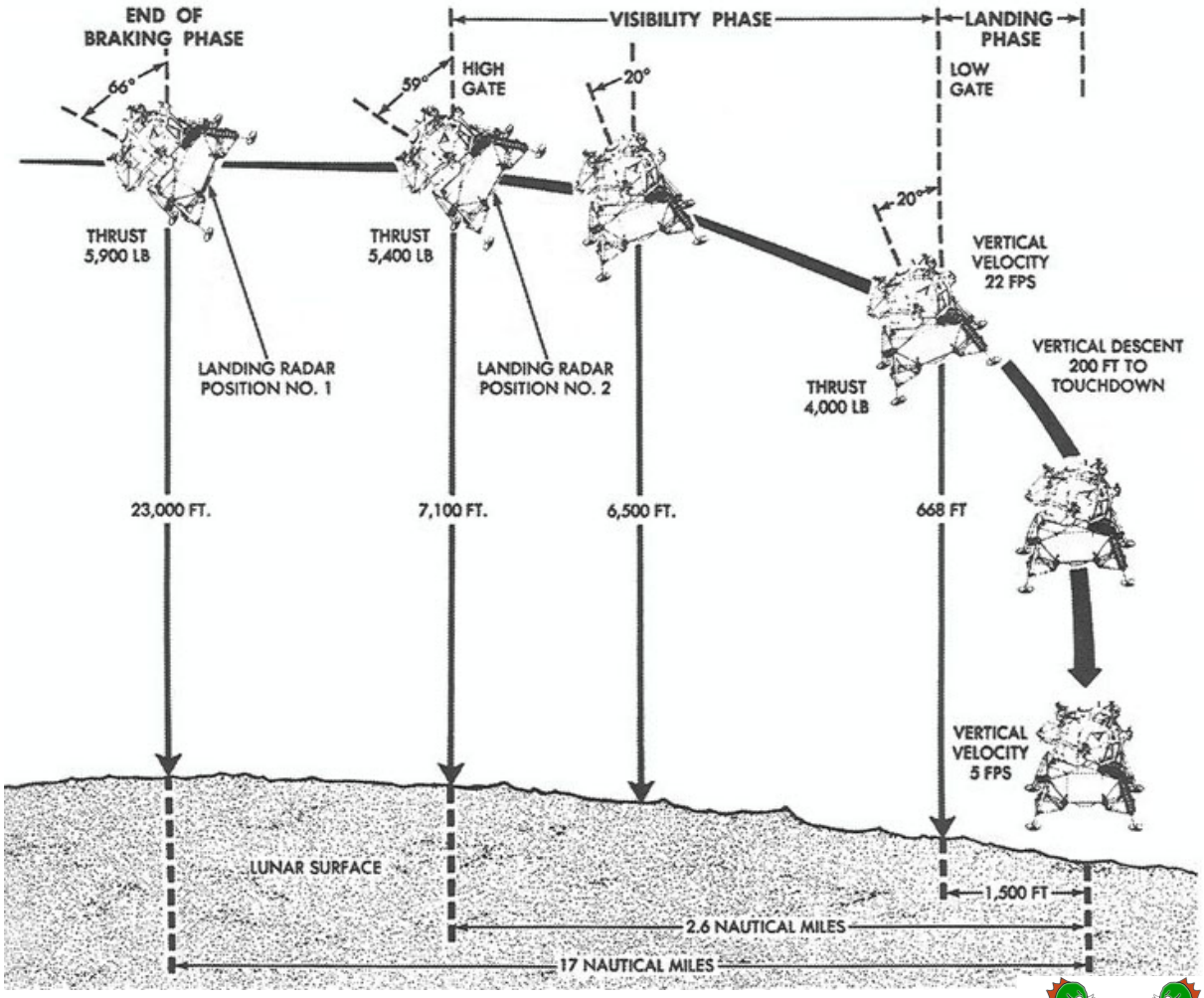
Vierte Dimension

4/2017

Das Forth–Magazin

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



In dieser Ausgabe:

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

Titelbild Apollo 11: Schema des historischen Landeanflugs auf dem Mond.
Quelle: NASA Apollo 11 Press Kit, S.46
`https://www.history.nasa.gov/alsj/a11/Apollo11_Press-Kit_restored.pdf`
`https://www.hq.nasa.gov/alsj/a11/A11_PressKit.pdf`

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

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

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00€ + Porto u. Verpackung

Liebe Leser,

mein Vater prophezeite mir vor wirklich vielen Jahren, dass jedes Jahr ein bisschen schneller zu vergehen scheint, als das Jahr davor. An diesen Satz muss ich in letzter Zeit öfter denken. So ist dieses Jahr eben doch vorüber, bevor das vierte Heft die Druckerei erreicht. So sitzen Bernd, Martin und ich hier auf den 34C3 in Leipzig und tätigen ein paar Handgriffe — nein, fertig wurden wir nicht. Michael liest aus der Ferne mit und Fred lauert unseren Schreibfehlern auf. Vielen Dank an alle Autoren, Hinweisgeber, Mitmacher.

Auf dem 34C3 in Leipzig haben wir mit vielen Leuten gesprochen und versucht, ihnen Forth schmackhaft zu machen. Wir hatten einen ausgezeichneten Platz und jede Menge Publikum. Einer hat uns mitgeteilt, dass wir ihm das vor zwei Jahren beigebracht hätten, und jetzt hätte er auch ein Projekt damit in Arbeit. Ein anderer kam kurz vor dem offiziellen Schluss noch einmal vorbei und bedankte sich dafür, dass wir ihm eine so interessante Sprache gezeigt hätten. Allen, die dabei waren, ein herzliches Dankeschön!

Die *Recognizer* sind ein von Anton Ertl anno 2007 vorgeschlagenes Konzept, welches mindestens in AmForth und Gforth Einzug gehalten hat. Außerdem wurden die Recognizer in den vergangenen Jahren im Standard–Komitee behandelt und haben jetzt den Stand eines *Committee supported proposal* erreicht. Das ist ein historisches Ereignis — das Standard–Komitee hat hiermit zum ersten Mal etwas klar definiert, *bevor* es in den offiziellen Forth–Standard aufgenommen wird. Das heißt sinngemäß: Wenn jemand das implementieren will, dann bitte so.

Rafael Deliano hat einen detailreichen Beitrag zu einer Simulation geschrieben, die aussieht wie ein Spiel. Da soll jemand behaupten, nicht einmal Spielen könne man mit Forth. Albert Nijhof erklärt die Innereien von noForth, und mein Uhrenfimmel erhält eine weitere Ausbaustufe.

Das Thema *Anfänger* aus dem letzten Editorial hat bislang keinerlei Reaktion hervorgerufen. Das ist schade. Gebt Euch einen Ruck! Ich warte ...

Ich wünsche allen viel Spaß beim Lesen und ein gutes neues Jahr 2018.

Erich Wälde

Nachruf

Unser Vereinsmitglied und „unser Bergmann, der Fritz“ Friederich Prinz ist am 29.11.2017 im Alter von 62 Jahren nach schwerer Krankheit gestorben. Die Urnen–Beisetzung fand am 14.12.2017 in Neukirchen–Vlyn im engsten Kreis statt. Viele von uns werden ihn in guter Erinnerung haben, privat ebenso wie von unseren Tagungen und der lokalen Forthgruppe Moers, die er ins Leben gerufen hatte. Mitglied seit 1988 hat er dem Verein immer die Treue gehalten, war Editor unserer Zeitschrift, im Direktorium und im Drachenrat. Er liebte den Gedankenaustausch bei den Treffen, philosophierte gern, immer getragen von seiner aufrichtigen, festen christlichen Einstellung.

Fritz, wir vermissen Dich.

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
http://fossil.forth-ev.de/vd-2017-04

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

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

AmForth 6.6 erschienen

Am 3. Oktober 2017 hat Matthias Trute die Version 6.6 von AmForth freigegeben. Damit sind weitere Änderungen zum Thema Interrupt Bearbeitung eingeflossen, die mit der Clock Works Geschichte aufgefallen sind. Es gibt einen weiteren *deferred prompt* namens `.INPUT`. Diese Funktion wird im äußeren Interpreter `QUIT` nach `REFILL` aufgerufen, und ist mit `CR` vorbelegt.

Die Ankündigung erwähnt außerdem, dass `theforth.net` eine Quelle für Code–Schnippsel werden soll und lädt zum Mitmachen ein. Die Dokumentation der *Clock Works* ist ebenfalls publiziert worden — in Englisch, in der Hoffnung, so mehr Interessierte zu erreichen. Darüberhinaus sind *Recognizer* vom Standard–Komitee als offiziellen Vorschlag aufgenommen worden.

`http://amforth.sourceforge.net`
`http://sourceforge.net/p/amforth/mailman/message/36063805/`

Erich Wälde

Event driven Forth?

Beim Stöbern im Zusammenhang mit der RISC–V CPU Architektur bin ich über das Projekt *kestrel* (wörtlich Falke) von Samuel A. Falvo II gestolpert. Zwar ist die Arbeit an diesem Projekt derzeit eingestellt, aber auf der github Seite findet sich ein Blog mit einigen Artikeln, auch zu Forth. Darin wird die Idee eines *event driven Forth* entwickelt.

Auch wenn ich nicht alles verstehe, was in diesen Artikeln steckt, so sind sie eine Fundgrube für Ideen. Wer also mit den Innereien von Forth experimentiert, wird hier vielleicht inspiriert.

`http://riscv.org`
`https://en.wikipedia.org/wiki/RISC-V`
`https://kestrelcomputer.github.io/kestrel/`

Aus dem Netz gefischt von *Erich Wälde*

forsh — Eine Shell in gforth

Description

forsh is a shell built on top of gforth. It allows one to easily operate a unix–like operating system without leaving the gforth environment and can serve as a fully featured replacement to the bourne shell and derivatives such as bash.

Support

forsh should run on any 64–bit unix–like operating system supported by gforth.

`https://bitbucket.org/cowile/forsh`

Aus dem Netz gefischt von *Carsten Strotmann*

Plugable Type System

Evaluation and implementation of an optional, pluggable type system for Forth

Gregor Riegler

Die Arbeit erschien 2015 als Diplomarbeit an der Technischen Universität Wien. Sie ist in englischer Sprache verfasst und als PDF frei verfügbar (Umfang ca. 80 Seiten). Hier die deutsche Zusammenfassung:

Normalerweise hat eine Programmiersprache genau ein Typsystem, das den Ausdrücken eines Programms Typen zuweist und deren korrekte Verwendung überprüft. Dabei beeinflusst das Typsystem die Semantik der Sprache, sodass die Typüberprüfung als Phase der Programmübersetzung nicht wegzudenken ist. Optionale Typsierung stellt eine Alternative zu dieser Kopplung von Programmiersprache und Typsystem dar und erlaubt damit optionale, verschiedenartige semantische Analysen eines Programms. Diese Arbeit konzentriert sich auf die Analyse statischer Typkonsistenz. Es wird dazu in Haskell ein optionales Typsystem für die stack–orientierte Programmiersprache Forth implementiert. Zusätzlich zu einer Literaturrecherche zu verschiedenen Ansätzen, sowohl statische als auch dynamische Typüberprüfung zu nutzen, dient die Implementierung speziell dazu, die Vorteile wie auch unvermeidbare Kompromisse und Limitierungen eines derartigen Typsystems aufzuzeigen. Im Besonderen wird die Frage behandelt, inwiefern die optionale Typisierung bei der inkrementellen Entwicklung eines Forth–Programms zu gesteigerter statischer Typkonsistenz führt, welche Sprachfeatures dabei unterstützt werden bzw. ein welches Maß an Typannotationen trotz Typinferenz notwendig ist. Dazu wird zuerst bestehende Literatur zur statischen Analyse von Forth–Programmen herangezogen, um eine theoretische Grundlage für die Implementierung der statischen Typüberprüfung zu gewinnen. Danach werden nach und nach Sprachfeatures wie Subtyping, Kontrollstrukturen, Referenztypen, Assertions und Casts eingebaut. In dieser Arbeit wird weiters als Neuheit sowohl die Integration von statisch typkonsistenten indeterministischen Stack–Effekten, als auch die Einbindung von objektorientierter Programmierung, Higher–Order–Programmierung und Compile–Time–Programmierung in den bestehenden Algorithmus zur Typüberprüfung demonstriert.

Zur praktischen Evaluation wird ein funktional korrektes Forth–Programm unter Konfigurationen steigender statischen Typkonsistenz überprüft und dabei werden Verstöße statischer Typkonsistenz ausgewiesen. Dabei zeigt sich, dass das optionale Typsystem bei der inkrementellen Weiterentwicklung des Programms zu gesteigerter statischen Typonsistenz hilfreich ist und aufgrund der implementierten Stack–Effekt–Inferenz dazu nur wenig Hilfe in Form von Typannotationen benötigt. Typannotationen für im Programm definierte Wörter können oft weggelassen werden, sind aber immer bei Gebrauch von Higher–Order–Programmierung notwendig.

`http://repositum.tuwien.ac.at/obvutwhs/content/titleinfo/1643514`

mk

Clock Works 5 — Die UTC Wanduhr

Erich Wälde

Im fünften Teil dieser Artikel–Reihe ([1], [2], [3], [4]) stelle ich eine weitere Anzeige vor: große 7–Segment–Anzeigen, welche mit Schieberegistern und FET–Schaltstufen angesteuert werden. Außerdem habe ich diverse Programm–Bausteine aufgefrischt — hoffentlich zum Besseren. Zwar stellt sich diese Uhr noch nicht per DCF77, aber sie ist mit einem ordentlichen Uhrenbaustein (DS3231) ausgerüstet, so dass die Abweichung auch über längere Zeit gering ausfallen sollte. Die Uhr läuft in Universal Time Coordinated (UTC), kann aber zwei weitere Zeitzonen anzeigen (MEZ, MESZ).

Zeiten

Wenn man sich mit Uhren und der Zeit beschäftigt, stolpert man darüber, dass der eine *UT* sagt, der andere *UTC* und ein dritter *GMT*. Für den praktischen Alltag macht das wenig oder keinen Unterschied, aber wie ist es denn jetzt genau? Wikipedia hilft, wenn man keine Bücher über Astronomie und/oder Zeitmessung zur Hand hat.

Greenwich Mean Time (GMT) war Weltzeit von 1884 bis 1928. Die Bezeichnung existiert zwar noch, gemeint ist damit aber UTC.

Universal Time (UT) folgt den Schwankungen der Erdrotation, misst quasi den Phasenwinkel der Erde. Diese Zeit ist nicht gleichförmig, sie hat *clock skew*. Diese Zeit verwenden hauptsächlich die Astronomen.

Universal Time Coordinated (UTC) ist Weltzeit seit 1972. Sie wird mit Schaltsekunden an UT angeglichen. UTC ist eine gleichförmige Zeit, ihre Sekunden sind von konstanter Dauer. Die Gleichförmigkeit wird durch Atomuhren gewährleistet. Diese Zeit verwenden wir im Alltag, sie wird von den Zeitsendern verbreitet. UTC kennt keine Sommerzeit–Umstellung.

