Vierte Dimension

4/2012

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

tematik GmbH
Technische Informatik

Feldstrasse 143
D–22880 Wedel
Fon 04103 – 808989 – 0
Fax 04103 – 808989 – 9
mail@tematik.de
www.tematik.de

Gegründet 1985 als Partnerinstitut der FH–Wedel beschäftigten wir uns in den letzten Jahren vorwiegend mit Industrieelektronik und Präzisionsmeßtechnik und bauen z. Z. eine eigene Produktpalette auf.

Know–How Schwerpunkte liegen in den Bereichen Industriewaagen SWA & SWW, Differential–Dosierwaagen, DMS–Messverstärker, 68000 und 68HC11 Prozessoren, Sigma–Delta A/D. Wir programmieren in Pascal, C und Forth auf SwiftX86k und seit kurzem mit Holon11 und MPE IRTC für Amtel AVR.

LEGO RCX–Verleih

Seit unserem Gewinn (VD 1/2001 S.30) verfügt unsere Schule über so ausreichend viele RCX–Komponenten, dass ich meine privat eingebrachten Dinge nun Anderen, vorzugsweise Mitgliedern der Forth–Gesellschaft e. V., zur Verfügung stellen kann.

Angeboten wird: Ein komplettes LEGO–RCX–Set, so wie es für ca. 230,–€ im Handel zu erwerben ist.

Inhalt:

1 RCX, 1 Sendeturm, 2 Motoren, 4 Sensoren und ca. 1.000 LEGO Steine.

Anfragen bitte an
Martin.Bitter@t-online.de

Letztlich enthält das Ganze auch nicht mehr als einen Mikrocontroller der Familie H8/300 von Hitachi, ein paar Treiber und etwas Peripherie. Zudem: dieses Teil ist „narrensicher“!

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

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

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

KIMA Echtzeitsysteme GmbH

Tel.: 02461/690–380
Fax: 02461/690–387 oder –100
Karl–Heinz–Beckurts–Str. 13
52428 Jülich

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

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

Bergstraße 10	D–18057 Rostock
Tel.: +49 381 496800-0	Fax: +49 381 496800-29

PC–basierte Forth–Entwicklungswerkzeuge, comFORTH für Windows und eingebettete und verteilte Systeme. Softwareentwicklung für Windows und Mikrocontroller mit Forth, C/C++, Delphi und Basic. Entwicklung von Gerätetreibern und Kommunikationssoftware für Windows 3.1, Windows95 und WindowsNT. Beratung zu Software–/Systementwurf. Mehr als 15 Jahre Erfahrung.

Hier könnte Ihre Anzeige stehen!

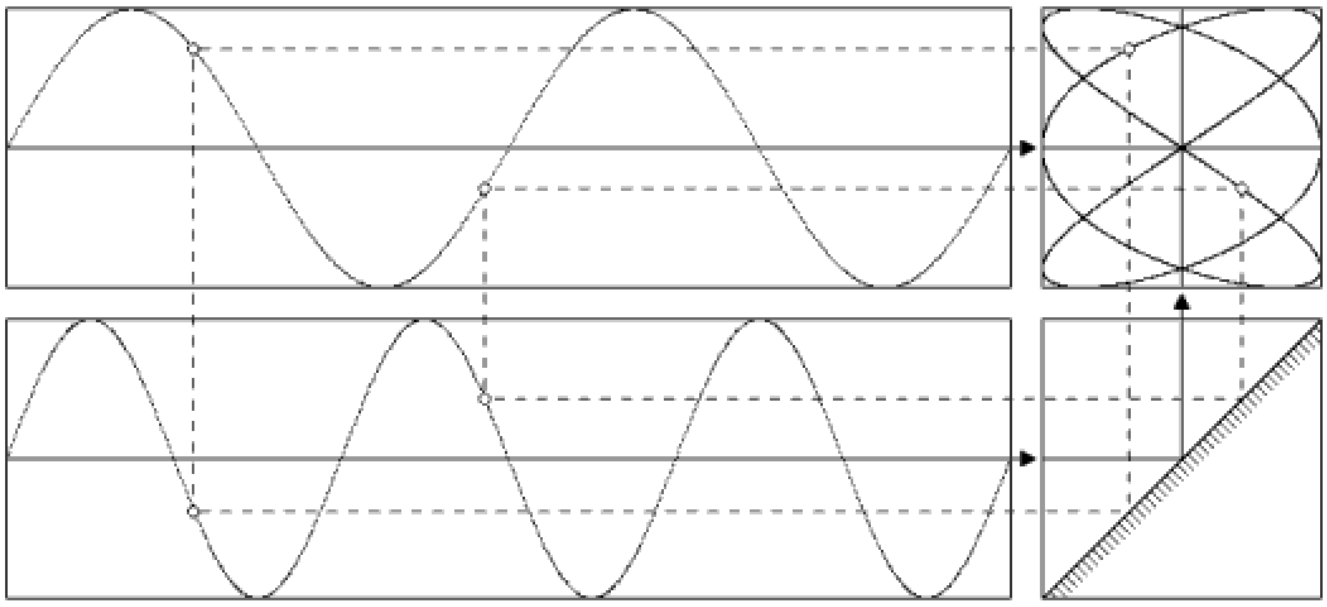
Wenn Sie ein Förderer der Forth–Gesellschaft e.V. sind oder werden möchten, sprechen Sie mit dem Forth–Büro über die Konditionen einer festen Anzeige.

`Secretary@forth-ev.de`

Ingenieurbüro
Klaus Kohl–Schöpe

Tel.: 07044/908789
Buchenweg 11
D–71299 Wimsheim

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



Zum Artikel von Hannes Teich, Wave Engine (4): Lissajous–Darstellung einer reinen Quinte (Schwingungsverhältnis 2:3). Eine temperierte Quinte würde das Lissajous–Quadrat schwärzen.

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,

Martin Bitter hat nach langjähriger Abstinenz die Ausgabe 3/2012 der VD editiert und dabei sehr erfolgreich das Zeitalter des digitalen Heftemachens erreicht. In diesem Zusammenhang ist ein Stück Dokumentation entstanden, wie denn so ein Heft ganz konkret *editiert* und *formatiert* wird. Wer also einen Artikel für die VD schreiben möchte und dabei schon vorher sehen will, wie das mit dem Layout wird, der kann den Anweisungung auf [1] folgen. Damit sollte es möglich sein, eine funktionierende Arbeitsumgebung zu erstellen. Dann macht das Textschreiben noch viel mehr Spaß und der Editor hat viel weniger Arbeit und viel mehr Spaß. Damit wäre nebenbei bewiesen, dass das Editor–Sein auch für Anfänger beherrschbar ist. Wer sich daran versuchen möchte, der meldet sich bei der Redaktion. Wir versprechen, dass wir weiterhin mithelfen.

An Ideen für Artikel mangelt es nicht. Allerdings formieren sich die Buchstaben für den fertigen Text im Heft bekanntlich nicht von alleine. Also, flugs den Füller oder die Tastatur zur Hand nehmen und die Ideen auftüpfeln. Nein, nicht unbedingt gleich sofort den ausgefeilten Text. Wie wär's mal mit einer Stichwort-/Halbsatz-Sammlung zum Aufwärmen?

Man lernt beim Heft editieren auch allerhand Neues und Erstaunliches. Beispielsweise, dass der Sämann Arepo mühsam am Rad dreht — ja, das ist jetzt die lose Übersetzung — aber das lateinische Original hat erstaunliche Eigenschaften, die man in Wikipedia nachlesen kann [2]. Selbstverständlich lernt man beim Lesen auch eine Menge: vom funkenden Füllstandsensor bis zur Neuerfindung des Internets bietet dieses Heft eine vielfältige Sammlung von Wissenswertem. Ich hoffe, dass alle Leserinnen und Leser etwas Neues oder Erstaunliches darin finden. Neu für mich ist die Tatsache, dass das Chinesische Neujahr am zweiten Neumond nach der Wintersonnenwende stattfindet [3]. Also endet am 9.2.2013 das Jahr des Drachen und der übergibt die Führung der Zeit an die Schlange.

Bald danach endet auch mein Jahr als Drachenhüter. Ich habe keine Ahnung, ob ich dem Drachen genügend forthiges Drachenfutter beschaffen konnte. Die Getreuen des Forth–Drachen sind wenige und weit versprengt. Nur durch das beharrliche Lehren der Kunst wird die Schar die Zeit überdauern und das nächste Drachenjahr (2024) in guter Verfassung begrüßen. Welche Farbe bekommt der Drache auf dem T–Shirt dann?

Wer einen *richtigen* Forth–Rechner haben will, der darf sich einmal auf der Webseite vom HIVE [4] umsehen. Drei Propeller–Chips von Parallax werden zu einem Rechner mit 24 Rechenkernen zusammengefügt und mit spin, assembler, forth oder m oder auch Basic betrieben. Komplett mit Bildschirm (VGA), Tastatur und Mouse (PS2).

Viel Vergnügen beim Lesen dieser Ausgabe wünscht

Erich Wälde

1. `http://fossil.forth-ev.de/vd-template/index`
2. `http://de.wikipedia.org/wiki/Sator_arepo_tenet_opera_rotas`
3. `http://de.wikipedia.org/wiki/Chinesische_Astrologie`
4. `http://hive-project.de`

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-2012-04`

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

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

Forthbildung auf der UnFUCK 2012

AmForth [1] war im Oktober auf dem *Unix Friends and Users Campus Kamp* [2] an der Hochschule Furtwangen vertreten, am Freitag Abend mit einem Vortrag, am Samstag Mittag mit einem Workshop. Die UnFUCK ist eine kleine, aber feine Veranstaltung. Es waren schätzungsweise etwa 50 Personen anwesend, die Veranstaltung wurde komplett per stream ins Wilde Weite Netz übertragen. Nach Aussage der Veranstalter waren am Freitag etwa 200, am Samstag knapp 300 Verbindungen angemeldet.

Der Workshop war wohl zu früh am Tag, es fanden sich um 11 h lediglich vier Besucher ein. Diese waren aber sehr interessiert und haben viele Fragen gestellt. So bin ich mit dem Stoff gar nicht durchgekommen. Aber das ist ja auch nicht wichtig. Ich kann nur hoffen, dass wenigstens einer der Teilnehmer mit dem Thema AmForth auf einem ATmega–Kontroller weiterhin beschäftigt.

