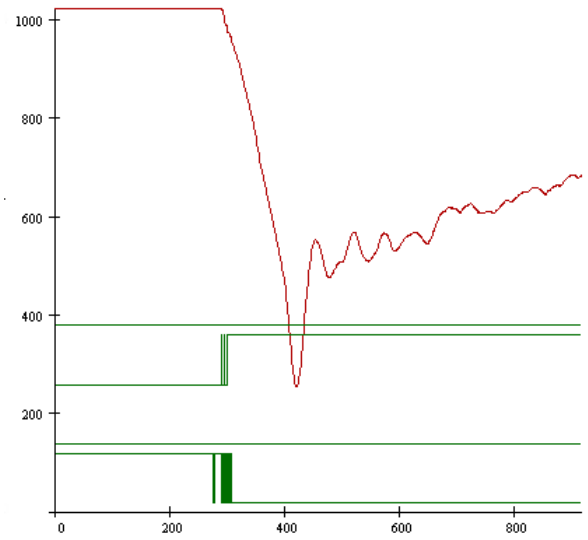


Abbildung 8: Plot der geloggten Daten. Oben: Ausschwingen der Blattfeder beim Öffnen der Kontakte. Mitte: Einschalten der Spule. Unten: Unterbrechung der Leitungen

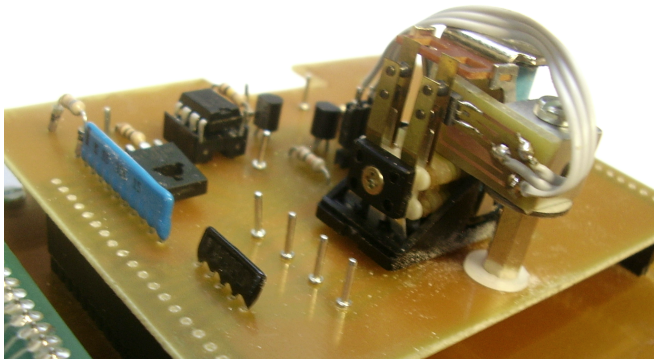


Abbildung 9: Testplatine. In der Mitte das Relais, Gehäuse abgenommen. Ein kleines PCB, mit dem Reflexsensor darauf, ist hinter die beiden Blattfedern eingeschoben worden

Drift und Rauschen eingesetzt, doch da die Resonanzfrequenz der Blattfedern bei ca. 1 kHz liegt, stört das die Beobachtung nicht.

Logger

Die Anwendung erfordert es, auch die 4 binären Kanäle der Kontakte aufzuzeichnen. Ein simples 2–Kanal–Oszilloskop reicht dann nicht mehr aus.

Der AW60 hat genug RAM, um ca. 920 Datenworte á 16 Bit abzuspeichern, je Datenwort 10 Bit A/D–Wert und 4 Bit für die Kanäle. Bei 15 µsec pro Sample (@ 4,19 MHz Busfrequenz) ergibt sich ein abgespeichertes Signal von 14 msec. Das reicht hier völlig.

Man sieht (Abb. 8): Erst öffnet der Kontakt und zeitlich versetzt schwingt die Blattfeder aus.

Anders als beim springenden Ball ist der „Boden“[2] elastisch, um das Prellen zu minimieren. Die Blattfederschwingung klingt dadurch stark gedämpft ab.

Fazit

Ich denke, auf ein Listing kann man verzichten, da solche Datenlogger schon öfter beschrieben wurden und der Code auch nicht portabel wäre.

Nützlich für den Leser ist immer die Einsicht, dass ein simpler SBC[3] für gute Snapshots ein Oszilloskop ersetzen kann. Der 10–Bit–A/D–Wandler hat mehr Auflösung als der 8–Bit–A/D des Oszilloskops. Und die zusätzlichen binären Kanäle sind billig zu haben.

Mit wenig Hardware an *seinem* SBC mit *seinem* Forth, die man beide ja gut kennt, lassen sich rasch Erkenntnisse gewinnen.