Interessant auch der Wikipedia–Artikel zu *UTC±0* [7]. Darin gibt es je eine Karte mit den hypothetischen und den tatsächlichen Zeitzonen für Europa.

Der Plan

In erster Näherung braucht man für eine Uhr einen Kontroller, eine batterie–gestützte Uhr (RTC) wider das Vergessen der Zeit, eine Anzeige und ein Programm, welches der Uhr das gewünschte Leben einhaucht. Das klang wie eine gelöste Aufgabe, als ich mich dazu hinreißen ließ, das Thema Uhr wieder aufzugreifen. Dass sich das mit allerlei Wirrungen und Unterbrechungen doch so lange hinziehen würde, ahnte ich nicht — sonst hätte ich mich wahrscheinlich nie auf den Weg gemacht und lieber was anderes rausgekramt[1]. Einige Details habe ich bereits vorgestellt, etwa, wie man aus dem Takt von Timer/ Counter2, der zugehörigen Interrupt–Service–Routine und ein wenig Buchhaltung eine Uhr mit Uhrzeit und Datum bauen kann, die periodische Aufgaben ausführt. Zu diesen Aufgaben gehört ebendiese Buchhaltung, das Steuern der Anzeige und was man der Uhr noch an Zusatzaufgaben beibringt[2].

In Abweichung von der in [1] vorgestellten Uhr wird diese Version von Timer/Counter0 getrieben, welcher seinerseits vom 32768–Hz–Takt der externen RTC (DS3231) getrieben wird. Die DS3231–Uhr hat einen temperatur–kompensierten Quarz–Oszillator (TCXO), die Abweichung der fabrikneuen Uhr beträgt deutlich weniger als 5 ppm (ca. 1/2 Sekunde pro Tag). Das spart die Anpassung der Uhrengeschwindigkeit über den Phasenakkumulator.

Die Uhr läuft in UTC, d.h. insbesondere ohne Sommerzeit–Umstellung. Die Zeit kann dennoch in MEZ oder MESZ angezeigt werden. Welche Zeit angezeigt werden soll, wird durch zwei Steckkontakte oder einen Schiebeschalter ausgewählt. Die Umrechnung der Zeit in die gewünschte Zeitzone findet über den Umweg der UNIXschen Epochensekunden [11] statt (in einer abgekürzten Form). Dieses ordentlich zu implementieren, war ein weiterer Umweg des Projekts. Damit werden Zeitzonen zu simplen Offsets (in Sekunden), andere Zeitzonen zu realisieren ist folglich trivial. Der Preis: man muss zu Beginn und Ende der Sommerzeit eine andere Zeitzone auswählen. Das wird mit dem Umstecken einer Brücke oder eines Schiebeschalters erledigt. was meines Erachtens ein akzeptabler Preis ist.

Die DS3231–Uhr hat sich als so zeitstabil erwiesen, dass die Geschichte mit dem DCF77–Signal deutlich an Dringlichkeit verloren hat. Das kommt in der nächsten Iteration. Ich muss die Uhr initial über die serielle Verbindung stellen — *very geek, indeed.*

Design–Entscheidungen

- Strom sparen ist nicht so wichtig — bei 10 mA pro beleuchtetem Segment und flimmerfreier Anzeige ist das im Ansatz erledigt.
- Die Uhr läuft in UTC; andere Anzeige–/Zeiten werden berechnet.
- Die Uhr läuft mit 128 Ticks/s — die Ticks werden später benötigt, um das DCF77–Signal bei jedem Tick abzutasten. Im Moment könnte man auch ohne Ticks auskommen.

[1] Ich befürchte, dass ich noch Projekte und Ideen für die nächsten 100 Jahre in meinem Kopf herumtrage. Prioritäten müssen her!

[2] Einen Kuckuck oder einen Stunden–Gong vielleicht?

Harte Ware

Die Liste der mechanischen und elektrischen harten Zutaten ist dann doch etwas länger. Der Vollständigkeit halber:

- atmega-644p mit Baudraten–Quarz
- DS3231 Uhrenbaustein mit Pufferbatterie/PowerCap, über i2c angebunden
- große 7–Segment–Ziffern, Versorgungsspannung > 8 V nötig
- Schieberegister mit FET–Schaltstufen zur flimmerfreien Ansteuerung (TPIC 6B595)
- 4 Status–LEDs
- zwei Kontakte zur Auswahl der Zeitzone
- Ordentliches Gehäuse!
- Ordentliche Stromversorgung!

Weiche Ware

Das Programm, welches die Uhr zum Leben erweckt, ist zwar länglich, besteht aber zum größten Teil aus bereits vorgestellten Bausteinen. Die aktuelle Version baut auf AmForth–6.6 auf. Im Folgenden werden hauptsächlich die neuen oder unterschiedlichen Teile näher erläutert.

Das Programm

Pin Definitionen

In dieser ebenso wichtigen wie langweiligen Abteilung finden sich zwei Neuerungen: Zum einen `bitmask:` zusätzlich zu `portpin:` — um mehr als einen Pin gleichzeitig zu bedienen. Das sind hier zwar nur zwei Pins, aber im Kleinen zu üben gilt bekanntlich als Tugend. Zum anderen `quotations`. Die Bezeichnungen der LEDs in `ewlib/leds.fs` trugen Bezeichnungen, die von ihrer alten Funktion in den Datensammelstationen herrührten. Die Bezeichnungen passten nicht zu den Anzeigen der Uhr. Also habe ich die Namen der LEDs langweilig verallgemeinert, und für die Aufgaben der Uhr Alias–Namen per `quotations` definiert.

```
PORTA $03 bitmask: _tz

\ PORTB 0 portpin: T0  \ 32768 Hz from RTC
PORTB 2 portpin: led.0
PORTB 3 portpin: led.1
PORTB 4 portpin: led.2
PORTB 5 portpin: led.3
: led_utc  [: led.1 ;] execute ;
: led_mez  [: led.2 ;] execute ;
: led_mesz [: led.3 ;] execute ;

PORTC 0 portpin: i2c_scl
PORTC 1 portpin: i2c_sda

\ abakus/4x7seg display
PORTD 4 portpin: sr_data
PORTD 5 portpin: sr_clock
PORTD 6 portpin: sr_latch
```

Wer einen Schaltplan vermisst — in diesem Abschnitt ist alles wichtige gesagt!

DS3231 RTC

In der letzten Version der Uhr habe ich noch meine steinalten, eigenen Worte benutzt, um mit dem i2c–Bus zu reden. Jetzt habe ich diese Funktionen ausgemistet und benutze die von AmForth mitgelieferten Worte. Außerdem habe ich mir die Geschwindigkeit des i2c–Busses mal genauer angesehen und auf knapp 400 kHz erhöht. Die Implementierung der Funktionen `bcd>dec` und `dec>bcd` habe ich ebenfalls erneuert — besser lesbar für meine Augen. Sie sind allerdings nur für kleine Werte geeignet, so wie sie in der Kommunikation mit der RTC vorkommen (größter Wert 99).

```
\ --- driver: i2c bus, rtc
$68 constant i2c_addr_rtc \ ds3231 7 bit addr
: bcd>dec  ( n.bcd -- n.dec )    $10 /mod  #10 * + ;
: dec>bcd  ( n.dec -- n.bcd )
                       #100 mod  #10 /mod  $10 * + ;

include lib-avr8/hardware/i2c-twi-master.frt
include lib/hardware/i2c.frt \ i2c.n! i2c.n@ i2c.m!n@
include lib/hardware/i2c-detect.frt \     i2c.detect
: +i2c  ( -- )
  i2c_scl pin_pullup_on
  i2c_sda pin_pullup_on
  i2c.prescaler/1
  #6   \ bit rate --- 400kHz @11.0592 MHz crystal
  i2c.init
;
#2000 constant Century
include ewlib/i2c_rtc_ds3231.fs
```

Die RTC zählt zwar Jahre, aber eben nur von 0 bis 99, das Jahrhundert muss man schon selber dazurechnen.

MasterClock

In der bisherigen Implementierung der Uhr gab es eine Liste mit den Zählern von Tick, Sekunde, Minute ... Jahr. Ab und zu hatte ich mir gewünscht, mehrere solche *Uhren* parallel verwalten zu können. Daher habe ich die Definition etwas aufgebohrt. Die Variable `_clock` enthält die Basis–Adresse einer solchen Liste. Funktionen, die auf die Zähler einer Uhr zugreifen, arbeiten relativ zum Wert in `_clock`.

Um die Bedeutung der Felder in der erwähnten Liste besser zu sehen, kann man `structures` verwenden.

```
include lib/forth2012/facility/structures.frt
begin-structure _clock_t
  field: __tick
  field: __sec
  field: __min
  field: __hour
  field: __day    \ offset by 1
  field: __month  \ offset by 1
  field: __year
  field: __tz
end-structure
```

Normalerweise würde man jetzt eine solche Struktur anlegen und ihren Namen bei allen Zugriffen voranstellen:

```
include lib/forth2012/core/buffer.frt
_clock_t buffer: myclock
myclock __sec @ .
```

Die Funktion `myclock` legt die Basisadresse der Struktur auf den Stapel. Die Funktion `__sec` addiert den für dieses Feld gültigen Offset–Wert dazu. Mit der so ausgerechneten Adresse lassen sich alle üblichen Operationen ausführen. Will ich also mehrere solche Strukturen verwalten können, so kann ich anstelle von `myclock` die Basis–Adresse *irgendwie* auf den Stapel legen:

```
myclock _clock !
_clock @ __sec @ .
```

Will ich auf die Zähler einer Uhr zugreifen, dann muss ich den Wert in `_clock` berücksichtigen. Es stellte sich aber heraus, dass `variable` in diesem konkreten Fall eine schlechte Wahl ist. Wenn man auf der seriellen Verbindung die Zähler einer software–Uhr abfragt, während eine andere software–Uhr in einem separaten Task läuft, dann trampeln beide Tasks auf der gleichen Variablen `_clock` herum — ziemlich doof. Die Lösung ist einfach: verwende `user`, hat aber ein unschönes Detail: man muss selbst herausfinden, bei welchem Offset der frei belegbare Platz anfängt [5]. In dieser Version von AmForth ist das der Wert #36.

```
#36 user _clock  \ not variable _clock!
: tick  ( -- addr ) _clock @ __tick  ;
: sec   ( -- addr ) _clock @ __sec   ;
: min   ( -- addr ) _clock @ __min   ;
: hour  ( -- addr ) _clock @ __hour  ;
: day   ( -- addr ) _clock @ __day   ;
: month ( -- addr ) _clock @ __month ;
: year  ( -- addr ) _clock @ __year  ;
: tz    ( -- addr ) _clock @ __tz    ;
```

Außerdem wollte ich eine Bezeichnung der Uhr oder ihr Zeitzonen–Kürzel in druckbarer Form haben. Die Zeichenkette wird ganz normal mit `s" UTC"` hergestellt, ihre Position mit `s,` im Flash–Speicher konserviert. Jetzt muss ich nur noch dafür sorgen, dass die Adresse der Zeichenkette zuverlässig im Feld `__tz` eingetragen wird, und dass die Basis–Adresse ebenso zuverlässig in der Variablen `_clock` landet. Ein *defining word* ist das Mittel der Wahl (auf den Gebrauch von `buffer:` habe ich an dieser Stelle verzichtet, weil die Datenstruktur `_clock_t` nicht unter eigenem Namen zugänglich sein muss).

```
\ define a clock data array including
\ a printing label for its time_zone.
: clock: ( s" tz_label" <clock_name> -- )
  dp >r s, r>                   \ -- flash-p
create
  here dup   ( ram-p )   ,      \ -- flash-p ram-p
  over       ( flash-p ) ,
  _clock_t allot                \ -- flash-p ram-p
  _clock !
  tz !
does>
  dup @i _clock !
  1+  @i     tz !
;
\ s" UTC" clock:  MasterClock
```

Wird im laufenden Programm `MasterClock` aufgerufen, so landet die Basisaddresse der zugehörigen Datenstruktur in der Variablen `_clock`, und die Adresse der Zeichenkette im Feld `__tz`. Die Ausgabe von Datum, Zeit und Angabe der Zeitzone ist jetzt für alle definierten Uhren gleichermaßen möglich.

```
: .date  ( -- )
  year  @    #4 u0.r  \ [char] - emit
  month @ 1+ #2 u0.r  \ [char] - emit
  day   @ 1+ #2 u0.r
;
: .time  ( -- )
  hour  @    #2 u0.r [char] : emit
  min   @    #2 u0.r [char] : emit
  sec   @    #2 u0.r
;
: .tz ( -- )
  tz    @    icount itype
;
```

Die MasterClock im Hauptprogramm zu definieren, sind dann zwei Zeilen:

```
include ewlib/clocks_v0.3.fs
                  \ newfangled clock data as array
s" UTC" clock: MasterClock
                  \ create the "master" clock
```

Als ich das dann alles hatte, war ich mir nicht mehr so sicher, ob ich das wirklich so brauche. Denn ich hatte entschieden, dass andere Zeitzonen durch Umrechnen (aus den Epochensekunden) realisiert werden, und nicht durch getrennte Uhren. Und wegen *einer* Uhr hätte ich den Aufwand ja nicht treiben müssen. Aber wer weiß, vielleicht brauch ich's ja doch noch. Sternzeit oder Swatch Beats, die eine andere Dauer ihrer Zeiteinheiten (quasi Sekunden) haben, wären Kandidaten dafür.