Außer Wertung nahm ich am Sonntag am *Nerd Pissing Contest* — einem Programmierwettstreit [3] — teil. Vier Programme mussten bearbeitet werden, wer keine Lösung hatte, gab ein "Hello World"–Programm ab, damit die Technik weiterlief. Es fanden sich Probanden, die die Sprachen C++, Java, PHP, perl, bash, Python und eben Forth vertraten, das Ganze war ein Riesenspaß. Der Moderator und das Publikum konnten den Probanden beim Programmieren über die Video–Schulter schauen, was hier und da große Heiterkeit verursachte.

Die Aufgaben und Lösungen sind online noch verfügbar.

Für die Aufgabe `"PENIS" auf stdout ausgeben ;p` habe ich dort noch eine nettere Lösung gebastelt.

```
: p
  cr ." PINKE" cr ." ELFEN" cr ." NAGEN"
  cr ." IHREN" cr ." SALAT" cr
;
```

1. `amforth.sourceforge.net`
2. `www.unfuck.eu`
3. `npc.the-compiler.org`

Erich Wälde

Neue DX-Forth Versionen für MS-DOS und CP/M

Der australische Entwickler Ed[1] hat in diesem Jahr neue Versionen des DX–Forth–Systems für MS-DOS und CP/M veröffentlicht. Die DX-Forth–Systeme sind als Public–Domain–Software inklusive der Quellen auf *Ed's Forth and Utilities Page* [1] erhältlich.

DX–Forth 1.11 für CP/M ist ein Forth–83–System. Die neue Version vom August passt das DX-Forth an die CPU–Geschwindigkeiten neuerer CP/M Maschinen an (ja, es werden auch im Jahre 2012 neue Industriesteueranlagen mit Z80–CPU und CP/M hergestellt und ausgeliefert; Viren wie Stuxnet sind unter CP/M kein Problem). Auch der Screen–Editor und die interne Tool–Wortliste wurden angepasst, sowie neue Wörter implementiert.

DX–Forth 3.97 für MS-DOS wurde schon im Mai 2012 aktualisiert. Das alte BDOS–Interface aus CP/M–Tagen wurde durch ein neues DOSCALL–Interface ersetzt.

Beide Forth-Versionen besitzen für Public–Domain–Software eine gute Dokumentation und die MS-DOS–Version wird regelmäßig weiterentwickelt (2-3 neue Versionen pro Jahr).

1. `http://dxforth.webhop.org`

Carsten Strotmann

Eine dieser schönen Geschichten...

Aus comp.lang.forth gefischt von Dirk Brühl, einem Beobachter der Szene in den USA.

Dienstag, 29. November 2012 10:35:15 PM UTC-5, Elizabeth D. Rather:

Ich erinnere mich an einen Anruf, den wir bei Forth Inc. bekamen, von einer Firma, die eine sehr teure und komplizierte Ausrüstung verkaufte und wartete. Das war damals in den frühen 90igern. Das war in Forth programmiert worden von einem Kerl, der sein eigenes Forth geschrieben hatte, und damit dann diese äußerst umfangreiche Applikation. Er starb unerwartet, hinterließ den Code, aber ohne Dokumentation oder gedruckte Listings. Und sie mussten nun eine wichtige Kontrollfunktion ändern. Man bat mich zu kommen, um zu sehen, was wir machen könnten.

Es war ein natives Forth, ohne Betriebssystem. Vor dem Terminal platziert, konnte ich in 30 Minuten herausfinden, wie man das Listing druckt. Gestützt darauf, und nach einigen DUMPs und etwas Decompilieren, gelang es, deren Programm zu flicken (patch), und damit deren unmittelbare Bedürfnisse zu erfüllen, alles in allem in etwa 2 Stunden — ohne Debugger, ohne irgendeine zusätzliche Hardware, einfach nur Forth.

Jenes Forth war FORTH83–ähnlich, obschon nicht völlig Standard. Wir veranschlagten einige Mann–Monate, um es in polyFORTH umzuwandeln und zu dokumentieren. Aber sie bevorzugten, alles in C neu zu schreiben, was darin endete, dass es dann doch mehrere Jahre gedauert hat.

Cheers,
Elizabeth

Fortsetzung auf Seite 21

[1] dessen vollen Namen ich auch nach langer Internetrecherche nicht ausfindig machen konnte

Timer in AmForth

Matthias Trute

Timer auf Microcontrollern sind eine never-ending-story. Aus der neueren Geschichte seien nur Erich Wälde in der VD 2006-03 und Michael Kalus in VD 2010-01 erwähnt. Sie alle lösen die gleiche Aufgabe, wobei die Hardware vielfältig ist, genau wie die Aufgaben. Bei amforth [1] sind einige Ideen zu einer Bibliothek kondensiert, deren Einsatz illustriert werden soll.

Voraussetzungen

Atmegas haben einige Timermodule in Hardware. Die Module unterscheiden sich in vielen Dingen voneinander, eines ist jedoch allen gemein: Sie können zählen und bei Erreichen eines Vergleichswerts einen Interrupt auslösen. Daneben können sie Signale für diverse Arten von PWM (Pulse Width Modulation) generieren, was hier jedoch nicht betrachtet werden soll.

Was zählen sie? Da fangen die Unterschiede schon an. Bei einigen kann man den CPU–Takt durch einen in Grenzen anpassbaren Frequenzteiler schicken und diesen reduzierten Takt zählen lassen. Eine andere Quelle sind Impulse an festgelegten IO–Pins. Ein Modul ist darauf abgestimmt, mit einem extern angeschlossenen Uhrenquarz zusammenzuarbeiten und zählt diese 32–kHz–Frequenz.

Die „Vergleichswerte“ fallen in zwei Kategorien: Überlauf und ein definierter Wert. Der Überlauf ergibt sich daraus, dass der Zähler 8 oder 16bit breit ist. Der Überlauf ist einfach das Erreichen des Wertes Null.

Die andere Kategorie ist das Erreichen eines bestimmten Wertes. Damit lassen sich andere Intervalle als 256 bzw 65536 erzielen. Die Art und Weise, wie hier konfiguriert wird, bietet übrigens Rückschlüsse darauf, wie die Hardware das intern macht: Es wird nicht der eigentliche Zielwert eingegeben, sondern die Differenz zum normalen Überlaufwert: Wenn man alle 100 Impule einen Interrupt haben will, muss man bei einem 8bit–Zähler 156 angeben. Ein 16bit Zähler benötigt dementsprechend 65436 als Startwert. Dieser Startwert wird automatisch immer wieder übernommen, da muss man sich nicht mehr drum kümmern.

Einige Atmegas können mehrere dieser Aufgaben gleichzeitig und unabhängig verarbeiten, das muss man bei der Codegestaltung beachten.

Einsatzzweck

Bei vielen Anwendungen muss man regelmäßig wiederkehrend Aufgaben erfüllen. Der Code ist meist so ähnlich, dass er klassischerweise per copy und paste von Projekt zu Projekt wandert. Danach wird häufig der Code subtil angepasst und die (zumindest vermeintlich) verbesserte Routine wandert weiter. Sie geht jedoch niemals zurück zu den älteren Projekten. Dies zu konsolidieren, ist die vornehmste Aufgabe einer Bibliothek:

- prüfen, ob eine (kurze) Zeitspanne abgelaufen ist.
- eine bestimmte Zeitspanne warten und dann eine Aktion ausführen.
- regelmäßig eine Aktion ausführen.
- Zeitdauer für eine Aktion erfassen.

Da die Ressourcen begrenzt sind, ist es erstrebenswert, die zeitbezogenen Aufgaben auf einem einzigen Timer aufzubauen. Das lässt die anderen für weitere Dinge frei. Der jeweils benutzte Timer sollte zudem austauschbar sein, ohne allzusehr den Quelltext ändern zu müssen. Das dritte Kriterium ist Genügsamkeit. Es ist sehr praktisch, wenn auch der am schlechtesten ausgestattete Timer genutzt werden kann.

Damit hat man einen zweischichtigen Bibliotheksaufbau: in der unteren Schicht der Zugriff auf den Timer und das Bedienen einer festzulegenden Schnittstelle und darauf aufbauend dann die generischen Worte.

Die Vielfalt der Anforderungen und auch so mancher subtile Unterschied zwischen den Controllern vereitelt leider einen vollkommen idealen Ansatz: Kleine Codeanpassungen wie die Registernamen sind wohl unabwendbar[1]. Die Anpassungen sind aber dergestalt, dass sie keine funktionalen Auswirkungen haben.

Die untere Schicht erledigt das Initialisieren der Hardware und bedient die Schnittstelle. Diese ist denkbar einfach: Eine Variable, die jede Millisekunde hochzählt. Der Lesezugriff erfolgt, bedingt durch die Art, wie die AmForth VM arbeitet (siehe 2), atomar. Das heißt, man muss nicht befürchten, dass sich während des Lesens der Variablen ihr Wert ändert.

Der Einfluss, den der Interruptbetrieb des Zeitzähler verursacht, dürfte im Normalfall zu vernachlässigen sein: bei 1MHz werden ca 1% „verbraucht“, bei höheren Frequenzen entsprechend weniger.

Ein Timer ist einfach ein spezifischer Wert für einen Zeitpunkt: Entweder der Startzeitpunkt oder der Endzeitpunkt. Die Zahlenwerte werden mit dem aktuellen Wert (`@tick`) verglichen und als *unsigned numbers* verarbeitet. Bedingt durch die Zellengröße sind jegliche Intervalle auf 65,5 Sekunden begrenzt. Keine Limits gibt es bei der Anzahl der gleichzeitig aktiven Timer. Dazu gleich mehr.

Die obere Schicht der Bibliothek basiert auf diesem Millisekundentakt und bietet viele weitere Dienste an:

@tick (– n) Liest den aktuellen Zählerstand aus (Millisekunden).

[1] Wer einen Weg finden sollte, es ohne zu schaffen, ist herzlich willkommen.

expired? (t – f) Prüft, ob der Timer t abgelaufen ist, ruft PAUSE auf.

elapsed (t – n) Liefert die für den Timer t bereits aufgelaufene Zeit in ms.

after (xt n –) wartet n Millisekunden und führt dann XT aus.

ms (n –) Alternative Routine für das Standardwort MS, das PAUSE nutzt.

every (xt n –) Führt XT alle n Millisekunden aus. Das Wort hinter XT muss einen Flag liefern. Ist dieser TRUE, läuft die Schleife weiter, ist er FALSE, beendet sich die Schleife.

every-second (xt –) Wie `every`, nur dass ein Intervall von 1 Sekunde vorgegeben ist.