Und noch eine Anmerkung

Das hier ist auch eine Demo zum vorherigen Artikel „Springender Ball“. Prellen träte deutlich auf, würde die bewegliche Kontaktstelle auf eine feste Oberfläche treffen, z. B. auf einen unbeweglichen Gegenkontakt. Anlass für diese Untersuchung war die Idee, ob man durch verlangsamte, rampenförmige Ansteuerung des Relais über einen D/A–Wandler solch ein Prellen reduzieren kann. Aber der Hersteller hat bereits federnde Kontaktstellen eingebaut, nichts Starres. So ein Relais ist also mechanisch schon optimiert, durch Schalten kann man da kaum noch etwas verbessern.

Doch das ist nicht immer so. Mechanische Endschalter sind bei kleinen China–CNC–Maschinen oft gar nicht vorhanden und müssen nachgerüstet werden. Da ist es nützlich, an die Optimierung durch federnde Kontakte zu denken.

[2] Der Hersteller baut *keine* starren Kontakte ein, sondern nur Federn, die Relais sind mechanisch optimiert.
[3] System–Basis–Chip; hier der AW60.

On Solving Hexadoku and Debugging Recursive Programs with Message Digests of the Data Stack

François Laagel[1][2]

Debugging Forth code is difficult, especially when dealing with recursive code. One has to maintain invariants and ensure that the data stack is not unduly modified. This document presents a new technique called *stack digesting* which helps the programmer quickly converge on coding errors. It is based on a cryptographic message digest algorithm that is completely specified in [1]. Although the NIST deprecated this message digest generation mechanism in 2011, it serves our debugging purposes well enough. The idea is based on the insertion of strategically placed probing points in the code being debugged so as to make sure that invariants are actually preserved.

Problem Statement

Recursion often is the most natural way to express an algorithm when dealing with intrinsically combinatorial problems (see [2]). Elektor Magazine is a publication that caters to electronics enthusiasts of all kinds. Every two months they publish an issue which includes a puzzle called *Hexadoku*. The rules are similar to the traditional Sudoku puzzle. However Hexadoku extends the regular 3 by 3 nature of Sudoku to a 4 by 4 use case. As a result we end up with a 16 by 16 grid instead of the traditional 9 by 9. Each spot in the grid can be any digit expressed in hexadecimal. The usual Sudoku integrity constraints apply; they are simply extended:

- digit uniqueness in a 4 by 4 sub–quadrant.
- digit uniqueness in an horizontal row.
- digit uniqueness in a vertical column.

This kind of puzzle begs for an automated problem solver and one way to approach it is to think of the problem in a three dimensional manner. The grid itself is implemented as a two dimension array of cells. Each cell of the array is interpreted either as a known value (a power of two corresponding to a resolved value) or a bitmask corresponding to the sum of viable alternatives for that spot. This is similar to the notion of "sum over all possibilities", a central tenet of the path integral formulation of quantum theory (see [3]).

The automated solver I implemented works not by looking for a solution but by converging on one by systematic elimination of unviable alternatives. It does so by alternating between phases of inference and speculation until eventually all spots are resolved (have a cell value that is a power of two).

- `infer` goes systematically over the whole grid and updates bitmasks based on resolved spots values, taking into account the constraints defined by the rules of the puzzle. In effect, this greatly reduces the size of the search space.
- `speculate` selects a currently unresolved spot location and recursively explores the space of open possibilities for that spot. It uses feedback supplied by `infer` to detect constraints violations and backtrack when appropriate. An application specific *transaction stack* is used to record all inferred changes to the grid and to undo them when constraints are detected as being violated. A transaction boundary occurs (and is flagged as such) whenever a speculative decision is made. Basically a transaction includes all inferred changes to the grid since the latest speculation.