Timeup

Die Funktion `timeup` wird nach Ablauf einer Sekunde aufgerufen und erledigt die Buchhaltung der Zeit: alle Zähler werden gegebenenfalls aufgefrischt. Auch hier muss ich den Inhalt der Variablen `_clock` berücksichtigen, was aber durch die vorigen Definitionen von `sec`, `min`, etc. schon erledigt ist. Zur besseren Lesbarkeit habe ich den Zähler–Limits andere Namen gegeben und die recht unleserliche Schleife über alle Zähler aufgelöst.

```
variable tu.limits  #6 allot
: tu.max.sec   tu.limits  1+  ;
...
: timeup.init
  0    tu.flags      !
  #60  tu.max.sec    c!
  ...
;

: timeup ( -- )
  1 sec +!                          \ sec++
  $02 tu.flags fset                 \ secflag++

  sec @ 1+  tu.max.sec c@ > if
    \ min.over
```

```
    tu.max.sec c@ negate  sec +!   \ sec -= max.sec
    1 min +!                       \ min++
    $04 tu.flags fset              \ minflag++

    min @ 1+  tu.max.min c@ > if
      \ hour.over
      0 min !
      1 hour +!
      $08 tu.flags fset
      ...
    then
  then
;
```

DS3231: hwclock>clock

Die Funktionen, die die Zeitdaten zwischen der RTC und der MasterClock hin– und herkopieren, müssen angepasst werden. Die DS3231 hat einen zusätzlichen Zähler, der den Wochentag anzeigt, und verzichtet aber auf Sekunden–Bruchteile. Der Wochentag wird im Moment nicht berücksichtigt.

```
: hwclock>clock ( -- )
  rtc.get    \ --
     year  !
  1- month !
  1- day   !
  ( wday ) drop  \ ds3231 only!
     hour  !
     min   !
     sec   !
  year @   month @ 1+  tu.upd.limits
;
: clock>hwclock ( -- )
  year @   month @ 1+  day @ 1+
  1 \ sunday ":-)
  hour @   min   @     sec @
  ( Y m d weekd H M S ) rtc.set
;
```

7–Segment–Anzeige

Für die einfachere Verwaltung der 7–Segment–Anzeigen habe ich beschlossen, zuerst den einzelnen Segmenten die entsprechenden Bitnummern zuzuordnen. Daraus werden anschließend die Codes für die Ziffern generiert. Die Nullen am Anfang der Zeilen dienen lediglich der visuellen Symmetrie (`or` nach jedem Segmentnamen). Das ist ein Tribut an die Faulheit und an die experimentelle Informatik.

```
\ 2017-07-09 7seg_1.fs

\ connected bit of each segment
$01 constant _DP
$02 constant _C
$04 constant _D
$08 constant _E
$10 constant _G
$20 constant _F
$40 constant _A
$80 constant _B

\ byte code of each digit
create 7SegDigits
0 _A or _B or _C or _D or _E or _F or       , \ 0
0       _B or _C or                        , \ 1
0 _A or _B or       _D or _E or       _G or , \ 2
0 _A or _B or _C or _D or             _G or , \ 3
0       _B or _C or             _F or _G or , \ 4
0 _A or       _C or _D or       _F or _G or , \ 5
0 _A or       _C or _D or _E or _F or _G or , \ 6
0 _A or _B or _C or                        , \ 7
0 _A or _B or _C or _D or _E or _F or _G or , \ 8
0 _A or _B or _C or _D or       _F or _G or , \ 9
```

Zum Übertragen der gewünschten Zahl an die Anzeige gibt es zwei Funktionen aus der `type`– bzw. `emit`–Klasse. Allerdings wird `emit.7seg` nicht in `type.7seg` in einer Schleife aufgerufen. Das liegt daran, dass das Signal `latch` erst am Schluss gesetzt werden darf. Auch bedient sich `type.7seg` vom Stapel und nicht aus einem angegebenen Speicherbereich.

```
\ transfer n digits, consult lookup table
: type.7seg ( xn .. x1 n -- )
  0 ?do
    dup 0 #10 within if
    else
      drop
      0 \ replace invalid by zero --- questionable!
    then
    7SegDigits + @i  byte>sr
  loop
  sr_latch low sr_latch high
;

\ transfer 1 digit
: emit.7sec ( n -- )
  1 type.7seg
;
```

Bei Überschreitung des zulässigen Wertebereichs wird eine Null ausgegeben. Das mag im vorliegenden Fall angemessen sein, könnte aber auch anders behandelt werden. Eine ordentliche Behandlung des Dezimalpunktes in der Anzeige fehlt im Moment.

Uptime

Streng genommen braucht man keine Buchhaltung darüber, wie lange die Uhr schon läuft. Ich habe das für meine Experimente auf der Suche nach der verlorenen Zeit aber geschätzt — ein kleines bißchen praktischer Luxus.

```
include common/lib/forth2012/double/d-plusstore.frt

2variable uptime
: .uptime  ( -- )
  uptime 2@
  decimal ud. [char] s emit
;
: ++uptime ( -- ) 1.  uptime  d+! ;

: job.sec  ...   ++uptime ;
: init  ...  0. uptime 2! ;
```

Clock Tick

Die Uhr wird durch das 32768–Hz–Signal der RTC angetrieben, allerdings werden die Pulse von Timer/Counter0 aufgenommen. Das Experiment, ob Timer/Counter2 damit auch zurecht käme, steht noch aus.

Der Takt der RTC scheint ausreichend genau zu sein, so dass ich auf den Phasenakkumulator verzichte. Im Hauptprogramm wird lediglich die neue Datei eingeschlossen.

```
\ --- timer/counter0 clock tick
\ 128 ticks/sec
\ timer_0_ overflow
\ clock source: pin T0 @ 32768 Hz (from DS3231)
include ewlib/clock_tick0_external.fs
```

Timer/Counter0 zählt 256 Pulse der RTC und läuft dann über. Der Überlauf produziert einen Interrupt, welcher die registrierte Funktion `tick_isr` zur Ausführung bringt. Diese Interrupt–Service–Routine erhöht den Wert in der Variablen `ct.ticks`, künstlich auf 8 Bit begrenzt.

```
#128 constant ticks/sec
variable ct.ticks
variable ct.ticks.follow

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

Die Hauptschleife der Uhr bekommt eine eigene Variable `ct.ticks.follow`. Ist die Differenz zwischen den beiden Variablen Null, dann hat die Hauptschleife alle aufgetretenen Ticks bearbeitet. Gibt es eine Differenz, dann muss die zugehörige Arbeit so oft erledigt werden, bis die Differenz wieder Null ist. Damit verliert man wenigstens keine Ticks, auch wenn sie vielleicht verspätet bearbeitet werden. Auch werden die beiden Variablen nur von je einer Task beschrieben, was gegenseitige Störungen hoffentlich auch ohne Semaphoren eliminiert. Die Funktion `c0<` wird für den korrekten Vergleich beim 8–Bit Überlauf von `ct.ticks` benötigt.

```
\ ct.ticks is used as 8-bit counter!
: c0<        ( -- )     >< 0< ;
: tick.over? ( -- t/f )
  ct.ticks.follow c@  ct.ticks c@  - c0< ;
: tick.over! ( -- )
  ct.ticks.follow c@  1+  ct.ticks.follow c! ;
```

Jetzt wäre noch zu diagnostizieren, ob eine Sekunde vergangen ist. Aus ästhetischen Gründen habe ich eine Funktion geschrieben, die nachsieht, ob eine ganze oder *nur eine halbe* Sekunde vergangen ist. Nach der halben Sekunde will ich eine LED ausknippsen. Um das zu erkennen, muss ich auf Bit 6 in `ct.ticks.follow` schauen. Änderte sich der Wert von 0 nach 1, ist eine halbe Sekunde vergangen. Ist es umgekehrt, dann ist eine ganze Sekunde vergangen. Zwar müssen wir jetzt nichts berechnen, wir müssen aber den alten Zustand des fraglichen Bits aufheben.

```
variable last.tick[6]

\ one second == 128 ticks
\ half second == 64 ticks
\ that is a toggle on bit 6 of ct.ticks.follow
: half.second.over? ( -- 0|1|2 )
  \ return: 0 == false
  \         1 == half second over
  \         2 == second over
  ct.ticks.follow c@
  $0040 and 0= 0=  \ extract significant bit as t/f
  dup last.tick[6] @ = if
    \ no change, done
    drop 0
  else
    dup 0= if
      \ falling edge, second over
      2
    else
      \ rising edge, half second over
      1
    then
    swap
    ( sig.bit-t/f ) last.tick[6] !
  then
;
```

Benutzen wir Bit 7 zur Diagnose, wissen wir, ob ganze Sekunden vergangen sind.

```
variable last.tick[7]

: second.over? ( -- t/f )
  ct.ticks.follow c@  $0080 and 0= 0=
  dup  last.tick[7] @  = if
    drop 0
  else
    last.tick[7] !
    -1
  then
;
```

Das Ein–/Ausschalten der Ticks sollte keine Überraschungen bergen:

```
: +ticks
  0 ct.ticks         !
  0 ct.ticks.follow  !
  0 last.tick[6]     !
  0 last.tick[7]     !

  [
    %00000000              \ OC.A,B off, normal mode
  ] literal TCCR0A c!
  [
    %00000110 \ ext. clocksource on T0, falling edge
  ] literal TCCR0B c!
                      \ pin T0 input + low  => highZ
  DDRA  c@ $01 invert and DDRA  c!
  PORTA c@ $01 invert and PORTA c!
                \ register interupt service routine
  ['] tick_isr TIMER0_OVFAddr int!
                       \ timer0 overflow int. enable
  TIMSK0 c@ $01 or TIMSK0 c!
;

\ disable ticks
: -ticks
                        \ disable timer0 ovf int
  TIMSK0 c@ $01 invert and TIMSK0 c!
```

```
  $00  TCCR0B c!
  $07  TIFR0  c!        \ clear interrupt flags
;
```

Andere Zeitzonen

Die Umrechnung der angezeigten Zeit in andere Zeitzonen geht den Weg über die Unix–Zeit, auch bekannt als *Epochensekunden*. Ich habe mich für diesen Weg entschieden, weil ich dann alle Buchhaltungsthemen mit Tages– oder Zeitzonenwechseln in der Umrechnung von und in Epochensekunden erledigt habe. Die Zeitzonen werden dann durch simple Offsets in Sekunden realisiert, *krumme* oder selbst definierte Zeitzonen sind dann sehr einfach zu realisieren.

Aber dann hat die Uhr ja einen Jahr–2038–Bug? höre ich Euch klagen [9]. Nein, den hat sich nicht, sondern erst den Jahr–2106–Bug, weil ich die Epochensekunden zwar in 32 Bit, aber ohne Vorzeichen verwende. Und ob die Bauteile der Uhr so lange durchhalten, vermag ich nicht zu sagen. Und man kann immer eine neue Epoche festlegen, wenn einem danach ist. Es ist *software*.

Weil es langweilig ist, die gleichen Sachen immer wieder zu berechnen, habe ich die Umrechnungsroutinen *abgekürzt*, die Datenhaltung hält z.B. für den Beginn des Jahres 2017 das bekannte Ergebnis 1483228800 vorrätig. Diese beiden Werte können beim Jahreswechsel erneuert werden.

Aus dem gleichen Grund werden die Epochensekunden nach dem Lesen der Zeit von der RTC berechnet und in einer Variablen abgelegt. Analog zur oben erwähnten `uptime` werden die Epochensekunden jede Sekunde lediglich um 1 erhöht.

```
\ --- epoch seconds, timezones
: u>= ( n1 n2 -- t/f ) u< invert ;
: d>s ( d -- n ) drop ;
variable              _last_epoch
2variable             _last_esec

#2017        Evalue   EE_last_epoch
#1483228800. 2Evalue  EE_last_esec

include ewlib/epochseconds.fs
2variable Esec
: ++Esec  ( -- )    1. Esec d+! ;
: .Esec   ( -- )    Esec 2@ ud. ;
```

Die Auswertung der gewünschten Zeitzone erfolgt mit der Funktion `_tz.set`. Es wird der zugehörige Wert in der Variablen `EsecOffset` abgelegt und die entsprechende Status–LED eingeschaltet.

```
: +sw  ( -- )  _tz pin_input ;

2variable EsecOffset
: UTC  ( -- )     0. EsecOffset 2! ;
: MEZ  ( -- )  3600. EsecOffset 2! ;
: MESZ ( -- )  7200. EsecOffset 2! ;
: _tz.set
  _tz pin@
  dup 0 = if  UTC
          led_utc on  led_mez off led_mesz off then
  dup 1 = if  MEZ
          led_utc off led_mez on  led_mesz off then
  dup 2 = if  MESZ
          led_utc off led_mez off led_mesz on  then
  dup 3 = if  UTC
          led_utc on  led_mez off led_mesz off then
  drop
;

: init  ...  +sw ;
```

Die Umrechnung und Anzeige der gewünschten Zeit ist dann keine Zauberei mehr, nur Fleißarbeit. Wird eine andere Zeitzone gewählt, dann wird diese erst zur nächsten Minute sichtbar, aber so lange ist eine Minute ja auch nicht.

```
: local.dt ( -- S M H d m Y )
  Esec 2@  EsecOffset 2@  d+  s>dt.short
;
: cd.localtime
  local.dt          \ -- S M H d m Y
  drop drop drop    \ -- S M H
  rot drop swap     \ -- H M
  >r #10 /mod swap  \ -- H.10 H.1
  r> #10 /mod swap  \ -- H.10 H.1 M.10 M.1
  #4 type.7seg      \ --
;
: job.min
  _tz.set cd.localtime
;
```

Task 2

Die ganzen Puzzlesteine müssen jetzt noch zusammenwirken. Das Programm ist mit dem Multitasker ausgerüstet. Task1 bedient weiterhin die Ein–/Ausgabe auf der seriellen Schnittstelle. Task2 macht die Buchhaltung der Zeit und ruft die periodischen Jobs auf.

Die Funktion `run-masterclock` läuft in Task2 — und zwar sooft wie möglich. Wenn ein Tick vergangen ist, werden die Tickzähler erhöht sowie die Funktion `job.tick` aufgerufen. Danach wird geprüft, ob eine halbe oder ganze Sekunde vergangen ist. Nach der entsprechenden Buchhaltung (`timeup`) werden die periodischen Jobs aufgerufen. Das ist sozusagen die innerste Scheife der Uhr. `pause` übergibt die Kontrolle an den nächsten Task in der Liste.