Bei vielen zeitbezogenen Aufgaben geht es um Warten. Da man die Zeit durchaus besser nutzen kann, kommt ein weiteres wichtiges Designelement der Bibliothek ins Spiel: Der Multitasker. Amforth nutzt den klassischen, kooperativen Multitasker, der auf dem Wort `pause` aufbaut.

Immer wenn eine Task „Zeit hat“ ruft sie `pause` auf und eine anderere Task kann die CPU nutzen. Normalerweise gilt das für IO–lastige Aufgaben wie `emit` oder `key`, hier ist das Wort `elapsed?` der Schlüssel. Wann immer geprüft wird, ob eine Zeitspanne abgelaufen ist, wird unterstellt, dass es normalerweise noch nicht geschehen ist und ein Taskwechsel sinnvoll durchführbar ist.

Ein zusätzlicher Vorteil des Multitaskers ist, dass er es erübrigt, ein eigenständiges Timermanagement zu implementieren, das vermutlich nicht weniger komplex sein dürfte. Wenn zwei Tasks sich koordinieren müssen, können sie dies mit den klassischen Techniken wie Semaphoren oder Locks machen, die, da amforth auf single–core–CPUs läuft und alle Assemblerworte wie `@` und `!` atomar sind, auch sehr einfach umzusetzen sind.

```
task1: Commandloop
task2: LED blinkt:
   begin
      200 ms led-red on
      200 ms led-red off
   again
task3: andere LED blinkt:
   begin
      300 ms led-green on
      100 ms led-green off
   again
```

Beispiele

Die „wichtigste“ Anwendung ist natürlich die Messung der Laufzeit eines Wortes. Nicht die profanen Benchmarks aus der Kategorie „mein Forth ist schneller als Deins“ sind hier natürlich im Fokus, als vielmehr eine der klassischen Profileraufgaben.

Das zweite Beispiel ist eine Uhr. Dazu wird als Background–Task eine Routine etabliert, die aus den Millisekunden des Timers die Sekunden Minuten etc. zählt. Die Genauigkeit ist natürlich nicht sonderlich hoch, bei mir kommen schon mal Abweichungen von 10–20 Sekunden pro Stunde zusammen. Eine externe RTC (Real Time Clock, z.B. am I2C Bus) kann das besser (bei geringerem Strombedarf). Ein Beispiel für einen Multitaskjob ist es allemal.

Das dritte Beispiel ist eine pragmatische Lösung für einen oft nicht reproduzierbaren Effekt: unbeabsichtigte Endlosschleifen. In der reinen Lehre meldet die Hardware jeden Fehler und das Programm kann darauf reagieren...

Die Timer benutzen eine Vielzahl von controllerspezifischen Registern. Dazu müssen deren Adressen entweder vorab definiert werden (WANT_TIMER_COUNTERx Option setzen) oder man benutzt die Amforth-Shell, die solche Konstanten selbstständig in die zugehörigen Zahlen umsetzen kann.

Generell erfüllen die hier vorgestellten Routinen die Anforderungen für kurze Aufgaben. Nicht geeignet sind sie für sehr kurze (Mikrosekunden) und sehr lange (Stunden) Aufgaben. Ebenso sind Aufgaben mit erhöhter Präzision nicht im Fokus. Wenn es aber mal eine Millisekunde mehr oder weniger sein darf, kann sie genutzt werden.

Referenzen

1. `amforth.sourceforge.net`
2. Interrupts in Amforth, VD 2011-02

Listings

Hardwarenahe Routinen

```
1   \ requires
2   \  in application master file
3   \    .set WANT_TIMER_COUNTER_0 = 1
4   \  from device.frt
5   \    TIMER0_OVFAddr
6   \ provides
7   \  timer0.tick      -- increasing ticker
8   \
9   \ older mcu's may need
10  \  TCCR0 constant TCCR0B
11  \  TIMSK constant TIMSK0
12
13  variable timer0.tick
14
15  : timer0.isr
16    1 timer0.tick +!
17  ;
18
19  \ some settings for 8bit timer to
20  \ get 1ms ticks
21  \ f_cpu  prescaler preload
22  \  16MHz   64       6
```

```
\   8MHz   64     131
: timer0.init ( preload -- )
    TCNT0 c!
    0 timer0.tick !
    ['] timer0.isr TIMER0_OVFAddr int!
;

: timer0.start
    0 timer0.tick !
    %00000011 TCCR0B c! \ prescaler 64
    %00000001 TIMSK0 c! \ enable interrupt
;

: timer0.stop
   %00000000 TCCR0B c! \ stop timer
   %00000000 TIMSK0 c! \ disable interrupt
;
```

Generische Routinen

```
\ _die_ API
: @tick
   timer0.tick @
   \ timer1.tick @
;

\ ab hier keine Anpassungen mehr
: expired? ( t -- flag )
   pause @tick - 0<
;

\ alternative implementation for ms
: ms @tick + begin dup expired? until drop ;

: elapsed ( t -- n )
    @tick swap -
;

\ execute XT nach u Milliseconden
\ ex: ' foo 10 after
: after ( xt u -- )
   ms execute
;

\ execute XT alle u ms. Das Wort hinter
\ dem XT muss den Stackeffekt ( -- f).
\ haben. Sobald f FALSE ist, wird der loop
\ beendet
: every ( xt u -- )
   begin over over after until drop drop
;

: every-second ( xt -- )
   1000 every
;
```

Einsatzbeispiel benchme

```
\ run a single word n-times and
\ print the milliseconds for the
\ whole run
\ usage
\  ' foo 10 benchme

: benchme ( xt n -- )
   dup >r
   @tick >r
    0 ?do dup execute loop drop
    r> elapsed r>
    u. ."  iterations in " u. ."  ms" cr
;
```

Einsatzbeispiel Datum und Zeit

```
\ create task space
$20 $20 0 task: t:date&time

variable seconds
\ runs every second
: job-date&time
  1 seconds +!
  \ more code for minute/hour/day...
  0 \ flag for an endless loop
;

\ set up the task
: setup-date&time
  t:date&time task-init  \ create TCB in RAM
  0 seconds !
  \ ... more code for minutes etc
  t:date&time tcb>tid activate
  \ code from here is executed later as task
  ['] job-date&time every-second
;

\ setup and start the task "date/time"
: turnkey-date&time
  \ set up multitasker
  onlytask
  6 timer0.init timer0.start  \ 16 MHz f_cpu
  \ insert task into task list
  setup-date&time t:date&time
  tcb>tid alsotask
  \ start multitasking
  multi
;
```

Einsatzbeispiel Timeout Loop

```
\ timeout-begin is a potentially endless
\ loop that terminates after a predefined
\ timeout. In case of timeout an exception
\ is thrown
variable alarmtime
: (init-alarm)
   @tick + alarmtime !
;
: (check-alarm)
   alarmtime @ expired? if -512 throw then
;

: timeout-begin
    postpone (init-alarm)
    postpone begin
    postpone (check-alarm)
; immediate

\ testcase. timeout after 100ms
: foo
  100 timeout-begin
    noop
  again
;
```

Adventures 12: Funk–Füllstandsensor für die Zisterne

Erich Wälde und Martin Bitter

Es kommt der Tag im Leben eines Kontrollettis[1]*, wo es was zu messen gibt, an einer Stelle, an der ein Kabel (Strom, Kommunikation) keine Option ist. Das ist der Tag, wo er sich vielleicht daran erinnert, dass es bei Pollin [6] so nette kleine Module mit dem Namen* rfm12 *gibt, die auf den ISM–Funkbändern senden und empfangen. Also flugs welche bestellt und bekommen. Damit geht der Ärger aber erst richtig los. Stellt sich heraus, dass die Module kleine Primadonnen sind, die richtig gestreichelt werden wollen, bevor das mit der drahtlosen Datenübertragung halbwegs funktioniert.*

Im ersten Teil [2] beleuchteten wir die Grundlagen. Im zweiten Teil [3] wurde gezeigt, wie man in der Betriebsart OOK (on–off–keying) käufliche Außensensoren für Luftfeuchte und –temperatur belauschen kann. Dieser Teil beschreibt den Bau eines Füllstandsensors für die Zisterne.

Problemstellung

Zu unserem Haus gehört eine Regenwasserzisterne. Die Kammer hat einen Durchmesser von 2.5 m, der Überlauf erlaubt einen Wasserstand von 2.3 m. Daraus ergibt sich ein Volumen von etwa 11 m^3.

Wasser wird in kleinen Mengen über eine Handpumpe entnommen, sowie in größeren Mengen über eine elektrische Pumpe zum Gießen des Gartens. Allerdings ist es sehr aufwändig, in die Zisterne zu gelangen. Die Handpumpe muss abgeschraubt werden, der Deckel mühsam mit einem Geißfuß angehoben und auf Latten verschoben werden. Dauer der Aktion: locker zwei Stunden, je nachdem, wie widerspenstig das Material ist.

Ein Füllstandsensor, der außerhalb der Zisterne abgelesen werden könnte, wäre eine feine Sache. Dann fällt die Überraschung nicht so groß aus, wenn der Füllstand unter das Ansaugrohr der Handpumpe fällt. Außerdem könnte man auch den Niederschlag messen, jedenfalls grob, denn die Größe der Dachfläche, die das Regenwasser einsammelt, ist bekannt.

Leider kommt erschwerend hinzu, dass in der Zisterne kein Strom verfügbar ist, und das auf absehbare Zeit auch so bleibt. Also muss der Sensor über einen Akku versorgt werden. Dessen Klemmenspannung soll ebenfalls gemessen und übertragen werden.

Messprinzip

Wenn man eine Weile nachdenkt, dann fallen einem etliche Möglichkeiten ein, wie der Füllstand gemessen werden kann. Ist man aber wählerisch und verschmäht bewegte Metallteile, die in der hohen Luftfeuchtigkeit sowieso nur rosten, dann wird die Auswahl geringer. Ich habe mich für einen Ultraschallsensor entschieden, der die Entfernung aus trockener Höhe bis zum Wasserspiegel misst. Um störende Reflexionen auf ein Minimum zu reduzieren, wurde der Sensor in ein 75 mm–Abwasserrohr montiert, welches senkrecht in der Zisterne angebracht ist. Wünschenswert wäre eine Auflösung von 1 mm, was einem Volumen in der Zisterne von knapp 5 Litern entspricht.