In essence it works as a greedy minimalization algorithm aiming at reducing the number of unresolved spots values to zero. Its kernel is as outlined in figure 1. Some implementation details need to be elaborated on at this point.

```
: speculate ( -- success-flag )
  rl+                           \ Increment recursion level

  get-unresolved                \ Look for an unresolved spot
  DUP 0= IF INVERT EXIT THEN    \ Problem solved

  DUP @                         \ S: saddr\sval
  \ The list of set bits in TOS indicate the possibilities
  \ for the selected spot. Explore these alternatives.
  16 0 DO
    DUP I 2^n AND IF
      OVER TRUE SWAP tstk-push \ Insert transaction boundary

      OVER I 2^n SWAP
        +ul |visual -ul !       \ Un-logged update-spot

      infer IF                  \ No inconsistencies detected
        RECURSE IF              \ Stop on the 1st solution found
          2DROP UNLOOP TRUE EXIT
        THEN
      THEN

      \ Backtrack up to the last transaction boundary.
      BEGIN tstk-pop UNTIL
      nbt 1+!               \ Increment the number of backtracks

    THEN
  LOOP

  2DROP FALSE                   \ Dead end reached
  rl- ;                         \ Decrement recursion level
```

Figure 1: The core speculation engine

[1] f.laagel@ieee.org

[2] Institute of Electrical and Electronics Engineers

- `rl+`/`rl-` increment and decrement a global variable that holds the current recursion level. The maximum recursion level also is maintained.

- `get-unresolved` selects the first unresolved grid cell for which the number of set bits is minimal but stricly greater than one. This is where the greedy aspect of the algorithm originates from since we always attack the problem from an angle where the number of options is the smallest at any given point in time.

- `tstk-push`/`tstk-pop` take care of handling the transaction stack. `speculate` goes over the alternatives for one selected spot only. Which explains the `16 0 DO ... LOOP` construct. Whenever a speculative decision is made, that change is logged to the transaction stack under a *boundary* marker. Changes later inferred will also be stacked up so that they can be undone, should the current speculative decision prove not to be conducive to a workable solution.

- `|visual` updates the on–screen representation of the current solution state. It is stack neutral but uses the *next of stack* as the new grid cell value for the spot pointed to by the *top of stack*.

- `+ul`/`-ul` select and deselect the underline font style in order to display the speculation spots in a way that stands out.

- `infer` will deduce all the consequences of a speculative decision until either a constraint violation is detected or the number of unresolved spots can no longer be decreased (a viable situation).

During the execution of the loop in `speculate`, it is essential that the top two values of the data stack be preserved. They are the cell address of the unresolved spot we are working on and that spot's original superposition of possible states. I quickly devised a working implementation that ultimately converged on a solution matching the constraints defined by the rules of the game. Yet, it was not entirely satisfying since zeroes — a zero grid cell value indicates an impossible condition for the associated spot — started surfacing in the grid and I thought that this situation was not detected early enough, thereby negatively impacting the overall performance of the solver. So I worked on early detection and avoidance of zero grid cell values. Associated changes to the source code were mostly in the `infer` code. As it turned out, this is where things started going sideways. I had introduced a bug somewhere in the 650 lines or so of code of the solver. In any sufficiently complex code, these things are bound to happen and I think it is no secret that the classic ways of handling such a conundrum are:

- systematic verification of assumptions. Pre–conditions can easily be verified by extra tests and references to `ABORT"`. Working code should already have this defense mechanism built–in but if it does not, this is definitely the first step to be taken toward fixing a bug.

- an efficient logging mechanism. This is priceless and also should be an integral part of the original code. Logging should be conditional based on some ad hoc *debug* vector flag.

- last resort measures such as improved code documentation and/or third party code reviews. These are either time consuming, expensive or both — definitely not practical ways to address the problem at hand.

Post–condition verification is somehow more elusive and the object of this paper is precisely to describe one, with respect to data stack integrity preservation. After having implemented a reasonable logging mechanism, I realized the code failed to preserve the key invariant `speculate` relies on. It did so at recursion level 16, as illustrated in figure 2. In order to fix this bug rapidly, I introduced the concept of *stack digests*.

```
>speculate at rl 15
                [ 4 , 1 ] <- 5           Cell owned at rl 15
>speculate at rl 16
                 [ 4 , 12 ] <- 7
>speculate at rl 17
                  [ 4 , 4 ] <- C
                  [ 4 , 7 ] <- E
                  [ 5 , 7 ] <- 4
                  [ 10 , 7 ] <- 4       Vertical constraint violation
                  [ 10 , 5 ] <- 8
                  >backtrack
                  [ 10 , 5 ] <-
                  [ 10 , 7 ] <-
                  [ 5 , 7 ] <-
                  [ 4 , 7 ] <-
                  [ 4 , 4 ] <-
                  <backtrack
<speculate at rl 17
                 >backtrack
                 [ 4 , 12 ] <-
                 <backtrack
                 [ 4 , 1 ] <- E          Cell contents altered at rl 16!
```