```
: run-masterclock
  begin
    tick.over? if
      tick.over!
      MasterClock
      1 tick +!
      job.tick
    then

    half.second.over?
    dup 0<> if
      dup odd? if       \ half second
        led.1 off
      else              \ second
        led.1 on
        MasterClock
```

```
        timeup
        0 tick !
        1 jobCount !
      then
    then
    drop

    \ run one job per loop, not all at once
    jobCount @
    bv tu.flags fset?
    if
      jobCount @ dup
      Jobs + @i execute
      bv tu.flags fclr
    then
    jobCount++

    pause
  again
;
```

Schluss

Bleibt noch übrig, alle Teile zu initialisieren und einzuschalten, den Tasker zu starten, die Zeit von der RTC zu lesen und in Epochensekunden umzurechnen, eine `turnkey`–Funktion zu schreiben und zu registrieren, dann startet die Uhr auto–magisch.

Natürlich kann man immer noch was dranflanschen:

- Anschluss von DCF-77–Empfänger und automatisches Stellen der Uhr — das ist der Plan
- Helligkeitssensor und automatisches Dimmen der Ziffern
- Piepser für die Ausgabe der Zeit in Morse etwa jede Viertelstunde, oder eben für einen Wecker
- Einen PIR–Sensor spendieren, zum Gucken, ob jemand guckt. Wenn nicht, dann die Anzeige ausschalten.

Dieses Mal könnte man tatsächlich auch weiter vereinfachen und diverse historische Altlasten wegschmeissen!

- Die Ticks weglassen und auf DCF77–Anbindung verzichten
- Die RTC instruieren, dass sie jede Sekunde (oder Minute!) eine fallende Flanke (Alarm, auf Pin 3 SQW) erzeugt
- Diese Flanke als externen Interrupt am Kontroller konfigurieren und bedienen: die Zeit von der RTC holen (und gar nicht selbst verwalten) und anzeigen. Damit werden alle Zähler im Programm überflüssig.

Bis hierher war das eine lange Reise — aber für mich hat sie sich gelohnt. Dank geht an Matthias Trute für AmForth, ein geduldiges Ohr und allerhand Rat; Matthias Koch für den Hinweis auf die TPIC 6B595–Chips; die Teilnehmer des Forth Chats via net2o, für ihre Zeit und Hirnzyklen; die Teilnehmer der Forth Tagung (nicht nur in Kalkar), für Diskussionen und Anregungen. Denn nur im Diskurs kann man scheinbar sicheren Grund als fehlerhaft oder ungünstig erkennen.

So Sachen

1 Auch die längste Reise beginnt mit dem ersten Schritt. Die ersten Schritte liegen bei mir Jahre zurück.

2 Ich wiederhole: Artikel schreiben ist auch ein *code review!*

3 Schöne Platinen und SMD–Bauteile sind schick. Bevor aber jemand die Platine für mich gemacht hat, ist eine handverdrahtete Lochrasterplatine viel schicker, denn die funktioniert wenigstens.

4 Man kann eine (neue) Sprache oder Programmiertechnik lernen wollen und darüber lesen und ein wenig *Hello, World!* spielen. Aber wirklich lernen heisst, sich die Finger an einem ganz konkreten Problem einzuklemmen. Siehe auch [10].

Listings

Das gesamte Programm ist viel zu lang, um es ohne schlechtes Gewissen auf Papier auszudrucken. Der komplette Programmtext kann im Wiki heruntergeladen werden [12]. Eine in englischer Sprache kommentierte Version des Quelltextes findet sich seit Anfang Oktober auf der Webseite von AmForth, in der Abteilung *Commented Projects* [8].

Verweise

1. VD 2016-04, S.15ff — E. Wälde, Clockworks 1 Die kleine Uhr
2. VD 2017-02, S.12ff — E. Wälde, Clockworks 2 Anzeige a la Abakus
3. VD 2017-03, S.12ff — E. Wälde, Clockworks 3 Auf der Suche nach der verlorenen Zeit
4. VD 2017-03, S.23f — E. Wälde, Clockworks 4 Des Rätsels Lösung
5. VD 2017-03, S.27ff — M. Trute, Erlebnisse im USER–space
6. Forth Tagung 2017 (Kalkar), `http://wiki.forth-ev.de/doku.php/events:tagung-2017`, Video und Folien
7. `https://de.wikipedia.org/wiki/UTC%C2%B10`
8. `http://amforth.sourceforge.net/Projects/index.html`
9. `https://de.wikipedia.org/wiki/Jahr-2038-Problem`
10. RFC 1924 Punkt 4: *Some things in life can never be fully appreciated nor understood unless experienced firsthand.* `https://tools.ietf.org/html/rfc1925`
11. Unix–Zeit, Epochensekunden `https://de.wikipedia.org/wiki/Unixzeit`
12. `https://wiki.forth-ev.de/doku.php/projects:clockworks`

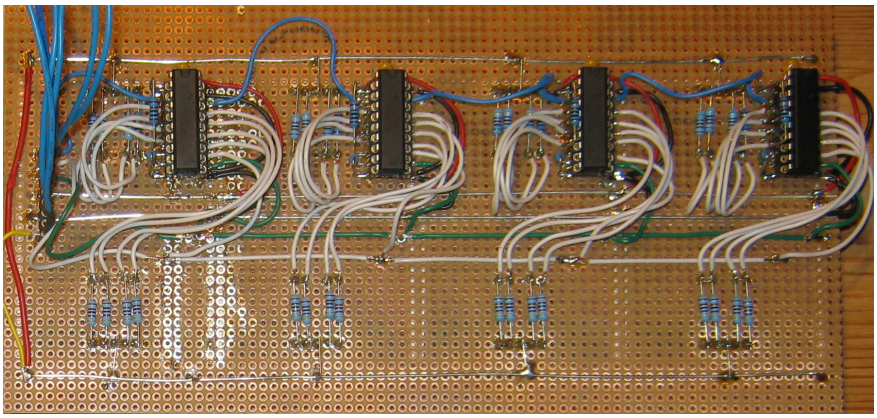


Abbildung 1: Prototypenplatine — echte Handarbeit!

Abbildung 2: Nach der Tagesschau!

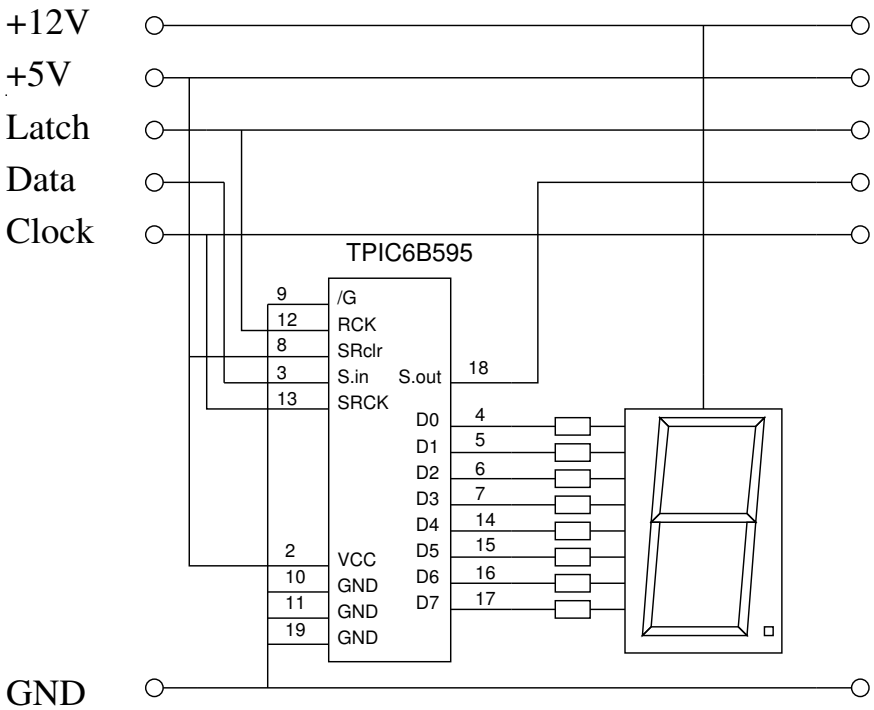


Abbildung 3: Schaltung einer 7–Segment–Ziffer

Lunar Lander

Rafael Deliano

Millionen sahen 1969 die Landung im Fernsehen. Das Apollo–Programm inspirierte den 17–jährigen Jim Storer Mitte der 60er Jahre, eine vereinfachte Simulation als textbasiertes Videospiel zu programmieren.

Teil 1 — Historisches

Storer: „Lexington High School had a PDP–8. It had 8 Teletypes, a small hard drive, and 12KB of main memory, where 8KB was used by the system and 4KB time shared by the users.“ Nicht jede Schule in den USA war so gut ausgestattet, es war eine der Eliteeinrichtungen im Großraum Boston. Sobald Minicomputer Ende der 60er preislich in Reichweite kamen, sprudelten Regierungsgelder. Da die Hardware rasch technisch obsolet wurde, kam in den 70er Jahren Kritik auf. Hat sich trotzdem bezahlt gemacht, hier lernten die Leute, die die PC-Industrie aufbauten, ihr Handwerk.

Storer schrieb das 50–Zeilen–Programm in FOCAL, damals naheliegend für die kleine PDP–8, und schickte es an DEC ein. Deren *Education Product Group* organisierte die Verbreitung von *public domain* Programmen unter den Anwendern. Mit überschaubarem Aufwand machte DEC so seine Minis für Schulen attraktiv. David Ahl, Chef und treibende Kraft der Abteilung, war begeistert. Das Apollo-Programm war noch voll in den Medien, Spiele beim Publikum beliebt. Die Simulation war einfach, aber physikalisch korrekt und passte damit gut für Schulen.

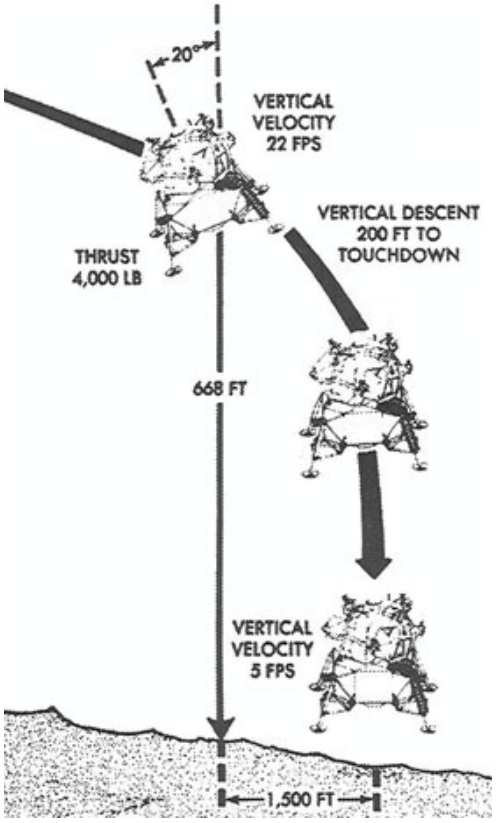


Abbildung 1: Die historische Landung

Er schrieb das Programm auf BASIC um, seine 4k PDP–8 platzte dabei fast aus allen Nähten. Aber damit war es portabel auch auf anderen DEC–Systemen lauffähig und erreichte über DECs EDU newsletter auch die anderen Anwender. Der Funke sprang sofort über, es entstanden weitere, komplexere Varianten.

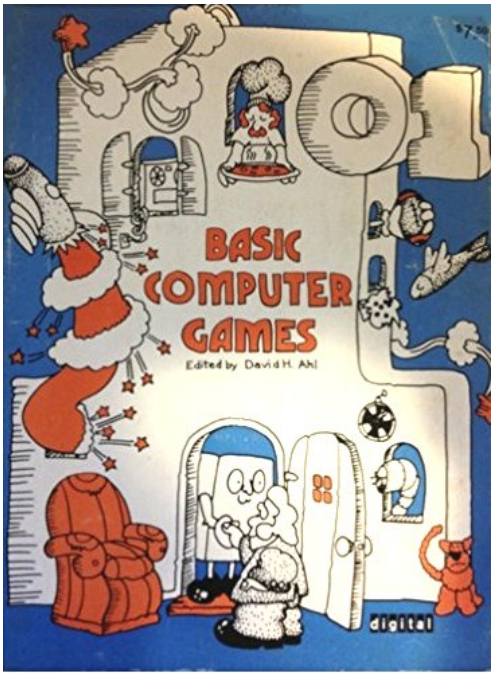


Abbildung 2: Buch–Cover, Ahl 1974

Als Ahl 1973 bei DEC ein Buch „101 BASIC Computer Games“ [1] veröffentlichte, enthielt es gleich drei Versionen: ROCKET (Storer, auf BASIC konvertiert), ROCKT1 (Eric Peters) und ROCKT2 (William Labaree II). Letztere wurden 1971 auch auf PDP–8, aber bereits in BASIC entwickelt.

GT40

Da also wohl jeder bei DEC das textbasierte Spiel gesehen hatte, war es naheliegend, daraus 1973 ein Demo für das neue Vektor–Grafik–Terminal GT40 zu machen. Die Grafiksoftware auf dieser Maschine war allerdings kein kleines BASIC-Programm mehr. Man beauftragte den Consultant Jack Burness, „Moonlander“ dafür zu schreiben. Der hatte sich im vergangenen Jahr den Start der letzten Mission Apollo 17 live in Cape Canaveral angesehen und hatte sichtlich Freude an dem Projekt..

Es dauerte 10 Tage, bis er die integrierte PDP–11 und den Grafik–Prozessor komplett in Assembler programmiert hatte. Die Steuerung erfolgte mit Light Pen, denn

Maus oder Joystick waren noch nicht üblich. Es musste ohnehin nicht sonderlich gut spielbar sein, bewegte Grafik zu zeigen war der Sinn und Zweck.

Abbildung 3: DEC GT40 Terminal [6]

Creative Computing

Ahl wurde Mitte der 70er Jahre Gründer und Herausgeber der Zeitschrift für den langsam entstehenden Markt für Home–Computer. Dort wurde es nochmal [2] und dann weitere Versionen publiziert [3]. Als er 1978 eine überarbeitete Ausgabe von BASIC–Computer–Games für Kleincomputer herausgab [4], herrschte bereits Goldgräberstimmung: die Auflage soll 1 Million überschritten haben. Die Namen hatten sich nun geändert: LUNAR (Ahl/ Storer), ROCKET (Eric Peters), LEM (William Labaree II). David Ahl: „LUNAR, also known as ROCKET, APOLLO, LEM, etc. is, next to STAR TREK and SPACE WAR, the most popular computer game. It is certainly the most popular on small machines“.

Abbildung 4: Der HP25 Taschenrechner

Ein Lunar Lander war auch unter den Spielen, die 1974 von RCA für Board ausgeliefert wurden, das ein Vorläufer des späteren COSMAC 1802 Elf war [5]. Es gehörte in den 70ern praktisch zur Grundausstattung eines Homecomputers. Und war in abgespeckter Form selbst auf programmierbaren Taschenrechnern wie dem HP25 [7] zu finden.

Atari

Arcade Games waren ab 1971 schneller aus den Startlöchern gekommen als die Homecomputer. 1979 hatten sie ihren Zenit fast erreicht. Auch Atari entwickelte nun eine Vektor–Grafik–Maschine. Wie zu erwarten, war das erste implementierte Spiel Lunar Lander.

Abbildung 5: Die Atari Spielkonsole

Atari–Entwickler Howard Delman: "I recall going over to [NASA's] Ames research center - some Atari folks and I had a tour there - and they showed us a Lunar Lander game running on some machine." Es ist nicht bekannt, ob es die alte DEC GT40 Demo war. Atari hatte keine Tastatur mehr, sondern Schieberegler für Schub und Schalter für Richtung. Die realistische Simulation wurde weitgehend durch vereinfachte Spielstufen ersetzt die spielerfreundlicher waren. Man verkaufte ungefähr 4700 Maschinen. Eine gute, aber keine spektakuläre Stückzahl. Auf gleicher Hardware lief das wesentlich erfolgreichere "Asteroids", auf das Atari dann die Produktion umstellte.

Homecomputer

Die Zeitschrift Electronic Games merkte 1981 an: „Sometimes it seems as though every company capable of copying a cassette is trying to sell a game on this theme.“ Auf den Computern der 80er Jahre war es kein Problem mehr, eine Grafikversion zu implementieren. Das Spiel lebte damals noch von seinem Bekanntheitsgrad. Die Umsetzung auf Grafik schien naheliegend, aber der ursprüngliche Lunar Lander war kein Flugsimulator, bei dem ein Pilot mit Joystick ein Raumschiff steuert. Die Atari Version, bei der man manövrierte, um einen Landeplatz zu finden, hatte Anklänge von Tetris. Das waren aber mehrere Ansätze

kombiniert, statt einer simplen Idee, wie es bei Tetris oder Rubik's Cube kompromisslos umgesetzt worden ist.

Die historischen Quellen

[1] „101 Basic Computer Games“, Digital Equipment Corporation, 1973

[2] Ahl, David; „LUNAR“ in: „The Best of Creative Computing Volume 1“, Creative Computing Press, 1976

[3] Cotter, Bill; „LEM“ in: Ahl, „The Best of Creative Computing Vol 2“, Creative Computing Press, 1977

[4] Ahl, David; „BASIC Computer Games“, Creative Computing Press, 1978 — http://www.vintage–basic.net/games.html

[5] Weisbecker; „A Practical, Low-Cost, Home/-School Microprocessor System“, IEEE Computer, August 1984

[6] Edwards, Benj; „Forty Years of Lunar Lander“ 2009 — www.technologizer.com

[7] www.hpmuseum.org/software/25moonld.htm

Teil 2 — Implementierung

Ein 50–Zeilen–Programm umzuschreiben, sollte eigentlich kein Problem sein.

Das alte BASIC–Listing hat aber keine Kommentare. Der Spaghetti-Code ist durch die fleißige Verwendung von GOTO schwer durchschaubar. Die Formeln sind komplex, ihr physikalischer Bezug nicht immer klar. Das war wohl einer der Gründe, warum so viele konkurrierende Versionen entstanden sind. Es erwies sich als praktikabler, eine Spezifikation neu zu codieren.

Modell

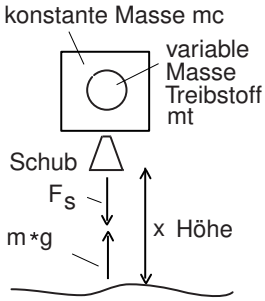


Abbildung 6: Raumschiff

Das Raumschiff ist recht simpel (Bild–6). Für die Messung der Höhe würde man Radar verwenden. Im Programm wird der Wert aus der Bewegung berechnet. „Bremsrakete einschalten“ ist in Science Fiction das übliche Verfahren für jede Landung. Sie ist durch ihren üppigen Treibstoffverbrauch aber technisch ineffizient. Dadurch ergibt sich die richtige Kernidee des Programms, dass eine Mondlandung zwangsläufig knapp am Absturz durch Treibbstoffmangel stattfinden wird. Dieses Optimierungsproblem wurde damals insbesondere in [2] dargestellt. Empfehlung dort: erst frei fallen lassen und dann für die Landung vollen Schub einschalten. In dieser Form fand sich extrem vereinfacht bis etwa 1970 der *soft lunar lander* häufig in Lehrbüchern dargestellt [3] [4].

[1] *Das dritte Newtonsche Gesetz.*

Freier Fall

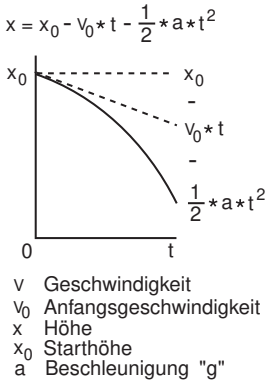


Abbildung 7: freier Fall Höhe

Ohne Schub gilt die Formel für frei fallende Bewegung (Bild– 7). Sie liefert den Wert für den nächsten 10sec–Schritt der Höhe. Wenn man sie differenziert, auch für die Geschwindigkeit (Bild–8). Auf Vorzeichen achten: Aufstieg ist hier - und Abstieg +.

$$v = \dot{x} = -v_0 - a*t$$

Abbildung 8: Geschwindigkeit

Die Masse `m` taucht in der Gleichung nicht auf. Dass schwere und leichte Gegenstände gleich schnell fallen, widerspricht der praktischen Erfahrung. Hier hat man aber den Sonderfall Vakuum, der die simple Formel begünstigt.

Die Startwerte für die Berechnung von Storer sind in Tabelle–1 aufgeführt. Hier wird intern in metrischen Einheiten gerechnet und erst für die Ausgabe wieder rückgewandelt. Der bekannten Konstante für die Beschleunigung auf der Erde g=9,807 m/s^2 entspricht auf dem Mond g=1,622 m/s^2. Sie ist gültig solange man sich nahe der Oberfläche befindet. Bei Raumfahrzeugen ist das nicht immer der Fall. Erforderlich ist ein abhängig von der Höhe modifizierter Wert für die Beschleunigung `g`.

Newton — *Lex tertia*[1]

Die Anziehung von zwei Körpern hängt von ihrer Masse, ihrem Abstand und der allgemeinen Gravitätskonstante ab (Bild–9). Die Konstante für die Mondoberfläche kann man daraus direkt berechnen (Bild–10). Davon abgeleitet die Variante für variablen Abstand (Bild–10). Bei Starthöhe ist die Anziehung also um 20% geringer. Der Verlauf ist ziemlich linear (Bild–11). Auf der Erde müsste `g` wegen der Erdrotation weiter modifiziert werden. Das entfällt auf dem Mond. Bei der echten Landung wird aus einer Umlaufbahn kreisförmig abgesenkt. Dabei muss dann auch ein Trägheitsmoment berücksichtigt werden.

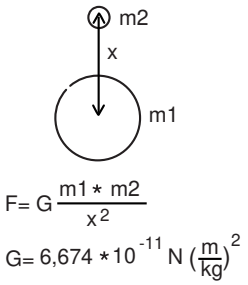


Abbildung 9: Newton 3

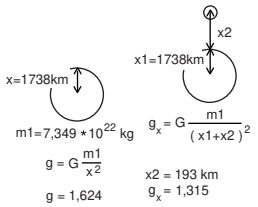


Abbildung 10: Konstante lunar g; modifiziertes g und Beispiel Starthöhe

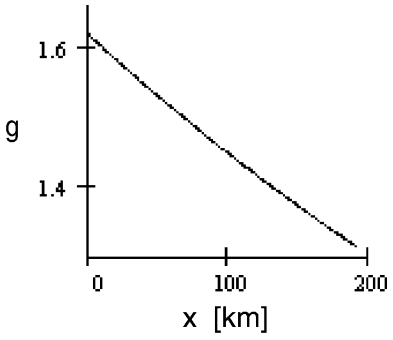


Abbildung 11: g abhängig von Höhe

Schub

Die Schubkraft wirkt der Schwerkraft entgegen (Bild–6). Letztere hängt vom Gewicht ab (Bild–12). Beim Schub (Bild–13) kann man die *burn rate* einstellen (Tabelle 1), während *Isp* eine Konstante des Triebwerks ist. Sie war beim LEM etwa 311 sec. Entsprechend wird `g` modifiziert (Bild–14). Wenn der Benutzer den Schub so weit aufdreht, dass die Kapsel nicht sinkt, sondern aufsteigt, wechselt das Vorzeichen von `g`. In Atmosphäre ist der Schub höhenabhängig, auf dem Mond kann man ihn als konstant annehmen.

$$m = mt + mc$$
$$F_{gx} = g_x * m$$

Abbildung 12: Formel: g abhängig von Höhe

$$F_s = I_{sp} * \dot{m}$$

I_{sp} specific impulse [N * sec/kg]

$\dot{m}$ mass flow rate [kg/sec]

Abbildung 13: Schubkraft

$$F_{gxs} = F_{gx} - F_s$$
$$F_{gxs} = g_x * m - I_{sp} * \dot{m}$$
$$g_{xs} = \frac{F_{gxs}}{m}$$

Abbildung 14: modifiziertes g

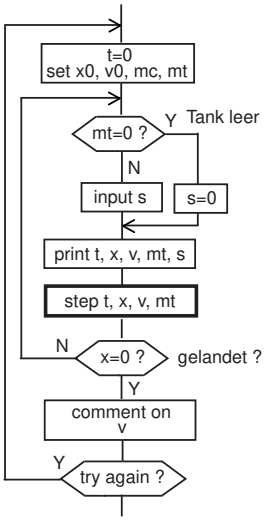


Abbildung 15: Flussdiagramm

Ablauf

Während der Simulation gibt man bei jedem Schritt einen Wert für den Schub `s` ein, daraus wird für den Zeitschritt 10sec ein neuer Satz Werte `x`, `v`, `mt` berechnet (Bild–15). Dann werden die Abbruchbedingungen geprüft. Sobald die Landung erfolgt ist, wird sie anhand der Landegeschwindigkeit („impact velocity“) bewertet (Tabelle 2).

Landegeschwindigkeit	Deutung
<=1.2 mph *1,609 = 1,93 km/h	perfekte Landung
<= 10 mph *1,609 = 16,09 km/h	gute Landung
<60 mph *1,609 = 96,54 km/h	LEM beschädigt
>60 mph LEM	zerstört

Tabelle 2: Auswertung der Landegeschwindigkeit

Ausgabewerte und Umrechnung in metrische Einheiten	Kommentar
32500 lbs *0,4536 = 14742 kg	Gewicht LEM inklusive mt0
7257,6 kg	konstantes Gewicht LEM ohne mt0
16500 lbs *0,4536 = 7484,4 kg	Treibstoff–Startgewicht mt0
120 miles *1,609 = 193,08 km	Starthöhe X0
3600 mph *1,609 = 5792,4 km/h	Startgeschwindigkeit V0
0 ... 200 lb/sec *0,4536 = ... 90,72 kg/sec	Eingabewert s („burn rate“)
3050 N*s/kg	spezifischer Impuls Isp

Tabelle 1: Startwerte

Schrittweite

Höhe `x` und Geschwindigkeit `v` werden quasi gleichzeitig verändert, aber `x` ist von `v` abhängig. Bei Storer und auch in anderen Simulationen [1] wird für `x` der Mittelwert aus altem und neuem Wert von `v` verwendet (Bild–18). Nur im freien Fall ist die Masse konstant. Bei Schub sinkt sie kontinuierlich. Deshalb wird auch für den Treibstoff der Mittelwert angesetzt. Damit sind die Formeln für die Kernroutine `STEP` komplett (Bild–16, Bild–17). Sie wurde in 32/64–Bit–Integer–Arithmerik auf einem *MC68HC908GP32* in *nanoFORTH* codiert. Da hier keine Rückkopplung über Sensoren erfolgt, summieren sich die Fehler, so dass auch diese nur leidlich ausreichend ist.

$$
\begin{aligned}
mc &= 7257{,}6 \ [kg] \\
mt &= 7484{,}4 \ [kg] \\
x &= 193{,}08 \ [km] \\
v &= 5792{,}4 \ [km/h] \\
t &= 0 \ [sec] \\
td &= 10 \ [sec] \\
I_{sp} &= 3050 \ [N*sec/kg] \\
g_{x0} &= 1{,}622 \ [m/sec^2] \\
k &= 1{,}5906
\end{aligned}
$$

Abbildung 16: Initialisierung

$$
\begin{aligned}
&\text{input } s \quad [kg/sec] \\
&t = t + dt \\
&mt_n = mt - s * dt \qquad mt_{avr} = \frac{mt + mt_n}{2} \\
&m_{avr} = mt_{avr} + mc \qquad g_x = g_{x0} - x * k \\
&g_{xs} = \frac{g_x * m_{avr} - I_{sp} * s}{m_{avr}} \\
&v_n = v + g_{xs} * dt \qquad v_{avr} = \frac{v + v_n}{2} \\
&x = x - v_{avr} * dt - \frac{1}{2} * g_{xs} * dt^2 \\
&mt = mt_n \\
&v = v_n
\end{aligned}
$$

Abbildung 17: Schritt

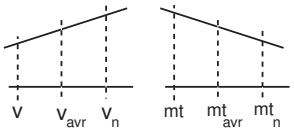


Abbildung 18: Mittelwerte Geschwindigkeit und Treibstoff

Von Storer sind einige Screen–dumps im www zu finden, die Daten davon wurden zum übernommen um das Verhalten der Versionen zu vergleichen. Es wurden drei Testläufe (A, B, C) verglichen (Bilder–19,20,21). Die Werte meiner neuen Version sind gepunktet eingetragen. Schub und damit Treibstoffverbrauch sind identisch. Höhe und Geschwindigkeit stimmen auch überein, solange das Triebwerk ausgeschaltet ist. Schub verringert die Höhe dann aber langsamer, jedoch sinkt die Geschwindigkeit schneller. Am Ende der Landung divergieren die Werte stark (Bild–21).

Für Tests erwies es sich als einfacher, die Schubdaten aus einer Tabelle zu entnehmen, statt einzutippen.

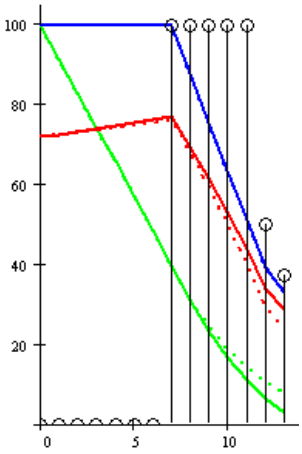


Abbildung 19: Vergleich mit Storer A (grün = Höhe; rot = Geschwindigkeit; blau = Treibstoff; schwarz = Schub; gestrichelt: Meine Version)

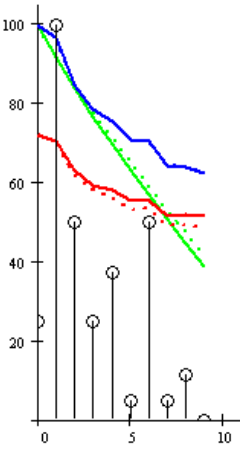


Abbildung 20: Vergleich mit Storer B

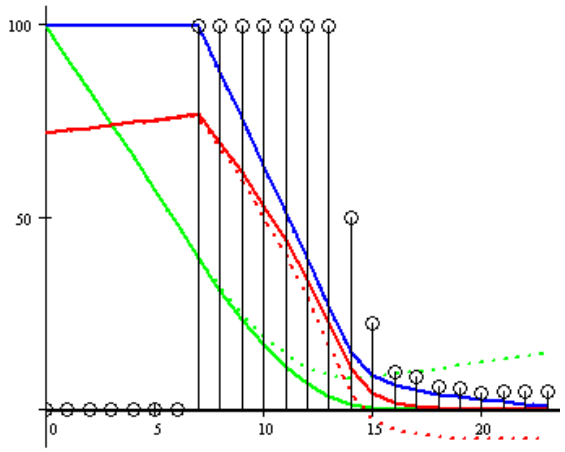


Abbildung 21: Vergleich mit Storer C

Quellen zur Implementierung

[1] www.braeunig.us/apollo/LM-descent.htm

[2] Meditch; „On the Problem of Optimal Thrust Programming for a Lunar Soft Landing“; IEEE Trans. Automatic Control 9 1964

[3] Greensite; „Elements of modern control theory“, Volume 1; Spartan 1970

[4] Kirk; „Optimal Control Theory“, An Introduction; Prentice Hall 1970

[5] Newton, Isaac; „Philosophiae Naturalis Principia Mathematica. 3. Auflage. Innys, Regiae Societatis typographos, London 1726

Listing