Gemessen wird die Zeit t zwischen der Aussendung eines Ultraschallpulses und dem Eintreffen des ersten Echos. Bei bekannter Schallgeschwindigkeit v kann daraus die zurückgelegte Entfernung s berechnet werden. Der Schall legt die Strecke zweimal zurück, die Entfernung $d = s/2$ vom Sensor zum Wasserspiegel ist also

$$d = 1/2 \cdot v \cdot t \tag{1}$$

Die zu messende Entfernung bewegt sich im Bereich von 0.4 m bis 2.7 m. Bei einer Schallgeschwindigkeit von 343 m/s erwartet man Zeiten von 2.4 bis 16 Millisekunden bis zum Eintreffen des Echos. Diese Zeiten sind viele Prozessorzyklen lang und sollten damit gut handhabbar sein.

Die Laufzeit wird mit Timer1 im sogenannten *input–capture–* Verfahren gemessen. Wählt man als Eingang für den Timer1 die Frequenz des Kontrollers $f_{\text{cpu}} = 11.059200\,\text{MHz}$ und einen Vorteiler von 8, dann beträgt die Zeit für eine Zählerdifferenz von N

$$t\,[\text{s}] = N \cdot \frac{8}{f_{\text{cpu}}} \tag{2}$$

Der Bereich von 2.4 bis 16 ms entspricht Zählerdifferenzen N von 3317 bis 22118.

Die Schallgeschwindigkeit v beträgt bei 20° C 343.5 m/s. Setzt man Gleichung 2 und die Zahlen in Gleichung 1 ein, dann ergibt sich daraus

$$d = N \cdot \frac{343500}{2764800} \quad [\text{mm}]$$

oder nach Kürzen des Bruchs um 512

$$d \approx N \cdot \frac{671}{5400} \quad [\text{mm}] \tag{3}$$

Die Schallgeschwindigkeit ist temperaturabhängig. Im Bereich von −20° C bis +40° C gilt mit guter Näherung

$$v(T) = (331.5 + 0.6 \cdot T\,[^\circ\text{C}]) \quad [\text{m/s}]$$

[1] a. zwanghaft Kontrolle ausübender Mensch; b. Mikrokontroller programmierender Mensch

In der Zisterne herrschen etwa 8° C, und damit eine Schallgeschwindigkeit von 336.3 m/s. Der Zähler in Gleichung 3 verringert sich von 671 auf 657.

Sensor 1: Füllstand

Der Ultraschallsensor ist von Seeedstudio [8] und wurde von Watterott [9] bezogen. Der Sensor wird über einen Puls von mindestens 10 Mikrosekunden veranlasst, eine Messung durchzuführen. Der Sensor meldet die Zeit bis zum Eintreffen des Echos direkt als Pulsdauer an den Kontroller zurück.

Der Sensor wird über Pin D.5 mit Strom versorgt. Mit dem Pin D.6 wird einerseits die Messung ausgelöst und andererseits der vom Sensor zurückgelieferte Puls ausgemessen.

```
PORTD 5 portpin: ping_pwr       \ sonar: power
PORTD 6 portpin: ping           \ sonar: measurement
```

Um die Dauer des Antwort–Pulses zu messen, verwende ich den zu Timer1 gehörigen *input–capture*–Pin `D.6`. Tritt an diesem Pin eine Flanke auf, so wird die dazu registrierte Interrupt–Service–Routine aufgerufen. Leider kann man in diesem Betriebsmodus den Inhalt des Zählerregisters nicht manipulieren, d.h., es ist nicht möglich, den Zähler bei der steigenden Flanke zu löschen. Wir behelfen uns mit Differenzen. Bei der steigenden Flanke wird der Inhalt des *input–capture*–Registerpaars in die Variable `ic_t0` kopiert, bei der fallenden Flanke nach `ic_t1`. Anschließend wird die Einheit umkonfiguriert, um auf die entgegengesetzte Flanke zu reagieren. Die Funktion `ic-int-isr` bewerkstelligt das Speichern der Timer1–Zählerstände. Die Funktion `+ic1` registriert die Interrupt–Service–Routine und aktiviert den Interrupt. Die Funktion `-ic1` deaktiviert den Interrupt.

```
variable ic_t0
variable ic_t1
: ic-int-isr
  TCCR1B c@
  [ 6 bv \ ICES1
  ] literal and if      \ if (rising edge) {
    ICR1L c@ ICR1H c@   \ . read IC registers
    >< or               \ . combine H+L
    ic_t0 !             \ . store in ic_t0
    [ 6 bv invert       \ . change detection
    ] literal           \ . to falling edge
    TCCR1B and!         \ . .
  else                  \ } else (falling) {
    ICR1L c@ ICR1H c@   \ . read IC registers
    >< or               \ . combine H+L
    ic_t1 !             \ . store in ic_t1
    [ 6 bv              \ . change detection
    ] literal           \ . to rising edge
    TCCR1B or!          \ . .
  then                  \ }
;
: +ic1
  $42 TCCR1B c!         \ ICES1 | f_cpu/8
                        \ register ISR
  [’] ic-int-isr TIMER1_CAPTAddr int!
  [ 5 bv                \ enable interrupt
  ] literal TIMSK or!   \ .
;
: -ic1
  $00 TCCR1B c!         \ disable timer1
;
```

Um eine Messung durchzuführen, sind mindestens folgende Dinge nötig:

`ping.power.on` der Ultraschallsensor wird mit Strom versorgt (Wartezeit 1 Sekunde)

`+ic1` die Interrupt–Service–Routine wird registriert und der Interrupt aktiviert

`+ping` die Zählervariablen der input–capture–interrupt–service–routine werden gelöscht

`ping.trigger` der Messpin wird auf Ausgang und Pegel low konfiguriert, auf high gesetzt und sofort wieder auf low, dann wird der Pin wieder auf Eingang zurückgestellt. Dieses Wort wartet dann 40 Millisekunden. In dieser Zeit findet die Messung des Antwortpulses statt.

`d.get`, `d.decode` Danach werden die beiden Zählerstände zu einer Differenz zusammengefasst, in eine Distanz umgerechnet und das Ergebnis an geeigneter Stelle aufgehoben

`ping.power.off` schaltet den Ultraschallsensor ab

```
: ping.power.off ( -- )  ping_pwr pin_highZ  ;
: ping.power.on  ( -- )  ping_pwr pin_output
                         ping_pwr high ;
: +ping
  ping pin_highZ
  0 ic_t0 !             \ clear counters
  0 ic_t1 !
;
: ping.init
  +ic1
  +ping
;
: ping.trigger ( -- )
  ping pin_output       \ trigger sonar,
  ping high ping low    \ . measurement taken by
  ping pin_input        \ . input capture
  40 0 do pause 1ms loop \ wait for completion
;
```

In der Funktion `d.get` wird zwar eine Adresse auf dem Stapel erwartet, aber nicht benutzt. Alle Funktionen in diesem Projekt, welche einen Messwert produzieren, erwarten so eine Adresse. Das könnte die Adresse auf dem i2c–Bus oder eine Pin–Nummer sein. In diesem Fall wird eine solche Information nicht benötigt.

```
: d.get ( addr -- delta_t ok | error )
  ( addr ) drop         \ --
  +ic1                  \ --
  ping.trigger          \ -- ( wait for ICP1 )
  ic_t1 @ ic_t0 @ -  \ -- delta_t
  dup &300 > if
    0                   \ -- delta_t 0  ( ok )
  else
    drop                \ --
    1                   \ -- 1 ( error )
  then
  -ic1
;
```

Die Beobachtung vieler Messungen zeigt, dass gelegentlich der Antwortpuls unerklärlich kurz ausfällt. Ich habe nicht herausgefunden, woran das liegt. Aber ich habe beschlossen, Pulse mit einem Zählwert kleiner 300 zu verwerfen. Diese Version von `d.get` ist weiter verbesserungsfähig. Zum einen wird nie überprüft, ob der Ultraschallsensor eine *kein–Echo–registriert*–Bedingung zurückliefert (Puls von 38 ms). Zum anderen könnte man nach einem zu kurzen Puls wenigstens eine weitere Messung durchführen.

Die Umrechnung der Zeitdifferenz in eine Entfernung wurde in Gleichung 3 gezeigt, der entsprechende Programmcode ist nahezu trivial. Allerdings ist das die Entfernung vom Sensor zum Wasserspiegel und nicht der Füllstand der Zisterne.

```
\ d = 1/2 * v * t
\   = 1/2 * 343.5 [m/s] *  N*(8 [s]/f_cpu)
\   =  N  * 343500 / 2764800   [mm]
\  ~=  N  *    671 / 5400      [mm]
: d.decode ( delta_t -- d[mm] )
  &671 &5400 */
;
```

Ich habe separat überprüft, dass diese Umrechnung über den geforderten Bereich (0.4 — 4 m) korrekt funktioniert.

Damit existiert alles, um eine erste Applikation zu schreiben. Die Schleife macht etwa jede Sekunde eine Messung und gibt das Ergebnis aus.

```
: init
  ping.power.on
  1000 ms
  ping.init
;
: run
  init
  begin
    ping.trigger
    $ff d.get
    0= if
      d.decode
      decimal u. cr
    then
    &1000 ms
  key? until
  key drop
  -ic1
  ping.power.off
;
```

Der Sensor steckt auf einem 1 m langen Abwasserrohr, welches auf dem Boden steht, quasi ein Probeaufbau mit einer Länge von 1.07 m.

```
> ver
amforth 4.8 ATmega32 ok
> run
1048
1048
1048
1048
1052
1048
1048
1048
1048
1048
1048
 ok
>
```

Klar ist, dass man den Abstand noch in eine Füllhöhe umrechnen muss. Schwierig daran ist lediglich die Kalibration. Dazu muss der Sensor in der Zisterne montiert sein, das kommt also noch.

Sensor 2: Versorgungsspannung

Um die Versorgungsspannung zu messen, habe ich mich für einen simplen Spannungsteiler entschieden. 6 Widerstände mit jeweils 470 kΩ bilden einen Teiler, über einen Widerstand fallen dann etwa 1.8 V ab. Diese Spannung wird gegen die interne Spannungsreferenz von 2.56 V verglichen und mit 10 Bit Auflösung gemessen. Die Sicherung und die Verpolschutzdiode zwicken von der Akkuspannung noch etwas ab. Hier kommt man nicht ohne eine vernüftige Kalibration aus. Ein Labornetzgerät diente als *Akku*.