Figure 2: Failure at recursion level 16

Proposed Solution and Proof of Concept

The original concept was proposed on the *Forth2020* Facebook group and was well received by some of its most respected contributors. [4] proposed a stack checking mechanism that only covers stack depth changes. Later on, [5] developed a complete testing framework for ANS94 compliance. To this day, it remains a reference tool for most testers.

Stack digesting covers both depth and actual stack contents but it does not try to perform stack effect characterization at all. It is a debugging aid only meant to be used as an integrity checking tool.

Basically, a stack digest is just what its name suggests: a cryptographic message digest of the state of the data stack (or a subset of it) as it is when sampled or verified — in this paper support for automated verification is not addressed; visual inspection of the logging output is required for this concept to be of any use. The API is restricted to a single word:

> `SDIGEST` (i*x u – i*x) prints a cryptographic digest of the contents of the data stack, omitting the topmost *u* cells. The algorithm used for producing this digest is implementation defined.

A quick survey of commonly off the shelf available hashing algorithms revealed those most generally agreed upon were crytographic signatures. Among them, it turned out that the easiest to implement and the best documented one was SHA1. [6] provides a detailed pseudo–code description for SHA1.

Once equipped with this new technology, we are now in a position to instrument the application code and to quickly converge on the problem's root cause (figure 3).

```
>speculate at rl 15
              [ 4 , 1 ] <- 5
              >infer 390D7AFB:87768414:F88C2553:C3F8F2A4:4C74CE4F
              <infer 390D7AFB:87768414:F88C2553:C3F8F2A4:4C74CE4F
>speculate at rl 16
               [ 4 , 12 ] <- 7
               >infer 36BF7E72:0BFB296F:E13498C8:D6A8C5DF:C3B6C3D9
               <infer 36BF7E72:0BFB296F:E13498C8:D6A8C5DF:C3B6C3D9
>speculate at rl 17
                [ 4 , 4 ] <- C
                >infer F0812B19:93C42F48:2C610131:FAD0D5D1:24E108A3
                [ 4 , 7 ] <- E
                [ 5 , 7 ] <- 4
                [ 10 , 7 ] <- 4
                [ 10 , 5 ] <- 8
                <infer 36BF7E72:0BFB296F:E13498C8:D6A8C5DF:C3B6C3D9
```

Figure 3: `infer` code is the culprit

The inference code is about 200 lines long but its primary routine is `reduceall` (see figure 4). Through additional code instrumentation, it was determined that an extraneous `2DROP` reference in `reduce4x4` was responsible for this unwarranted data stack corruption.

```
: reduceall ( -- failure-flag )
  reduce4x4 IF                  \ Constraint violated
    TRUE EXIT
  THEN

  16 0 DO
    I get-horiz-mask IF         \ Constraint violated
      UNLOOP TRUE EXIT
    THEN
    ( S: new-possibly-zero-mask ) I SWAP set-horiz-mask IF
      UNLOOP TRUE EXIT
    THEN

    I get-vert-mask IF          \ Constraint violated
      UNLOOP TRUE EXIT
    THEN
    ( S: new-possibly-zero-mask ) I SWAP set-vert-mask IF
      UNLOOP TRUE EXIT
    THEN

  LOOP
  FALSE ;
```

Figure 4: The `reduceall` word

Status and Further Work

An actual implementation for `SDIGEST` and the underlying SHA1 message digest generation is available at [7]. It has been validated against well known test vectors on 64–bit and 32–bit cell targets, regardless of their endianness.

This framework has been successfully used to fix a nasty bug in the inference code of my unpublished proprietary *Hexadoku* automated solver.

The concept could be further extended by having a dedicated digest stack and manually coded verification checkpoints. Each entry on the digest stack would be the output of the SHA1 message digest code (5 cells). This would most likely require some extension of the API and is left as an exercise to the interested reader.