```
1   TABLE TXT-TAB
2   \  0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF
3   ," THIS IS A COMPUTER SIMULATION OF AN APOLLO LUNAR                " \  0
4   ," LANDING CAPSULE.                                                " \  1
5   ," THE ON-BOARD COMPUTER HAS FAILED (IT WAS MADE BY                " \  2
6   ," XEROX) SO YOU HAVE TO LAND THE CAPSULE MANUALLY.                " \  3
7   ," SET BURN RATE OF RETRO ROCKETS TO ANY VALUE BETWEEN             " \  4
8   ," 0 (FREE FALL) AND 200 (MAXIMUM BURN) POUNDS PER SECOND.         " \  5
9   ," SET NEW BURN RATE EVERY 10 SECONDS.                             " \  6
10  ," CAPSULE WEIGHT 32500 LBS; FUEL WEIGHT 16500 LBS.                " \  7
11  ," GOOD LUCK                                                       " \  8
12  ," SEC     MI + FT     MPH    LB_FUEL  BURN_RATE                  " \  9
13  ," TRY AGAIN??                                                     " \  A
14  ," FUEL OUT AT % SECONDS                                           " \  B
15  ," ON MOON AT % SECONDS - IMPACT VELOCITY % MPH                    " \  C
16  ," PERFECT LANDING!                                                " \  D
17  ," GOOD LANDING (COULD RE BETTER)                                  " \  E
18  ," CRAFT DAMAGE... YOU'RE STRANDED HERE UNTIL A RESCUE             " \  F
19  ," PARTY ARRIVES. HOPE YOU HAVE ENOUGH OXYGEN!                     " \ 10
20  ," SORRY THERE WERE NO SURVIVORS. YOU BLOW IT!                     " \ 11
21  ," IN FACT, YOU BLASTED A NEW LUNAR CRATER % FEET DEEP!            " \ 12
22  ," FUEL LEFT  %  LBS                                              " \ 13
23
24  \ token % prints number from stack
25
```