V	N	V_{supply}
8.00	474	7.870
9.00	541	8.875
10.00	607	9.865
11.00	673	10.855
12.00	739	11.845
13.00	804	12.820
14.00	871	13.825
15.00	938	14.830

Die Messwerte N bei verschiedenen Spannungen V_{supply} modelliert man mit einer Ausgleichsgeraden. Deren Steigung sollte sehr nahe bei $6 \cdot 2.56/1024$ liegen.

$$V_{\mathrm{supply}} = \mathrm{a}_1 \cdot 6 \cdot 2.56/1024 \cdot N + V_{\mathrm{offset}} \quad [\mathrm{V}] \qquad (4)$$

Die Berechnung der Ausgleichsgeraden ergibt die Koeffizienten

$$\mathrm{a}_1 = 1.00081$$

$$V_{\mathrm{offset}} = 0.826$$

Die Steigung stimmt gut genug, $\mathrm{V}_{\mathrm{offset}}$ habe ich mutwillig auf 0.93 erhöht, dann passen die umgerechneten Messwerte besser zur Anzeige von meinem Multimeter — reine Glaubenssache. Der dazugehörige Programmteil sieht recht übersichtlich aus.

```
PORTA 0 portpin: vsupply
include ewlib/portpin.fs \ pin>channel
include ewlib/adc.fs    \ +adc -adc
                        \ adc.init.pin adc.get{,10}
: +adc ( -- )
  [ %11 6 lshift    \ refs1 refs0 == internal V_ref
    5 bv         or \ adlar == left adjust result
  ] literal ADMUX c!
  \ AD enabled, prescaler=128
  [ 7 bv     \ ADEN
    2 bv or  \ ADPS2
```

```
    1 bv or  \ ADPS1
    1    or  \ ADPS0
  ] literal ADCSRA c!
;
: v.get    ( addr -- u ok )
  ( addr ) drop
  vsupply pin>channel adc.get10 0
;
: v.decode ( N[10bit] -- V_supply[0.01V] )
  \ V_supply = 6*2.56[V]/1024 * N + 0.83 + 0.1
  \ 6*256/1024 = 6/4 = 1.5
  3 2 */
  93 +
;
```

Der Strom durch den Spannungsteiler beträgt

$$I = \frac{U}{R} = \frac{14}{6 \cdot 470000} = 5\,\mu\text{A}$$

Angesichts der hohen Ströme, die in dieser Schaltung ständig fließen, ist das vernachlässigbar. Ist man beim Stromsparen (s.u.) allerdings in die μA–Region vorgedrungen, dann kann es sich lohnen, den Spannungsteiler abschaltbar zu machen.

Sensor 3: Temperatur

Weil wir schon gerade dabei sind, und weil die Temperatur einen Einfluss auf die Schallgeschwindigkeit hat, habe ich noch einen LM92–Sensor am i2c–Bus angehängt. Diesmal allerdings so, dass der Bus und der Sensor über einen extra Pin erst eingeschaltet wird — immer in der Hoffnung, ein paar mWh zu sparen.

```
PORTC 3 portpin:  pwr.i2c
$90 constant a_th1      \ lm92

include ewlib/twi.fs    \ +twi -twi twi.ping?
include ewlib/i2c.fs    \ >i2c <i2c
include ewlib/i2c_lmXX.fs \ lm.get
                        \ lm92.decode
: i2c.power.on
  pwr.i2c pin_output  pwr.i2c high ;
: i2c.power.off
  pwr.i2c low  pwr.i2c pin_input ;
\ note: call i2c.power.on *before* +twi
: twi.scan.blink
  $ff 0 do
    i twi.ping? if
      ledsensor led.blink
    then
  2 +loop
;
```

Die beiden Funktionen `lm.get` und `lm92.decode` beschaffen die Temperatur in 1/100 °C (zwei Nachkommastellen), der Sensor bietet eine Auflösung von 1/8 °C. Auch bei diesen Sensoren ist eine Kalibration kein Luxus. Die können schon einmal um 2 oder 3 °C daneben liegen.

Kalibration des Temperatursensors? Auflösung?

Schlafenszeit

Der Wasserstand in der Zisterne ändert sich nicht so besonders schnell. Daher ist es völlig ausreichend, wenn alle 10 Minuten eine Messung stattfindet. In der Zwischenzeit hat der Kontroller nichts zu tun. Damit der nicht unötig Strom frisst, lege ich ihn in den Wartezeiten schlafen. Dazu benötigt man eine Funktion `sleep`, die den gewünschten Schlafzustand (es gibt mehrere) vom Stapel nimmt und den Schlafmodus aktiviert. Es bietet sich an, den Modus `$03` zu verwenden. Der schaltet den Kontroller fast aus, lediglich `timer2` läuft weiter, getrieben vom externen Uhrenquarz (32768 Hz) und dem dazugehörigen Vorteiler. Stellt man den Vorteiler auf sein Maximum (1024), dann läuft der Timer alle 8 Sekunden über. Der Kontroller wird daraufhin aufgeweckt und die Interrupt–Service–Routine wird abgearbeitet (Buchhaltung der Zeit). Danach läuft die Hauptschleife weiter, stellt fest, ob es was zu tun gibt (lange genug gewartet?), und erledigt gegebenenfalls die Arbeit. Danach geht der Kontroller wieder schlafen.

Auf diese Weise lässt sich der Stromverbrauch in der Wartezeit von ca. 18 mA auf < 1 mA reduzieren. Das lohnt sich richtig. Dieser Programmteil zeigt nur die wesentlichen Teile, das komplette Programm befindet sich in der Datei `sonar-3.fs`

```
variable ticks      \ 8 seconds each
4 value idle.ticks  \ 4 * 8 = 32 seconds wait

\ timer2 overflow isr
: tick_isr_uptime
  1 ticks  +!
;
\ power save 8 seconds ticks
: +ticks-uptime
  $07 TCCR2 c!    \ clock_ts2/1024
  $08 ASSR  c!    \ source: 32 kiHz crystal
  \ register interupt service routine
  ['] tick_isr_uptime TIMER2_OVFAddr int!
  \ enable timer2 interupt
  TIMSK c@ $40 or TIMSK c!
;

: init
  \ ...
  idle.ticks 1+ ticks !
  +ticks-uptime
;
: once
  \ do work here
;
: run
  init
  begin
    ticks @ idle.ticks > if
      \ run job
      once
      1 ticks !
    then

    \ back to power down mode
    $03 sleep
  repeat
;
```

Notausgang 1

Das hier vorgestellte Programm benutzt keinen multitasker, weil die Daten selbständig verschickt und nicht über die serielle Schnittstelle abgefragt werden. Deswegen spendiere ich noch einen Taster, der in der Hauptschleife abgefragt wird. Ist der Taster gedrückt, dann wird die Schleife verlassen. Sonst muss man beim Ändern von Parametern den Kontroller komplett neu programmieren. Wenn die Schleife verlassen wurde, leuchtet die rote LED (`lederr`) auf und der Prompt erscheint an der seriellen Schnittstelle.

```
PORTA 3 portpin:  sw.user

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

    $03 sleep
  repeat
  leds.power.on
  +leds
  lederr low
;
```

Notausgang 2

Ein funktionierendes Programm ist eine erfreuliche Sache. Ein langzeitstabiles Programm ist aber möglicherweise mit Mehraufwand verbunden. Im Probebetrieb blieb die Sendestation hin und wieder stehen. Nach ausgiebigem LED–Gucken und Grübeln stellte sich der Verdacht ein, dass das Funkmodul sehr gelegentlich nicht mit dem erwünschten Pegelwechsel antwortet. Dann bleibt das bisher vorgestellte Programm einfach an dieser Stelle stecken[2]:

```
: w? ( -- ) _rfm12 low begin _miso pin_high? until ;
```

Was tun? Wenn *irgendwo* in der Datenversendung etwas nicht klappt, dann soll doch bitte das Funkmodul neu gestartet und der Transfer wiederholt werden. Das könnte ein Fall für `catch` und `throw` werden. Also habe ich den Artikel von Michael Kalus [4] hervorgekramt und ein paar Versuche gemacht. Derzeitiger Stand:

```
-200 constant #timeout

10   constant w.maxtimeout \ in 1ms
     variable w.state
: w? ( -- )
  _rfm12 low
  1 w.state !
  w.maxtimeout 0 do
    1ms
    _miso pin_high? if
      0 w.state !
      leave
    then
  loop

  w.state @ 0= not if
    \ timeout
    #timeout throw
  then
;

: w.restart
  lederr high
  rfm12.power.off &200 ms
  rfm12.power.on &1000 ms
  +spi +rfm12 rfm12.init
  wc? drop
;
variable D_x
: try2df.send ( D_x -- )
  dup D_x !
  leddata low
  ( D_x ) df.size [’] w.tx.data catch if
    \ timeout
    drop drop
    w.restart
    D_x @ df.size [’] w.tx.data catch if
        \ 2nd timeout
        drop drop
      then
  then
  leddata high
;
```

Ich definiere also eine eigene *exception*–Nummer, die von `w?` geworfen wird, falls sich das Funkmodul nicht innerhalb von 10 ms meldet. Viel weiter oben in der Aufrufhierarchie wird die Ausnahme mit `catch` abgefangen. Es werden höchstens zwei Versuche gemacht, den Datenblock zu versenden. Den Programmablauf habe ich mit dem schon erwähnten Taster getestet. Erst die Zeit wird zeigen, ob sich das bewährt: seit vier Wochen funkt der Kontroller ohne Ausfälle!