Conclusion

This article was written in the hope that stack digesting could become a useful tool in the Forth programmer's toolbox. It provides a convenient synthetic overview of the state of the data stack, which can be very deep indeed.

Recursion is a powerful technique which allows the developer to formulate solutions to complex problems in very simple terms. However, its use is generally frowned upon in the context of embedded systems software development because stack utilization is basically unpredictable, especially when dealing with heavily data driven algorithms.

Referenzen

[1] D. Eastlake 3rd, P. Jones *RFC 3174: US Secure Hash Algorithm 1 (SHA1)* The Internet Society, September 2001. `https://www.rfc-editor.org/rfc/rfc3174`

[2] Donald L. Kreher, Douglas R. Stinson *COMBINATORIAL ALGORITHMS Generation, Enumeration and Search Chapter 4, Backtracking Algorithms* CRC Press, 2019.

[3] Markus Pössel *The sum over all possibilities: The path integral formulation of quantum theory* Einstein Online, 2006. `https://www.einstein-online.info/en/spotlight/path_integrals/`

[4] Ulrich Hoffmann *Stack checking — A debugging aid* euroFORML Conference Proceedings, 1991. `http://www.euroforth.org/ ef91/hoffmann.pdf`

[5] John Hayes S1I *Core ANS94 Test Harness* Online contents, November 27, 1995. `http://www.forth200x.org/ tests/ttester.fs`

[6] National Security Agency (original designers) *SHA–1* Wikipedia.org, various contributors. `https://en.wikipedia.org/wiki/SHA-1`

[7] François Laagel *SHA–1 sample code for GNU Forth 0.7.3 or SwiftForth 3.7.9* Online contents, August 15, 2023. `https://github.com/forth2020/frenchie68/blob/main/ sdigest-generic.4th`

Forth–Gruppen regional

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

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

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

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

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

Dienste der Forth–Gesellschaft

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

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

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

μP–Controller–Verleih **Carsten Strotmann**
`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**$_p$

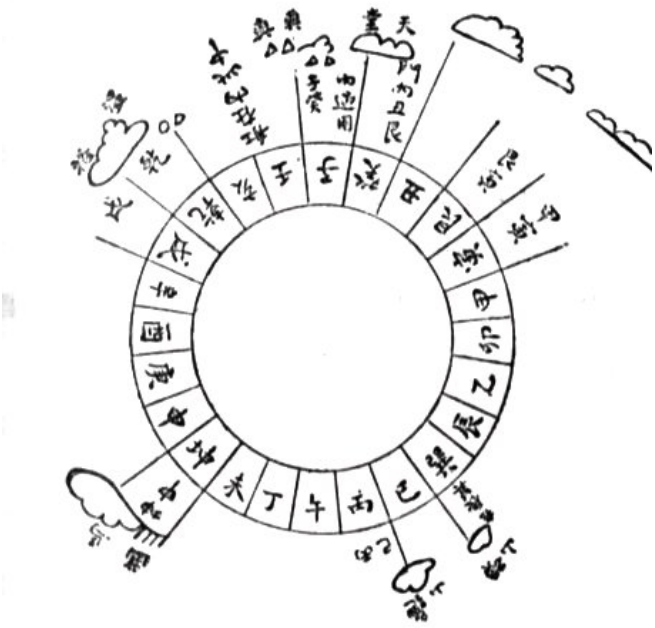
Termine

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*`

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

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

Details zu den Terminen unter `http://forth-ev.de`



Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen hinausgehen? Schreiben Sie einfach der VD — oder rufen Sie an — oder schicken Sie uns eine E–Mail!

Hinweise zu den Angaben nach den Telefonnummern:
Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeitszeiten
g = geschäftlich
Die Adressen des Büros der Forth–Gesellschaft e.V. und der VD finden Sie im Impressum des Heftes.

Ob, und wenn, wo, die Tagung 2024 stattfinden wird,
ist noch völlig offen.
Wünscht euch was!

„Was ist wichtiger“, fragte der große Panda,
„die Reise oder das Ziel?“

„Die Gesellschaft!“, sagte der kleine Drachen.