```
1   <| HEX \ Lunar Lander
2
3   : INPUT   \ (  --- UN1 )
4   \ 0...999 decimal terminated with CR
5   8000 0 0 0 BEGIN
6   KEY DUP EMIT 30 - DUP 9 U>
7   UNTIL DROP SWAP D%  10 U* DROP +
8   SWAP D% 100 U* DROP +
9   BEGIN SWAP 8000 = UNTIL ;
10
11  : Y/N?    \ (  --- flag ) Y=y=1
12  KEY DUP 59 = SWAP 79 = LOR ;
13
14  \ include:
15  \ U*    ( UN1 UN2  --- UD1 )
16  \ UD*   ( UD1 UN1  --- UD2 )
17  \ UD*/  ( UD1 UD2 UD3  --- UD4 )
18  \     UD4 = ( UD1 * UD2 ) / UD3
19  \ D! D@ DABS DNEGATE D2/
20
21  : TXT.     \ ( UC1 ---  )
22  \ print 64char text line  UC1 = 0 ... 13h
23  CR 40 U* DROP TXT-TAB + DUP 3F + SWAP
24  DO I C@ DUP 25 = IF DROP ND. ELSE EMIT THEN
25  LOOP ;
26
27  BD80 006E DCONSTANT MC  \ 7257472 [g]
28  336C 0072 DCONSTANT MT0 \ 7484268 [g]
29          4 ZVARIABLE MT
30          4 ZVARIABLE MTN
31  CA20 0B82 DCONSTANT X0  \ 193120800 [mm]
32          4 ZVARIABLE X
33  8E80 0018 DCONSTANT V0  \ 1609344 [mm/sec]
34          4 ZVARIABLE V
35          4 ZVARIABLE VN
36   D% 3050  CONSTANT  ISP \ [N*sec/kg]
37          4 ZVARIABLE S   \ [g/sec]
38   D%   10  CONSTANT  DT  2 ZVARIABLE T  \ [sec]
39   D%  100  CONSTANT  DT^2               \ [sec]
40
41  4 ZVARIABLE GXS
42
43  : INIT     \ (  ---  )
44  0 T !   MT0 MT D!   X0 X D!   V0 V D!
45  0 0 S D! ;
46
47  : 0-MIN \ ( D1 --- D2 ) saturate at zero
48  DUP 8000 AND IF 2DROP 0 0 THEN ;
49
50  : D*    \ ( D1 UN1  --- D2 )
```

```
OVER 8000 AND IF 0 2SWAP DABS
2SWAP DROP UD* DNEGATE ELSE UD* THEN ;

\ GX = D% 1622  x = 0
\ GX = D% 1315  x = 193120800 [mm]

: GX             \ (  --- UN1 ) \ [mm/sec^2]
D% 1622 X D@  1 0  9940 0009  \ D% 629056
UD*/ DROP - ;

: STEP                   \ (  --- )
\ mtn = mt - s*dt
  MT D@ S D@ DT UD* D- 0-MIN
  2DUP MTN D!            \  --- mtn )
\ mtavr = (mt + mtn)/2
  MT D@ D+ D2/           \  --- mtavr )
\ mavr = mtavr + mc
  MC D+                  \  --- mavr ) [g]
\ gxs = [ gx * mavr - isp * s ]/ mavr
  2DUP              \ 7258000 ... 14742000 [g]
  GX 0              \ * 1622 [mm/sec^2]
  D% 1000 0 UD*/    \ [milliNewton]
  S D@              \ 0 ... 90720 [g/sec]
  ISP               \ 3050 [N*sec/kg]
  D*                \ [milliNewton] isp * s
  D-
  DUP 8000 AND VN !
  DABS
  2SWAP             \ mavr [g]
  D% 1000 0         \ *1000
  2SWAP
  UD*/
  VN @ IF DNEGATE THEN
                    \ --- gxs ) [mm/sec^2]
2DUP GXS D!

\ vn = v + gxs * dt
  2DUP                   \ [mm/sec^2]
  DT D*
  V D@                   \ [mm/sec]
  D+
  2DUP VN D!             \ --- gxs vn )
\ vavr = (v + vn) / 2
  V D@ D+ D2/            \  --- gxs vavr )
\ x = x - vavr *dt - 0,5 * gxs * dt^2
  X D@                   \ [mm]
  2SWAP                  \ [mm/sec]
  DT UD*
  D-
  2SWAP                  \ [mm/sec^2]
  DT^2 UD*
  D2/
  D-
  0-MIN
  X D!
\ mt = mtn             v = vn
  MTN D@ 0-MIN MT D!   VN D@ V D!
;

: X.           \ (  --- UD1 ) [mm] 193120800
2DUP
1 0 8E7C 0018 UD*/ OVER ND. \ /1609340 = [miles]
8E7C 0018 1 0 UD*/ D-      \ *1609340 / 1.0
D% 1000 0 A69F 0004 UD*/   \ *1000 / 304799
DROP ND.
2 SPACES ;

: V.             \ (  --- D1 ) [mm/sec]
DUP 8000 AND IF DABS 2D ELSE 2B THEN EMIT
                           \ may be negative
D% 100 UD* D% 44704 U/ ND.  \ [mph] /447,039
2 SPACES ;

: M.             \ (  --- UD1 )  [g]
D% 100 UD* D% 45359 U/ ND.   \ [lbs] /453,59
2 SPACES ;
: S.             \ (  --- UD1 ) [g/sec]
D% 454 U/ ND. ;  \ [lb/sec]

: STATUS.        \ (  ---  )
T @ ND. X D@ X. V D@ V. MT D@ M.  \ GXS D@ SD.
;

\ ----------------------------------------
\ test version: read S from RAM

$HEAP CONSTANT BUFFER    \ 100 bytes RAM
1 ZVARIABLE >S

: INPUT                  \ (  --- UC1 )
BUFFER >S C@ + C@  1 >S +C! ;

\ ----------------------------------------

: MT-REPORT   \ (  --- )
MT D@ OR
IF
  MT D@ D% 453 U/  \ LBS
  13 TXT.     \ FUEL LEFT ...
ELSE
  T @         \ seconds
  0B TXT.     \ FUEL OUT AT ...
THEN ;

: V-REPORT   \ (  --- )
V D@ IF FFFF OR THEN
DUP D% 537 U<
IF          0D TXT.   \ perfect landing
ELSE DUP D% 4471 U<
  IF        0E TXT.   \ good landing
  ELSE DUP D% 26823 U<
    IF 0F TXT. 10 TXT. \ craft damaged
    ELSE 11 TXT.
      DUP 0 D% 2000 U/
      12 TXT. \ crater V*0,227  deep
    THEN
  THEN
THEN DROP ;

: RUN      \  (  ---  )
8 0 DO I TXT. LOOP
BEGIN
  INIT CR 09 TXT. CR
  BEGIN
    STATUS.
    DT T +! \ inc T
    MT D@ OR
      IF  INPUT   \ 0 ... 200 [lb/sec]
        D% 454 U* \ *453,592 = [g/sec]
        S D@ S.
      ELSE 0 0    \ S=0 if no fuel left
      THEN
    S D! CR
    STEP
    X D@ OR LNOT   \ until surface reached
  UNTIL
    V D@  D% 447 U/ \ IMPACT VELOCITY % MPH
    T @             \ seconds
    0C TXT.         \ ON MOON ...
    MT-REPORT
    V-REPORT
    0A TXT. Y/N? LNOT
UNTIL CR ;
|>
```

Screenshot

```
THIS IS A COMPUTER SIMULATION OF AN APOLLO LUNAR
LANDING CAPSULE.
THE ON-BOARD COMPUTER HAS FAILED (IT WAS MADE BY
XEROX) SO YOU HAVE TO LAND THE CAPSULE MANUALLY.
SET BURN RATE OF RETRO ROCKETS TO ANY VALUE BETWEEN
0 (FREE FALL) AND 200 (MAXIMUM BURN) POUNDS PER SECOND.
SET NEW BURN RATE EVERY 10 SECONDS.
CAPSULE WEIGHT 32500 LBS; FUEL WEIGHT 16500 LBS.
GOOD LUCK
  SEC     MI + FT   MPH    LB_FUEL BURN_RATE
    0   120     0   + 3600   16500       0
   10   109  4848   + 3629   16500       0
   20    99  3977   + 3659   16500       0
   30    89  2658   + 3689   16500       0
   40    79   882   + 3721   16500       0
   50    68  3921   + 3752   16500       0
   60    58  1205   + 3785   16500       0
   70    47  3285   + 3818   16500       0
   80    38   673   + 3418   14498     200
   90    29  4351   + 2989   12496     200
  100    22  4247   + 2527   10494     200
  110    17   926   + 2026    8492     200
  120    13   334   + 1480    6491     200
  130    10  3262   +  880    4489     200
  140     8  2361   +  781    4088      40
  150     6  2949   +  679    3688      40
  160     4  4547   +  611    3388      30
  170     3  1899   +  540    3087      30
  180     2   308   +  468    2787      30
  190     0  5078   +  394    2487      30
  200     0  2061   +  205    1886      60
  210     0   777   +   87    1486      40
  220     0   344   +   29    1246      24
  230     0   137   +   14    1116      13
  240     0    44   +    6    1006      11
  250     0    68   -    1     895      11
  260     0    35   +    2     815       8
ON MOON AT   270  SECONDS - IMPACT VELOCITY     5  MPH
FUEL LEFT    736   LBS
GOOD LANDING (COULD RE BETTER)
TRY AGAIN??
```

How noForth is made

Albert Nijhof

noForth is a standalone forth for MSP430. It is written in Forth. The metacompiler evolved from september 2014 to january 2016. This year some changes are made to get an even more compact kernel. But that does not effect the way it's build. So we consider it to be a stable version.

What is needed?

- The target code (written in noForth) that describes the whole noForth
- The meta code (written in forth) that will include also the MSP430 meta assembler (cross assembler)
- A standard 32bit forth on the host computer

What to do?

- Include the meta code on the host computer (for example in Win32forth).
- Then include the target code.
  - You will be asked to choose your processor+board. Then the meta compiler converts the target code into a *binary image* of noForth. That image is saved as an *intel-hex* file.
- The intel-hex file can be sent to the chip with a *programmer*.

The target code

The target code defines noForth "in terms of itself".

At first sight this seems nonsense but we use a metacompiler (an auxiliary noForth in the host forth) that is able to convert the target code into a noForth image for the MSP430.

Image building

The binary image of noForth is being built while the metacompiler is interpreting the target code. The metacompiler is able to look up words in the growing image. Of course the noForth words in the image can not be executed on the host computer.

First, let's take the *blue words*[1] in the example 1. These words (and numbers) must be compiled into noForth colon definitions. A word can be compiled when:

a) it already exists in the image

b) it is not immediate

c) STATE is true

The *red words*[2] are executed by the metacompiler: immediate words within colon definitions and words not in colon definities.

And the *green words*[3]? - Green text is handled by the preceding red word, not by the metacompiler itself.

```
code dup    tos sp -) mov next end-code
code drop   sp )+ tos mov next end-code
code !      sp )+ tos ) mov
            sp )+ tos mov next end-code

20 constant bl
: space ( -- )    bl emit ;
: spaces ( n -- ) 0 ?do space loop ;
: type ( a n -- ) 0 ?do count emit
                  loop drop ;
variable base
variable state
: decimal   0A base ! ;
: [ ( -- )  false state ! ; immediate
: ] ( -- )  true state ! ;
: char ( -- ch )
   bl word count 0= ?abort c@ ;
: [char]
   char postpone literal ; immediate
: repeat postpone again
         postpone then ; immediate

* red  blue  green
```

Figure 1: Examples of target code

x-words

Probably all red words already exist in the host forth but most of them are useless for the metacompiler.

Example: host forth headers differ from noForth headers, so the red ":" has to do other things than the original ":" in the host forth.

Therefore we have to redefine the red words in such a way that they will behave like noForth words. In fact we make an (incomplete) noForth simulator on the host computer. That's not really hard, but a very confusing problem arises: *Conflicting names.*

Our solution is very simple and brute: we put an "x" before the names of redefined red words. So we get:

[1] Print is black&white. „Blue words" are printed in strong letters. — For better understanding see the original file on the internet, there it is colorfull.) [fette Buchstaben.]

[2] „red words" are underlined. [unterstrichene Buchstaben]

[3] „green words" are printed in italic letters. [cursive Buchstaben]

```
x; x: xCODE xLOOP xIF xVARIABLE xIMMEDIATE
...etc.
```

When all the necessary x-words are defined we put them, without the "x", in a vocabulary `META`.

```
VOCABULARY META
ONLY FORTH ALSO META DEFINITIONS FORTH
: : x: ;
: ; x; ;
: CODE xCODE ;
: LOOP xLOOP ;
: IF xIF ;
...etc.
```

The red words end up in the META vocabulary with their normal names while META is not in the search-order. This approach seems a bit clumsy and not very elegant, but it is effective and above all: the code remains clear and easy to read.

Why not define the red words right away into META? - Because red words often contain other red words and that would require a lot of vocabulary juggling.

Keep it simple is the motto.

Our own meta-interpreter

Interpreting the target code (the red and blue words) becomes simple if we don't use the host forth interpreter, but write our own interpreter instead.