Messen und Daten versenden

In der Funktion `once` wird eine komplette Messreihe durchgeführt, die Daten aufbereitet und per Funk verschickt. Die einzelnen Messwerte werden zunächst in einer Datenstruktur `filter_mean:` akkumuliert. Ist die vorgesehene Anzahl Messungen beisammen, werden die Werte in `data.frame:` Strukturen eingetragen. Diese enthalten einen Kopf mit der Adresse von Sender und Empfänger (je ein Byte) sowie am Ende eine Checksumme. Die Sensoren und das Funkmodul werden nach Bedarf ein– und ausgeschaltet.

```
: once
  leds.power.on +leds
```

[2] Ein atmega ist kein Supercomputer. Deswegen ist es auch nicht erstaunlich, dass er mit einer endlosen Schleife eben nicht in zwei Sekunden fertig wird.

```
  i2c.power.on  +twi
  ping.power.on +ping +ic1
  +adc_internal vsupply pin_highZ
  &500 ms

  \ collect data
  data.reset
  collect.rounds 0 ?do
    ledsensor low
    data.collect
    ledsensor high
    &500 ms
  loop
  data.ls cr

  -adc
  ping.power.off
  -twi i2c.power.off

  \ prepare data frames
  data>frames

  \ send data
  lederr low
  rfm12.power.on &1000 ms
  +spi +rfm12 rfm12.init
  wc? drop \ read status

  D_T1 try2.df.send &500 ms
  D_D1 try2.df.send &500 ms
  D_U1 try2.df.send &100 ms

  rfm12.off wc? drop   \ switch radio off
  rfm12.power.off      \ power rfm12 off
  -rfm12               \ disable select
  -spi                 \ disable spi
  -leds leds.power.off \ disable spi

  lederr high
;
```

Selbstverständlich könnte man diese Funktion weiter faktorisieren, und die einzelnen Datenstrukturen in Listen verwalten. Allerdings sehe ich hier keine Notwendigkeit, das Programm flexibler zu gestalten. Änderungen an den Sensoren oder der Umrechnung erwarte ich nach dem Kalibrieren nur noch selten oder gar nicht.

Eine detaillierte Beschreibung der gewählten Datenstrukturen, sowie der Empfang der hier verschickten Datensätze erfolgt im nächsten Teil der Artikelserie.

So Sachen

1. Eigentlich find ich Ultraschallsensoren doof. Wir hören sie zwar nicht mehr, aber Fledermäuse und wahrscheinlich auch anderes Getier sehr wohl. Für mich ist eine Ultraschallquelle eine Art Umweltverschmutzung, so wie unnötiges Licht, das den Nachthimmel aufhellt. Zwar hat das nichts mit Forth zu tun, zugegeben, aber deswegen hab ich das Ding wenigstens in ein Rohr eingesperrt, so dass es nicht die ganze Gegend (lies Zisterne und die angeschlossenen Rohre) volldröhnt.

Eine lautlose Messmethode wäre diese: an einem Gewichtssensor hängt ein zylindrischer Körper, z. B. ein teilweise mit Sand gefülltes, geschlossenes Kunststoffrohr. Das Rohr reicht von seiner Aufhängung bis fast zum Boden der Zisterne. Ist die Zisterne leer, dann misst der Sensor das volle Gewicht des beschwerten Rohres. Steigt der Wasserspiegel, dann verringert sich das gemessene Gewicht um den Auftrieb. Dieser lässt sich aus dem ins Wasser ragenden Volumen des Rohrs berechnen. Ist die Zisterne voll, dann ist das Gewicht am geringsten, der Zusammenhang ist linear.

2. Mit dem Programm und dem zusammengekabelten Versuchsaufbau ist es halt nicht getan. Für mich ist das Herstellen einer mechanisch belastbaren und einigermaßen ordentlich aussehenden Kiste mit abgedichteten Kabeldurchführungen etc. durchaus ein Brocken, dessen Bewältigung sich länger hinzieht. Und ich vermute, dass das nicht nur mir so geht.

3. Ohne lange zu überlegen, habe ich irgendwann einen 7.2 Ah 12 V Bleigel–Akkumulator bestellt. Der sah groß genug aus, um den Zisternensensor monate– oder gar jahrelang zu versorgen. Schließlich hält der Technoline–Sensor mit zwei AAA–Batterien auch über ein Jahr durch. Also, das gute Stück mit einem alten 7805–Spannungsstabilisator für 5 Volt angeschlossen und los geht's. Nur dass die Klemmenspannung nach 10 Tagen schon nach Aufladen verlangte.

Dann tat ich das, was ich natürlich schon viel früher hätte tun sollen: Nachmessen. Es stellte sich heraus, dass die Schaltung locker 25 bis 35 mA schluckte. Dabei waren das Funkmodul und der Ultraschallsensor doch nur zum Messen bzw. Versenden überhaupt eingeschaltet. Das hatte ich so nicht erwartet. Also lötete ich nach und nach Teile ab, um die Verbraucher zu finden. Der MAX232 schluckte ein paar mA, aber nicht übermäßig. Meine Spannungsmessung der Akku–Klemmenspannung war auch nicht verantwortlich. Der Kontroller brauchte nur mit 11.0592 MHz–Quarz und Reset beschaltet stolze 24 mA. Unglaublich. 2 mA hätt ich ihm vielleicht gegönnt, 0.2 mA wären schick.

Es half, den 7805–Spannungsstabilisator, der auch deutlich Abwärme produzierte, durch einen Schaltwandlerchip (MAX667) zu ersetzen. Der wird kaum warm.

Aber sparen geht nur, wenn man den atmega32 schlafen legt, wie im Text beschrieben. Erst damit konnte ich den mittleren Stromverbrauch auf rechnerisch 1.3 mA reduzieren. Das ist fast ein Faktor 20 weniger. Also sollte der Akku damit etwa 200 Tage Strom liefern können — jetzt wird wahrscheinlich die Selbstentladung des Akkus zum Thema.

4. Messen will auch gelernt sein. Der Spannungsteiler ist sehr hochohmig. Der ADC verkraftet das gut, denn sein Eingangswiderstand liegt in der Größenordnung von über 100 MΩ. Das Multimeter allerdings bringt zu wenig Innenwiderstand mit und verändert damit die über einen dieser Widerstände gemessene Spannung. Hat man alles schon mal gewusst. Aufgefallen ist es, weil sich die mit

dem ADC gemessenen Werte ändern, wenn man mit dem Multimeter herumfuchtelt.

5. Ich habe mich schon länger gewundert, warum manche meiner Stationen nach Strom aus–/einschalten nicht mehr mit mir reden. Dabei tappte ich relativ lang im Dunkeln. Des Rätsels Lösung war die fehlerhafte Implementierung des Multitaskers: der speicherte die wichtigen Daten nicht im Flash–Speicher. Inzwischen wundere ich mich eher, dass manche Stationen einen kurzen Stromausfall überhaupt verkraftet haben. Der reparierte Multitasker ist ab Version 4.7 von `amforth` erhältlich. Dank geht an Matthias Trute für die Mithilfe.

6. Keine Antwort vom Funkmodul? Dann bleibt das Programm halt einfach in einer endlosen Schleife stecken — sowas soll ja schon mal vorkommen. Richtig unpraktisch daran ist nicht nur, dass keine Daten mehr versendet werden, sondern auch, dass zu der Zeit der Mikrokontroller und das Funkmodul eingeschaltet sind, was seinerseits den Akku ruckzuck leersaugt. Hier helfen `catch` und `throw` und das Detektieren einer Zeitüberschreitung, wie im Artikel beschrieben.

7. Für die Kalibration des Entfernungssensors hatte ich separate Rohre verschiedener Länge besorgt. Steckt man den Sensor auf ein solches Rohr, welches auf dem Boden steht, dann wird die Entfernung zum Boden gemessen. Steckt man unten ein zweites Rohr zur Verlängerung an, dann misst man die Entfernung zur Steckstelle! Die Steckstelle produziert ausreichend Echo, um den (zu empfindlichen?) Empfänger zu täuschen!

8. Ein letztes Rätsel: der Entfernungssensor funktioniert wunderbar, wenn ich ihn an einem Stück Draht in den Abschlussdeckel des Plastikrohrs hänge. Später habe ich ein schickes Gehäuse spendiert und den Sensor mit einem Stück Plexiglas fixiert — dann war er beleidigt. Die Messwerte waren unbrauchbar und v.a. unabhängig von der Entfernung. Ideen?

Nachtrag (11.08.2012)

Noch während diese Ausgabe der VD in Arbeit ist, habe ich weitere Erfahrung sammeln dürfen.

9. Nachdem der Sensor und der Kasten mit Akku und Elektronik in der Zisterne endlich eingebaut waren, zeigten sich immer wieder mehrstündige Lücken in den empfangenen Datensätzen. Also habe ich die Antenne auf die andere Seite der Platine geschafft und die Sendeleistung auf ihrem Maximalwert erhöht. Das hilft, aber am Ende nur begrenzt. Wahrscheinlich muss der Amateurfunker meines Vertrauens mir mal was zu besseren Antennen beibringen.

10. In diesem Zusammenhang habe ich die Übertragungsgeschwindigkeit von 4800 auf 1200 Baud reduziert. Mit einem billigen USB–DVB-T Empfänger und dem Programm `gnuradio` kann ich jetzt dem Funksensoren–Zoo beim Funken zugucken.

11. Der Ultraschallsensor wurde, wie oben angedeutet, ohne das Schallschutzrohr eingebaut. Dennoch war die erste Messung immer irgendwie *daneben* und produzierte eine viel zu geringe Entfernung. Diese eine Messung verfälscht den Mittelwert bekanntermaßen erheblich. Aus diesem Grund bin ich dazu übergegangen, $2 \cdot N$ Messungen durchzuführen, die Ergebnisse zu sortieren und nur die mittleren N Messungen zu einem Mittelwert zusammenzufassen. Der reinen Lehre folgt das nicht, aber es hilft.

12. Die Daten laufen seit heute auch in die produktive Datenbank und landen auf der Anzeige. Und ja, ich kann feststellen, wann ich zwei Gießkannen Wasser aus dem Vorrat entnommen habe, das entspricht einer Änderung der Füllhöhe von etwa einem halben Zentimeter.

Der geneigte Leser vermag schon allein an der Länge dieser *Sachen* erahnen, dass sich dieses Projekt über einige Zeit hingezogen hat. Meine ersten Versuche mit dem rfm12–Funkmodul fanden im Mai 2009 statt. Diese Dinge habe ich im April 2011 wieder hervorgekramt, nachdem Martin Bitter die im letzten Artikel vorgestellten Technoline–Thermometer–Sensoren erfolgreich belauschte. Zusammen mit einem nagelneuen Ultraschall–Entfernungsmesser entstand das hier vorgestellte Projekt. Gut Ding will auf jeden Fall Weile haben! Mein Dank gilt allen, die meine Fragen beantwortet und neue Ideen hervorgezaubert haben. Der wöchentliche IRC–Chat ist für mich inzwischen eine wichtige Einrichtung — vielleicht haben noch mehr Leser Lust auf einen *elektrischen Stammtisch*?

Referenzen

1. E. Wälde, Adventures in Forth 5, Die Vierte Dimension 4/2008, Jahrgang 24
2. E. Wälde und M. Bitter, Funklöcher!, Die Vierte Dimension 3/2011, Jahrgang 27
3. E. Wälde und M. Bitter, Technoline–Funksensoren belauschen, Die Vierte Dimension 4/2011, Jahrgang 27
4. M. Kalus, Catch & Throw, Die Vierte Dimension 3/2008, Jahrgang 24
5. `http://amforth.sourceforge.net/`
6. `http://www.pollin.de`
7. `http://www.hoperf.com`
8. `http://www.seeedstudio.com`
9. `http://www.watterott.com`
10. `http://en.wikipedia.org/wiki/Fletcher%27s_checksum`

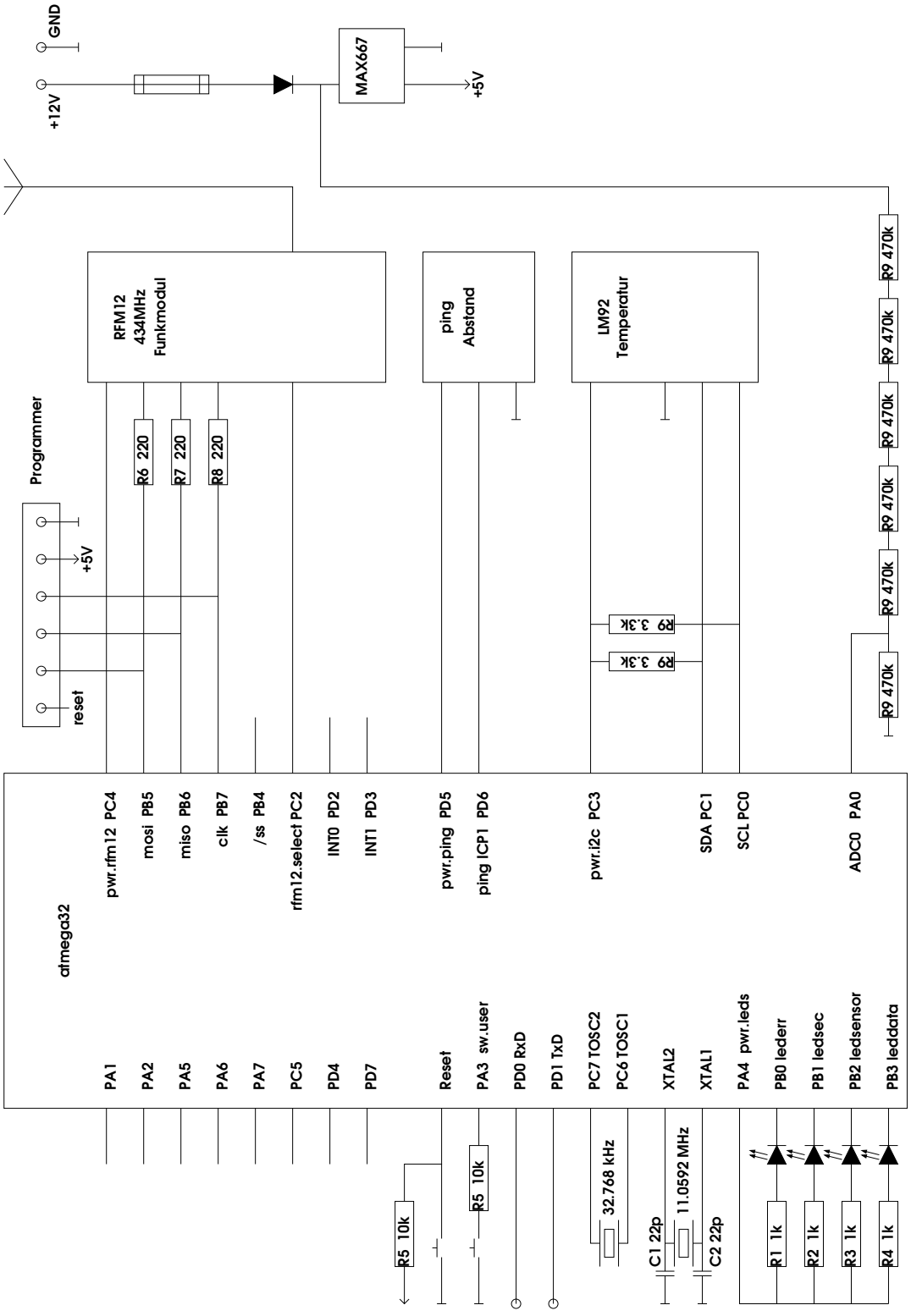


Abbildung 1: Schaltplan des Füllstandsensors. Entkoppelkondensatoren, Adressbeschaltung des LM92 und ähnliche Kleinigkeiten fehlen.