```
\ The metacompiler, interpret one word
: WINTERPRET ( bl-word -- )
   xSTATE @
   IF search-word-in-image
      found?
      IF not-immediate?
         IF xCOMPILE, EXIT
         THEN
      THEN
   THEN
   search-word-in-meta
   found?
   IF EXECUTE EXIT
   THEN
   is-it-a-number?
   IF xSTATE @ IF xLITERAL THEN EXIT
   THEN
   Error ;

: METACOMPILING
  BEGIN BL WORD WINTERPRET
  AGAIN ;
```

Figure 2: The metacompiler

This metacompiler as such is straightforward[4] and the task to build a noForth from the target code is now divided into a lot of relatively small problems: (re)defining all red words that appear in the target code.

The target code starts with the word `:::NOFORTH:::` that activates the metacompiler. The word `;;;NOFORTH;;;` at the end of the target code stops the metacompiler.

Natural order of definitions

The definitions in the target code appear in the natural forth order, a blue word must already exist in the image before it can be compiled in a colon definition. We made this choice because consequently no registration of addresses that must be filled afterward is needed. This can be problematic for defining words, but we solve that by giving `DOES`-parts a name.

Doers

In the target code the named `DOES`-part, *the doer*, is defined apart from and long before the `CREATE`-part of the defining word.

`:DOER ccc`

defines a doer in high level forth, replaces `DOES>`

`CODEDOER ccc`

defines a doer in assembler, replaces `;CODE`

```
:DOER DOCON     @ ;
CODEDOER DOCOL  ip push
                w ip mov NEXT end-code

20 CONSTANT bl
: SPACE         BL EMIT ;

\ And later on in the target code
: CONSTANT      CREATE DOCON , ;
: :             CREATE DOCOL ... ;
```

Figure 3: Doer

As soon as `DOCOL` and `DOCON` do exist in the image, the metacompiler is able to build colon definitions and constants in the image.

A doer is a data word. The doer body contains the `DOES`-routine (the "data"). A doer (when executed) puts its body address in the CFA of the newest word. This method makes it possible to metacompile noForth with only one or two forward references, not regarding the jump forward within colon definitions (`IF WHILE ELSE`).

A side effect is that a noForth decompiler easily detects word types.

The metacompiler is completed before metacompiling starts

Nothing is added to the metacompiler during the metacompiling process, the knowledge in the red meta words

[4] In WINTERPRET two words are special: xCOMPILE, compiles only the „blue“ words and EXECUTE executes only the „red“ ones.

[5] Again, we strongly recomend looking things up on our internet site. There it is printed in colors for ease of understanding.

and the possibility to look up things in the image, must be enough.[5]

There is no confusing intermingling of host compiling and target compiling.

It means too that we can put the noForth image in the dictionary space of the host forth (The host `C,` `HERE` and `ALLOT` can be used for image building).

Late binding

At the moment that `xCONSTANT` is defined as a red word in the meta code, the address of `DOCON` is not known. We put `"DOCON"` as a string in the `xCONSTANT` definition, not its address.

```
: xCONSTANT xCREATE DOCON x, ;
```

becomes something like:

```
: xCONSTANT xCREATE xDOER" DOCON" x, ;
```

Every time the red `CONSTANT` is executed `xDOER" DOCON"` must look up `DOCON` in the image. Thus it happens in all red defining words and also in the compiling red words as

```
LITERAL ." S" DO ?DO LOOP +LOOP POSTPONE ?ABORT
```

At compile time the words they compile must be searched in the image.

Two reusable labels

`AMTSTERDAM` and `ROTTERDAM` are two labels. They can be used again and again. With them we can jump to code fragments that we want to reuse.

`LABEL-AMSTERDAM` puts the address where we are into the label, `AMSTERDAM` puts that address on the stack. The same with `ROTTERDAM`.

```
code TRUE
  tos sp -) mov
LABEL-AMSTERDAM
  #-1 tos mov NEXT
end-code

code FALSE
  tos sp -) mov
LABEL-ROTTERDAM
  #0 tos mov NEXT
end-code

code =
  sp )+ tos cmp
  =? ROTTERDAM label-until,
  AMSTERDAM jmp
end-code

code MIN
  sp )+ w mov tos w cmp
LABEL-AMSTERDAM
  >? if, w tos mov then, NEXT
end-code

code MAX
  sp )+ w mov w tos cmp
  AMSTERDAM jmp
end-code
```

A closer look at "Image building"

The metacompiler is able to show how the image grows. The word `TRACE` activates and `NOTRACE` deactivates this function. Put these words around a few definitions in the target code and you can study in detail how the image is being built. This was our debugger. Use the space bar for wait/continue. Three examples:

1. Example

```
trace
forth: : .S ( -- )
?stack (.) space
depth false
?do depth i - 1- pick
base @ 0A = if . else u. then
loop ;
notrace
```

The output:

```
E55A forth: ( )
E55A : ( )
E55A <<<<< .S >>>>>
E55A E49E , E55A 0208 ! ( )
E55C 81 c, 82 c, ".S" m, DOCOL C176 , ( 44 )
E562 ( ( 44 )
E562 ?stack D1F4 , ( 44 )
E564 (.) D58A , ( 44 )
E566 space CF8E , ( 44 )
E568 depth D2C8 , ( 44 )
E56A false C7C2 , ( 44 )
E56C ?do ?DO( C2C4 , 0000 , ( 44 E56E 33 )
E570 depth D2C8 , ( 44 E56E 33 )
E572 i C326 , ( 44 E56E 33 )
E574 - C9E0 , ( 44 E56E 33 )
E576 1- C9A4 , ( 44 E56E 33 )
E578 pick C738 , ( 44 E56E 33 )
E57A base C438 , ( 44 E56E 33 )
E57C @ C578 , ( 44 E56E 33 )
E57E 0A 0015 , ( 44 E56E 33 )
E580 = C7D4 , ( 44 E56E 33 )
E582 if 8FFF , ( 44 E56E 33 E582 11 )
E584 . D104 , ( 44 E56E 33 E582 11 )
E586 else 7FFF , 8805 E582 !
     ( 44 E56E 33 E586 11 )
E588 u. D0F6 , ( 44 E56E 33 E586 11 )
E58A then 7803 E586 ! ( 44 E56E 33 )
E58A loop LOOP) C300 , E58C E56E ! ( 44 )
E58C ; EXIT C102 , ( )
E58E notrace
```

2. Example

```
trace
forth: 20 constant BL
notrace
```

The output:

```
C342 forth: ( )
C342 20 ( 20 )
C342 constant
( 20 ) ( 20 )
C342 <<<<< BL >>>>>
C342 C320 , C342 0200 ! ( 20 )
C344 81 c, 82 c, "BL" m,
     DOCON C19E , 0020 , ( )
C34C notrace
```

3. Example

```
trace
forth: code EXIT
LABEL-AMSTERDAM rp )+ ip mov NEXT
end-code
extra: code ?EXIT ( flag -- )
#0 tos cmp sp )+ tos mov =?
AMSTERDAM label-until,
NEXT end-code
notrace
```

The output:

```
C0FA forth: ( )
C0FA code ( )
C0FA <<<<< EXIT >>>>>
C0FA 0000 , C0FA 0202 ! ( )
C0FC 81 c, 84 c, "EXIT" m, C104 , ( 55 )
C104 LABEL-AMSTERDAM ( 55 )
C104 rp ( 55 1 )
C104 )+ ( 55 1 -3 )
C104 ip ( 55 1 -3 5 )
C104 mov 4135 , ( 55 )
C106 NEXT 4F00 , ( 55 )
C108 end-code ( )
C108 extra: ( )
C108 code ( )
C108 <<<<< ?EXIT >>>>>
C108 C0E6 , C108 0204 ! ( )
C10A 83 c, 85 c, "?EXIT" m, FF c,
     C114 , ( 55 )
C114 ( ( 55 )
C114 #0 ( 55 3 )
C114 tos ( 55 3 7 )
C114 cmp 9307 , ( 55 )
C116 sp ( 55 4 )
C116 )+ ( 55 4 -3 )
C116 tos ( 55 4 -3 7 )
C116 mov 4437 , ( 55 )
C118 =? ( 55 2000 )
C118 AMSTERDAM ( 55 2000 C104 )
C118 label-until, 23F5 , ( 55 )
C11A NEXT 4F00 , ( 55 )
C11C end-code ( )
C11C notrace
*
```

Link

Please check this site:
http://home.hccnet.nl/anij/nof/how%20noforth%20is%20made.pdf

Its HTML and prints in color. You will get an easy understanding of the metacompiler words there. This black and white paper print is to spread the news.

Recognizer – Interpreter dynamisch verändern

Matthias Trute

Der Titel ist nicht neu[1],[2]. Neu ist, dass das titelgebende Wort das Fegefeuer der Forth–Wächter durchlaufen hat und der Forth–Standard offener geworden ist, ohne an Verbindlichkeit einzubüßen.

Fegefeuer?

Die Idee der Recognizer ist, dass sie den Forth–Interpreter erweitern. Der wird damit deutlich flexibler und schließt zum Forth–Compiler auf, der das schon immer war. Man denke an `cs-pick` und `cs-roll`.

Die Wächter über den Forth–Standard haben in den regelmäßigen Treffen Recognizer praktisch jedes Jahr neu bewertet, insgesamt vier mal. Waren es am Anfang vor allem Erklärungen und Erläuterungen, was das alles überhaupt soll, war es dieses Jahr anders. Es wurde gestrichen und vereinfacht und ein interessantes englisches Wort tauchte auf: Bikeshedding. Dabei haben die Wächter so ganz nebenbei etwas ermöglicht, was sie selbst als unnötig eingeschätzt haben, nämlich etwas zu standardisieren, was nicht bereits jeder benutzt. Dafür wurde eine eigene Kategorie erschaffen: Committee supported proposal. Das sind Kapitel, die in den Standard kommen können, wenn sie denn hinreichend viele nutzen.

Was blieb übrig?

Betrachtet man die 4 RFD–Texte mit etwas Abstand, merkt man, dass sie anfänglich sehr speziell formuliert waren. Danach folgten zwei Fassungen mit umfangreichen und detaillierten Erklärungen. Die jetzt vorliegende vierte Fassung ist vergleichsweise spartanisch geraten. Die zwei Zwischenfassungen beschrieben, wie sich Recognizer auf eine Vielzahl von bestehenden Forth–Worten auswirken. Das war für die Meinungsbildung zweifellos wichtig, der Übersichtlichkeit und Verständlichkeit aber sicher nicht zuträglich.

Am Ende blieben genau 7 Worte.

`RECOGNIZE` ist das Kernstück geworden. Es macht die ganze Arbeit. `FORTH-RECOGNIZER` liefert die Schnittstelle zum Forthsystem, damit Worte wie `EVALUATE` und `LOAD` ein konsistentes Verhalten hinbekommen können. Daneben sind `RECTYPE:` und `RECTYPE-NULL` wichtig. Ersterer erschafft neue Datentypen und letzterer ist das Nullelement in dieser Welt, Mathematiker werden jetzt lächeln. Die verbleibenden drei Worte sind für Systemimplementierer.

Die Evolution des Nullelements `RECTYPE-NULL` ist noch erwähnenswert. Anfangs symbolisierte es den Fehler, dass der String nicht erkannt wurde. Daher der Name `R:FAIL`. Das hat sich als zu kurzsichtig herausgestellt. Denn es ist ja kein Fehler, sondern nur die einfache Information *hat nicht geklappt*. Solche Flags werden normalerweise mit einer 0 kommuniziert (oder eben -1 für TRUE). Da bei den Recognizern die „echte“ Null nicht optimal nutzbar ist, wurde eine eigene Null geschaffen. Das ist eine Konstante, die nicht nur einfach eine Zahl ist, sondern innerhalb der Recognizerstrukturen ein voll einsetzbares Element für einen Datentyp. Die Aktionen dieses Datentyps sind eher destruktiv und normalerweise mit den Fehlerroutinen des Systems verbunden. Damit kommt man ohne zusätzliche Checks aus und hat einen einfachen und klaren Programmcode. Details sind im RFD in epischer Länge enthalten.

Inhaltlich gab es über die Zeit nur eine wirkliche Änderung. Anfangs gab es einen globalen Recognizerstack, der implizit in allen relevanten Worten genutzt werden sollte, den gibt es jetzt nicht mehr. Die Idee dahinter war in Anlehnung an den ORDER Stack, der genauso gestrickt ist. Jetzt muss man immer angeben, welcher Stack genutzt werden soll, ein Pendant im bisherigen Forth gibt es da nicht. Im Zweifel kommt man mit folgender Sequenz ins Ziel:

```
... ( addr len -- )
 FORTH-RECOGNIZER RECOGNIZE
( -- data rec-type|rec-null )
...
```

Der Interpreter aus [2] hat sich daher kaum geändert. Er ist jetzt komplett mit Standardworten umsetzbar.

```
: interpret
  begin
    parse-name
  ?dup while
    FORTH-RECOGNIZER RECOGNIZE
    state @ if RECTYPE>COMP else RECTYPE>INT then
    execute ?stack
  until drop ;
```

Der Gleitkommazahlenrecognizer, der ganz am Anfang dieser Geschichte stand, sieht jetzt so aus

```
' noop              \ interpret
:noname fliteral ;  \ compile
:noname fs. -48 throw ; \ no real postpone
RECTYPE: rectype-float

: rec-float >float
   if rectype-float else RECTYPE-NULL then
;
```

Entfallenes

In den ersten Versionen gab es noch eine Reihe von Worten, die den Recognizer–Stack verwaltet haben. Die sind komplett entfallen. Warum ist das eine gute Idee?

Im Laufe der Diskussionen kam mehr als einmal die Ähnlichkeit zwischen Recognizern und dem Search–Order Stack zur Sprache. Eine Idee war, dass man die beiden

einfach vereinigt. Das hat nicht allen gefallen, mir auch nicht. Damit hat man zwei Stacks, die sich ergänzen, aber nicht ersetzen. Da kommt natürlich die Frage auf, warum die beiden Stacks unterschiedlich behandeln? Ist es nicht vielmehr einfacher, einen eigenen Datentyp `STACK` einzuführen, der dann beide Stacks gleichermaßen abdeckt? Systemimplementierer werden das ganz freiwillig ohnehin machen. Am Ende hat Salomon gesprochen und den Auftrag erteilt, einen Entwurf nur für Stacks zu erstellen. Damit reicht es im Recognizer–RFD aus, zu erwähnen, dass Recognizer in einem Stack liegen, wie der entsteht und verwaltet wird, ist nicht mehr relevant. Dass es den Stack–RFD noch nicht gibt, ist nicht so wichtig, Vorarbeiten sind auf theforth.net schon gemacht, auch der VD–Artikel[8] liefert einige Eckpunkte. Weniger wird's bestimmt nicht werden.

Zukünftiges

Recognizer sind einen großen Schritt auf dem Standardisierungsweg vorangekommen. Es gibt eine Vielzahl von Beispielen, was man alles damit anfangen kann. Einige Beispielrecognizer in dieser Richtung sind als Anregung auf Geralds Seite theforth.net zu finden. Da gibt es auch weniger einfache Dinge wie *literate programming* und mehrzeilige Kommentare. Morsezeichen zu erkennen, sei dem geneigten Leser überlassen.

An gleicher Stelle ([6]) ist auch das Grundgerüst für eigene Implementierungen. Das ist aus Gründen der Verständlichkeit nicht allzu optimiert, kann aber zumindest mit vielen Testcases beim Entwickeln weiterhelfen. AmForth und gforth sind auf der Höhe der Zeit. Fragt euren Forth–Provider danach!

Daneben hat der Forth–Wächterrat Mittel und Wege erschaffen, neue Ideen zu integrieren: Committee Supported Proposals. Die Recognizer sind das Erste dieser Art. Ich halte das für fast wichtiger, da es die Hürde für Innovation absenkt, da nicht mehr nur Etabliertes und bereits von allen Genutztes das Forum des Standard–Gremiums haben kann. Das Feedback bei den Recognizern war über die Jahre hinweg durchweg positiv und konstruktiv. Wer einen oder sogar mehrere Advokatus Diaboli gefunden hat, darf sich glücklich schätzen, die Arbeit macht sich keiner, dem das Thema egal ist. Das ist nur manchmal schwer.

Ach ja: Bikeshedding. *Native Speaker* zucken schon mal mit den Schultern. Hier die Auflösung: Jeder hat schon von Murphy's Gesetz gehört, da geht es um herunter fallende Marmeladenbrötchen. Daneben gab es etwa zur gleichen Zeit den Herrn Parkinson, der ein paar Regelmäßigkeiten im professionell–zwischenmenschlichen Bereich gefunden hat. Bikeshedding bezieht sich auf das Gesetz der Trivialität: „The time spent on any item of the agenda will be in inverse proportion to the sum involved." Dabei wird meist der Bau eines Fahrradunterstands an einem zu errichtenden Atomkraftwerk herangezogen. Während die Kosten des Letzteren nicht weiter thematisiert werden, sind die Kosten für Ersteren in allen Projektmeetings das bestimmende Thema. Bei den Recognizern waren es die Namen der Worte. Bikeshedding bei den Recognizer–Diskussionen ist jedoch unfair, es gibt nichts Wichtigeres als die Namen für Worte. Notation matters.

Referenzen

[1] Matthias Trute, *Recognizer – Interpreter dynamisch verändern*, Vierte Dimension 2011–02

[2] Matthias Trute, *Recognizer – Interpreter dynamisch verändern*, Vierte Dimension 2015–02

[3] Bernd Paysan, *Recognizer*, Vierte Dimension 2012–02

[4] Matthias Trute, „Forth Recognizer — Request For Comments" `http://amforth.sourceforge.net/Recognizer.html`

[5] gforth GIT Repository `http://git.savannah.gnu.org/cgit/gforth.git`

[6] Recognizer at TheForth.net `https://theforth.net/package/recognizers`

[7] Notizen der Forth 200x Sitzung 2017 `http://www.forth200x.org/meetings/minutes2017.pdf`

[8] Matthias Trute, *Stacks für Forth*, Vierte Dimension 2016–04

EuroForth 2017 in Bad Vöslau

Bernd Paysan

Die EuroForth 2017 fand in Bad Vöslau statt, im gleichen Konferenzhotel, in dem 2014 schon die Forth–Tagung stattfand; wie immer im Anschluss an 2 Tage Forth–200x–Standardisierung. Alle Vorträge kann man auf Video ansehen [1] und von vielen die Proceedings lesen [2]. Die Fotos werden in Kürze auf der neuen Forth–eV–Website zu sehen sein.

Aber natürlich ist es spannender, wenn man dabei ist. Die EuroForth ist wegen des vorgeschalteten Standard–Meetings immer etwas anstrengend, und erschwerend kommt noch hinzu, dass es immer mehr Speaker gibt, und die wollen natürlich alle vortragen und gehört werden. Entsprechend lang ist das Programm, entsprechend viele Sprecher melden sich zu Wort. Auch die offene Diskussion am Ende wird gut angenommen, auch mit kurzen Wortmeldungen.

Diese Zusammenfassung soll daher nur ein Teaser auf die Videos sein, die selbstverständlich alle auf unserer Website zu finden sind *[1]*.

Freitag, 8. September

1. 14:00 Bernd Paysan: *MINOΣ2 — a GUI for net2o*

   Als Warmup diente mein Vortrag über MINOΣ2, das einen etwas anderen Schwerpunkt hat als das 20 Jahre alte MINOΣ, nämlich als Browser–Engine für net2o dienen soll. Entsprechend ist der komplette Vortrag mit dem Projekt selbst gerendert, und nicht mit LaTeX, wie bei mir sonst üblich.

   Als Bonus Slide (ein von Fefe übernommenes Konzept) ist noch ein Kurzvortrag über das heiße Hype–Thema des Kryptoherbsts 2017 drin, nämlich wie man die BlockChain richtig macht.

2. 14:40 Andrew Haley: *A multi-tasking wordset for Standard Forth*

   Multitasking gibt es in Forth seit fast 50 Jahren, und Andrew hat mal zusammengefasst, welche Konzepte dafür nötig sind, damit man das endlich mal standardisieren kann. Erstaunlicherweise funktioniert der anfänglich mal kooperative Forth–Multitasker nämlich auch problemlos auf modernen Multi–Core–Systemen, weil eigentlich schon alles da war.

3. 15:50 Andrew Read: *Forth: a new synthesis*

   Der zweite Andrew vergleicht Forth vom Konzept her mit einer biologischen Zelle. Denn ähnlich wie Forth sind Zellen aus gut faktorisierten Einzelteilen aufgebaut, und gibt es eine Kernsemantik, an denen nachfolgende Generationen nichts mehr verändern, etwa der DNA–Codierung (da gibt es nur bei einigen Bakterien leichte Abweichungen). Ähnlich stabil ist die Kernsemantik von Forth.

4. 16:30 Ulrich Hoffmann: *A Recognizer Influenced Handler Based Outer Interpreter Structure*

   Ulrich Hoffmann stellt einen „einfacheren“ Recognizer vor: Einen, bei dem es zwar Handler gibt, aber keine Übergabe von Tabellen und Werten auf dem Stack. Allerdings gibt es halt auch Gründe für den komplexen Recognizer–Ansatz.

5. 17:30 Howerd Oakford: *cryptoColorForth*

   Howerd Oakford hat sich den (Assembler–)Quelltext von ColorForth angeguckt, und es zu entschlüsseln versucht. Denn der Assemler–Quelltext ist nicht die eigentliche Quelle; vermutlich ist die aber noch weniger dokumentiert.

6. 18:20 Stephen Pelc: *Special Words in Forth*

   Nach längerer Diskussion mit Anton ist Stephen Pelc jetzt doch der Meinung, dass man special compilation Semantics nicht mit COMPILE, ausführt, sondern was neues braucht. SET–NDCS ist sein neuer Ansatz.

Samstag, 9. September

1. 9:00 Bill Stoddart: *Halting Misconcieved?*

   Das Halte–Problem hat Bill schon letztes Jahr beschäftigt. Meine Anmerkung in der Diskussion nach dem Vortrag, dass das Halte–Problem nur für endlose Bänder unlösbar ist, und für endliche Bänder eine „triviale“ (im mathematischen Sinn, versteht sich) Lösung hat, hat ihn zu mehr Recherche gebracht: Auch andere theoretische Informatiker haben schon Minderheiten–Meinungen, wonach das eigentlich kein Problem ist, wenn man den kaputten Selbstbezug entfernt.

2. 9:45 M. Anton Ertl: *SIMD and Vectors*

   Moderne CPUs haben SIMD, und die richtige Abstraktion dafür sind Vektor–Operationen, also in Forth ein Vektor–Stack. Anton Ertl präsentierte den aktuellen Stand seiner Forschung.

3. 10:40 Klaus Schleisiek: *Compilation on Demand*

   Auf ganz kleinen Systemen will man wirklich nur das haben, was unbedingt drauf muss, also selbst so Hilfswörter wie –ROT nur bei Bedarf. Man will aber nicht jedes Wort in eine Datei auslagern, sondern seine Library ganz normal 'reinladen: Sie soll dann aber nur das compilieren, was später (!) gebraucht wird. Klaus Schleisiek zeigt, wie er das macht.

4. 11:10 Sergey Baranov: *A Formal Language Processor Implemented in Forth*

Formale Sprachen beackert man mit Parser–Generatoren. Sergey Baranov zeigt, wie er das mit Forth löst.

5. 11:30 Rick Carlino: *Farmbot*

   Farmville zu langweilig, weil man die angepflanzten Produkte nicht essen kann? Selber in RL was anbauen zu dreckig und schweißtreibend? Der (natürlich in Forth programmierte) Farmbot ist die Lösung: Lasse einfach einen Roboter das Anbauen erledigen.

6. 11:40 Gerald Wodni: *F — news from the package manager*

   Gerald Wodni erklärt, wie man Packages für F, das Forth–Repository, baut. Damit es zum am schnellsten wachsenden Repository werden kann (aufgrund der bisher kleinen Anzahl an Paketen ist das gut machbar).

Traditionell ist die Exkursion am Nachmittag, die uns zunächst nach Schloss Schönbrunn führte, wo Gerald Wodnis Schwester, die dort eine Lizenz für eine professionelle Führung hat, uns alles zu Maria Theresias und Franz–Josef erklärte. Danach ging es durch die Innenstadt, zum Prater ins Riesenrad und Abendessen dann dort im Biergarten; dieses Mal nicht Schweizerhaus, sondern Luftburg, das ist gleich daneben.

Sontag, 10. September

1. 9:20 Nick Nelson: *Dynamic Integration of Forth, GTK+ and Glade*

   Nick erklärt, wie man aus einem Glade–Projekt die Nummern und die Namen zu den Objekten herausfieselt, damit man das dann elegant programmieren kann.

2. 9:35 Bob Armstrong: *CoSy: Iverson's APL alive in Moore's Forth*

   APL ist auch eine eher unbekannte Sprache. Bob Armstrong stellt ein APL vor, das in Forth implementiert ist.

3. 11:00 Paul E. Bennett: *Open discussion*

   Wie bekommt man Nachwuchs für Forth?

4. 11:20 Klaus Schleisiek: *Use the 9th bit of FPGA block RAM*

   Im FPGA gibt es ein neuntes Bit im RAM–Block. Was kann man damit in Microcore anstellen? Klaus nutzt es, um Calls und Returns zu beschleunigen.

5. 11:30 Bill Stoddard: *A new test construct using reversible computation*

   Bills reversibles Forth bekommt einen Input–Generator und einen Output–Test.

6. 11:40 Anton Ertl: *The new Gforth header*

   Anton erklärt, wie der neue (eigentlich schon 5 Jahre alte) Gforth–Header aussieht: Mit Methoden für die verschiedenen Aktionen, die Forth mit einem Wort macht.

7. 11:50 Leon Wagner: *Satellite tracking project*

   Forth fing mal an mit der Steuerung einer großen Radioastronomie–Schüssel. Wie man damit heute Satelliten trackt, zeigt Leon Wagner.

8. 12:00 Sergey Baranov: *Publish on IEEE*

   Sollen/können/wollen wir unsere Proceedings über die IEEE publizieren? Antwort: Fragen kostet nichts.

9. 12:05 Ulrich Hoffmann: *Literate Programming with Markdown*

   Früher hat man mit LaTeX Literate Programming gemacht. Wie man das heute mit Markdown macht, zeigt Ulrich Hoffmann.

10. 12:15 Howerd Oakford: *More on ColorForth*

    Howerd zeigt noch etwas mehr Entschlüsselung des ColorForth–Quellcodes.

11. 12:20 Stephen Pelc: *SWD file to Forth*

    Auf ARM sind die Header–Files in einem XML–Format names SWD abgelegt. Die kann man gleich direkt mit Forth bearbeiten, statt den Umweg über C zu gehen.

12. 12:25 Paul E. Bennet: *EuroForth 2018 in Edinburgh, Scottland*

    Die nächste EuroForth findet 2018 in Edinburgh statt, und Chuck Moore ist geladener Gast — 50 Jahre Forth werden gefeiert.

Referenzen

[1] `https://wiki.forth-ev.de/doku.php/events:euroforth-2017`

[2] `http://www.complang.tuwien.ac.at/anton/euroforth/ef17/genproceedings/papers/`

Abbildung 1: Gruppenfoto vor Schloss Schönbrunn

Abbildung 2: Am Prater

Forth–Gruppen regional

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

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

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

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

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

Gruppengründungen, Kontakte

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

μP–Controller Verleih

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

Spezielle Fachgebiete

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

Termine

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

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

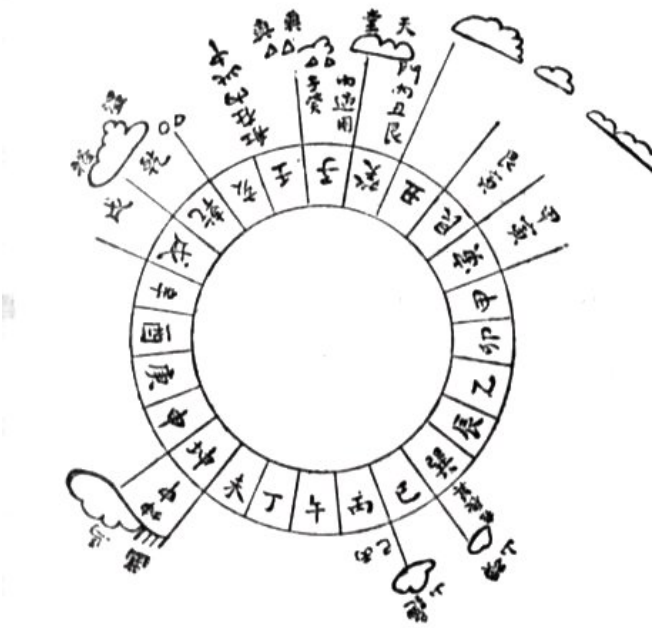
10.–11.03.2018 Maker Faire Ruhr in Dortmund
https://www.makerfaire-ruhr.com

05.–08.04.2018 Forth–Tagung in Essen–Horst
https://tagung.forth-ev.de

14.–16.09.2018 Maker Faire Hannover
http://www.makerfairehannover.com

September 2018? EuroForth
http://www.euroforth.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.

Einladung zur

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

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

Anreise

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

Anmeldung

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

Programm

Donnerstag

ab 14:00	Frühankommer Workshops

Freitag

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

Samstag

vormittags	Ausflug
nachmittags	Vorträge und Workshops

Sonntag

09:00	Mitgliederversammlung
12:00	Ende der Tagung