Listing

Das komplette, kommentierte Programm beläuft sich auf ca. 1200 Zeilen, wenn man alle `include`–Anweisungen auflöst. Deswegen ist hier lediglich die letzte Version des Hauptprogramms (`sonar-5.fs`) abgedruckt. Die vollständigen Quelltexte sind über die Webseite der Vierten Dimension verfügbar.

```
\ 2012-04-28  sonar-5.fs

\ included by make marker:
\ lib/misc.frt
\ lib/bitnames.frt
\ lib/ans94/marker.frt
\ $(AMFORTH)/devices/$(MCU)/$(MCU).frt
\ first.fs

marker --start--
include ewlib/format.fs \ sign! .i +.f
\ bitvalue, convert bit number [0..7] to mask
: bv    ( bit# -- mask )
  1 swap lshift ;
: or!   ( mask addr -- )
  dup c@ rot or  swap c! ;
: and!  ( mask addr -- )
  dup c@ rot and swap c! ;
: comma ( -- )   [char] , emit ;
: colon ( -- )   [char] : emit ;
: dot   ( -- )   [char] . emit ;
: .id   ( id -- )   space .i colon ;

\ additional exception numbers
-200 constant #timeout

\ pin definitions
PORTA 0 portpin:  vsupply
\   A 1
\   A 2
PORTA 3 portpin:  sw.user
PORTA 4 portpin:  pwr.leds
\   A 5
\   A 6
\   A 7

PORTB 0 portpin:  lederr
PORTB 1 portpin:  ledsec
PORTB 2 portpin:  ledsensor
PORTB 3 portpin:  leddata
PORTB 4 portpin:  /ss
PORTB 5 portpin:  _mosi
PORTB 6 portpin:  _miso
PORTB 7 portpin:  _clk

\   C 0           _scl
\   C 1           _sda
PORTC 2 portpin:  _rfm12
PORTC 3 portpin:  pwr.i2c
PORTC 4 portpin:  pwr.rfm12
\   C 5
\   C 6           32 kiHz Quarz
\   C 7           .

\   D 0           RxD
\   D 1           TxD
PORTD 2 portpin:  _int0
PORTD 3 portpin:  _int1
\   D 4
PORTD 5 portpin:  pwr.ping
PORTD 6 portpin:  ping
\   D 7           /rs485.rw

\ special addresses
$90 constant a_th1      \ lm92

\ --- famous includes -----------------------
include ewlib/mstarslash.fs
include lib/ans94/2x.frt

\ --- clock tick, sleep ---------------------
8 value collect.rounds \ number of samples
variable ticks      \ 8 seconds each
4 value idle.ticks  \ 4 * 8 = 32 seconds wait
2variable uptime
: .uptime_sec uptime 2@ d.i [char] s emit ;
: .uptime
  uptime 2@
  2dup decimal d.i [char] s emit space
  &60 ud/mod  &60 ud/mod  &24 ud/mod
  d.i    [char] d emit space    \ days
  2 u0.r [char] : emit          \ hours
  2 u0.r [char] : emit          \ minutes
  2 u0.r                        \ seconds
;
: ++uptime  1 s>d uptime 2@ d+ uptime 2! ;

\ timer2 overflow isr
: tick_isr_blink
  ledsec toggle
;
: tick_isr_uptime
  1 ticks  +!
  8 s>d uptime 2@ d+ uptime 2!
;
\ enable ticks
\ crystal:   32768 /sec
\ 1 second ticks, blink ledsec
: +ticks-blink
  $05 TCCR2 c!    \ clock_ts2/128 )
  $08 ASSR  c!    \ source: 32 kiHz crystal
  \ register interupt service routine
  ['] tick_isr_blink TIMER2_OVFAddr int!
  \ enable timer2 interupt
  TIMSK c@ $40 or TIMSK c!
;
\ power save 8 seconds ticks
: +ticks-uptime
  $07 TCCR2 c!    \ clock_ts2/1024
  $08 ASSR  c!    \ source: 32 kiHz crystal
  \ register interupt service routine
  ['] tick_isr_uptime TIMER2_OVFAddr int!
  \ enable timer2 interupt
  TIMSK c@ $40 or TIMSK c!
;
\ disable ticks
: -ticks
  TIMSK c@
  [ $40 invert $ff and ] literal
  and TIMSK c!    \ clr Timer 2
```

```
  $00 ASSR  c!
  $00 TCCR2 c!
  $C0 TIFR  c!    \ clr interrupt flag
;


include leds.fs         \ leds.power.{on,off}
                        \ +leds -leds
                        \ led.blink

\ --- sensor 3: temperature -----------------
include ewlib/twi.fs    \ +twi -twi twi.ping?
                        \ twi.scan
include ewlib/i2c.fs    \ >i2c <i2c
include ewlib/i2c_lmXX.fs \ lm.get
                        \ lm{75,92}.decode
: i2c.power.on
  pwr.i2c pin_output  pwr.i2c high ;
: i2c.power.off
  pwr.i2c low   pwr.i2c pin_input ;
\ note: call i2c.power.on *before* +twi
: twi.scan.blink
  $ff 0 do
    i twi.ping? if
      ledsensor led.blink
    then
  2 +loop
;

\ --- sensor 1: sonar -----------------------
include ewlib/input_capture.fs  \ +ic1 -ic1
include ewlib/sonar.fs
: d.get ( addr -- delta_t ok | error )
  ( addr ) drop
  +ic1                  \ --
  ping.trigger            ( wait for ICP1 )
  ic_t1 @ ic_t0 @ -     \ -- delta_t
  \ fixme: check for timeout
  dup &300 > if
    0                   \ -- delta_t 0 ( ok )
  else
    drop                \ --
    1                   \ -- 1 ( error )
  then
  -ic1
;
: d.decode ( delta_t -- d[mm] )
  &671 &5400 */
;

\ --- sensor 2: v_supply --------------------
include ewlib/portpin.fs \ pin>channel
include ewlib/adc.fs    \ +adc -adc
                        \ adc.init.pin
                        \  adc.get{,10}
: v.get    ( addr -- u ok )
  ( addr ) drop
  vsupply pin>channel adc.get10 0
;
: v.decode ( N[10bit] -- V_supply[0.01V] )
  \ V_supply = 6*2.56[V]/1024 * N + 0.93
  \ 6*256/1024 = 3/2
  3 2 */
  93 +
;


\ --- wireless (rfm12) ----------------------
include ewlib/spi.fs    \ /ss +spi -spi
include ewlib/rfm12.fs  \ +rfm12 rfm12.init
                        \ >wc wc? w? >w <w
: rfm12.power.on
  pwr.rfm12 high
  pwr.rfm12 pin_output
;
: rfm12.power.off
  pwr.rfm12  low
  pwr.rfm12  pin_input
;

: rfm12.tx-21dB
  $9857 >wc ( 90kHz hub, -21db power )
  wc? drop
;
: rfm12.tx.on  ( -- )  $8238    >wc ;
: rfm12.tx.off ( -- )  $8208 w? >wc ;
: rfm12.off    ( -- )  $8201    >wc ;
: rfm12.sync   ( -- )  $aa >w $aa >w $aa >w ;
: rfm12.magic  ( -- )  $2d >w $d4 >w ;

: w.tx.data ( addr n -- )
  rfm12.tx.on
  rfm12.sync rfm12.magic    \ send preamble
  ( n ) 0 ?do               \ send data frame
    ( addr ) dup i + c@ >w
  loop
  drop
  rfm12.sync                \ send tail
  rfm12.tx.off
;

\ --- data handling -------------------------
include ewlib/filter_mean.fs   \ filter_mean:
            \ mean_reset mean_addup mean_eval
include ewlib/crc-fletcher16.fs
       \ crc.fletcher16 ( addr len -- chsum )
include ewlib/data-frame.fs
       \ data.frame: df.erase df.dump df.show
include ewlib/dot-d-n.fs
: df.head:
  create ( D.idid  idid  --  )
  ,         \ store idid
  ,         \ store *D.idid
does>    ( -- addr )
  \ place addr on stack
;
: dd ( D.x -- ) \ debug
  hex
  dup df.dump cr
  dup df.show  ." ...checksum:" space
  dup df.size crc.fletcher16 dup 4 u0.r space
  0= if
    ." ok"
  else
    ." CRC ERROR"
  then cr
  ." ...data.set:" space
  dup df:src+ c@ over df:dat+
  2 .Dn cr \ print id:N,min,mean,max record
  drop ( D.x )
;

\ temperature
```

```
$01 filter_mean: F_T1
\ distance
$02 filter_mean: F_D1
\ supply_voltage
$03 filter_mean: F_U1
\ data.frame: D.T1
\ D.T1 $7201 df.head: X1  \ X1 >dfList

: data.reset
  F_D1 mean_reset
  F_T1 mean_reset
  F_U1 mean_reset
;
: data.collect
  $ff    d.get 0= if
    d.decode    F_D1 mean_addup then
  a_th1 lm.get 0= if
    lm92.decode F_T1 mean_addup then
  $00    v.get 0= if
    v.decode    F_U1 mean_addup then
;
: data.ls
  decimal
  1 .i colon .uptime_sec
  F_T1 mean_eval .F2 space
  F_D1 mean_eval .F0 space
  F_U1 mean_eval .F2 space
;

$72 value EEstationID
variable stationID
$71 value EEdestID

\ --- manage dataframes ---------------------
: df.head ( dest counter addr -- )
  >r
              r@ df:cnt+ c!
              r@ df:dst+ c!
  stationID @ r@ df:src+ c!
  r> drop
;
: df.data.invalid ( id 0 addr -- )
  df:dat+ df.payloadsize $ff fill
  drop drop
;
: df.data ( id max d.mean min N addr -- )
  df:dat+ >r
      r@ 2 + ! \ N
      r@ 4 + ! \ min
  d>s r@ 6 + ! \ mean
      r@ 8 + ! \ max
      r>     ! \ id
;
: df.checksum ( addr -- )
  dup >r
  df.size 2 - crc.fletcher16
  >< r> df:crc+ !
;

: df.fill ( SENSOR destination addr -- )
  >r                    \ -- S dst
  $aa r@                \ -- S dst $aa addr
  df.head               \ -- S
  ( S ) mean_eval
              \ -- id max mean min N>0 | id 0
  dup if
    r@ df.data          \ --
  else
    r@ df.data.invalid  \ --
  then
  r> df.checksum        \ --
;

data.frame: D_D1 \ distance
data.frame: D_T1 \ temperature
data.frame: D_U1 \ supply-voltage

: data>frames
  F_T1 EEdestID D_T1  df.fill
  F_D1 EEdestID D_D1  df.fill
  F_U1 EEdestID D_U1  df.fill
;

: w.restart
  lederr high
  rfm12.power.off &200 ms
  rfm12.power.on &1000 ms
  +spi +rfm12 rfm12.init
  wc? drop
;
variable D_x
: try2.df.send ( D_x -- )
  dup D_x !
  leddata low
  ( D_x ) df.size [’] w.tx.data catch if
    \ timeout
    drop drop
    w.restart
    D_x @ df.size [’] w.tx.data catch if
        \ 2nd timeout
        drop drop
      then
  then
  leddata high
;


\ --- main loop -----------------------------
decimal
: init
  EEstationID stationID !

  \ -jtag \ moved to applturnkey.asm
  -spi
  sw.user pin_pullup_on
  rfm12.power.off -spi

  leds.power.on +leds
  lederr    low   200 ms
  ledsec    low   200 ms
  ledsensor low   200 ms
  leddata   low   200 ms
  lederr    high  200 ms
  ledsec    high  200 ms
  ledsensor high  200 ms
  leddata   high  200 ms
  ledsec low
  +ticks-blink
  5000 ms
  -ticks

  i2c.power.on
```

```
385     +twi 20 ms
386     hex twi.scan.blink
387     -twi
388     i2c.power.off
389
390     -leds
391     leds.power.off
392
393     0 s>d uptime 2!
394     idle.ticks 1+ ticks ! \ run job immediately
395     +ticks-uptime
396
397     data.reset
398   ;
399
400   : once
401     leds.power.on +leds
402     i2c.power.on  +twi
403     ping.power.on +ping +ic1
404     +adc_internal vsupply pin_highZ
405     &500 ms
406
407     \ collect data
408     data.reset
409     collect.rounds 0 ?do
410       ledsensor low
411       data.collect
412       ledsensor high
413       &500 ms
414     loop
415     data.ls cr
416
417     -adc
418     ping.power.off
419     -twi i2c.power.off
420
421     \ prepare data frames
422     data>frames
423     D_D1 df.show \ debug
424     D_T1 df.show \ debug
425     D_U1 df.show \ debug
426
427     \ send data
428     lederr low
429     rfm12.power.on &1000 ms
430     +spi +rfm12 rfm12.init
431     w.status cr \ wc? drop \ debug
432
433     D_T1 try2.df.send &500 ms
434     D_D1 try2.df.send &500 ms
435     D_U1 try2.df.send &100 ms
436
437
438     rfm12.off wc? drop   \ switch radio off
439     rfm12.power.off      \ power rfm12 off
440     -rfm12               \ disable select
441     -spi                 \ disable spi
442     -leds leds.power.off \ disable spi
443
444     lederr high
445   ;
```

```
446   : run
447     init
448     begin
449       ticks @ idle.ticks > if
450         1 ticks !
451         \ run job
452         once
453       then
454
455       \ leave loop when sw.user
456       \ is pressed (pin low)
457       sw.user pin_high? while
458
459       \ back to power down mode
460       $03 sleep
461
462     repeat
463
464     leds.power.on
465     +leds
466     lederr low
467   ;
468
469   : run-turnkey
470     applturnkey
471     run
472   ;
473
474   \ ' run-turnkey is turnkey
```

```
1   ; sleep-m32.asm
2   ;
3   ; sleep   ( mode -- )
4   ;
5   ; mode selects the type of sleep mode to
6   ; use. For details refer to the datasheet.
7   ;
8   ; on atmega-32 sleep modes are handled via
9   ; register MCUCR, not SMCR and bits at a
10  ; different bit location 6-4, not 3-1.
11  ; SE is bit 7 not bit 0.
12
13  ; ( mode -- )
14  VE_SLEEP:
15    .dw  $ff05
16    .db  "sleep", 0
17    .dw  VE_HEAD
18    .set VE_HEAD = VE_SLEEP
19  XT_SLEEP:
20    .dw  PFA_SLEEP
21  PFA_SLEEP:
22    andi    tosl, 7     ; mask illegal bits
23    swap    tosl        ; move to correct pos.
24    ori     tosl, $80   ; set the SE bit
25    out     MCUCR, tosl ; set the sleep config
26    sleep               ; nighty-night
27
28    in      tosl, MCUCR ; clear upper nibble
29    andi    tosl, $0F   ; of MCUCR
30    out     MCUCR, tosl ; mega32 specific!
31    loadtos             ; pop argument from stack
32    jmp     DO_NEXT
```

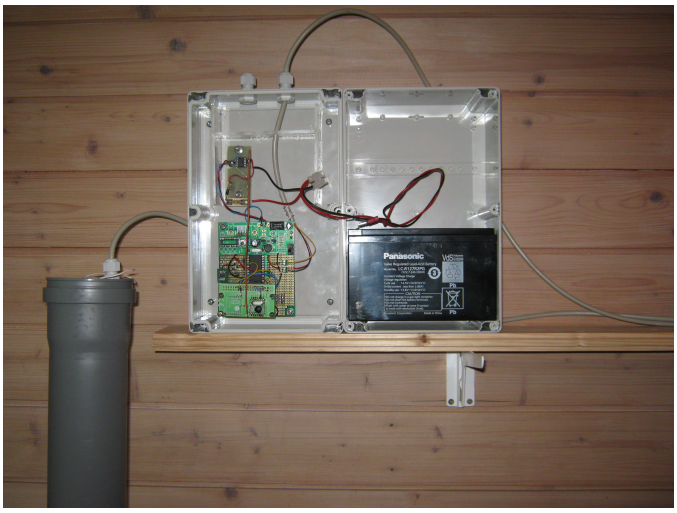


Abbildung 2: Der Zisternensensor im Probebetrieb

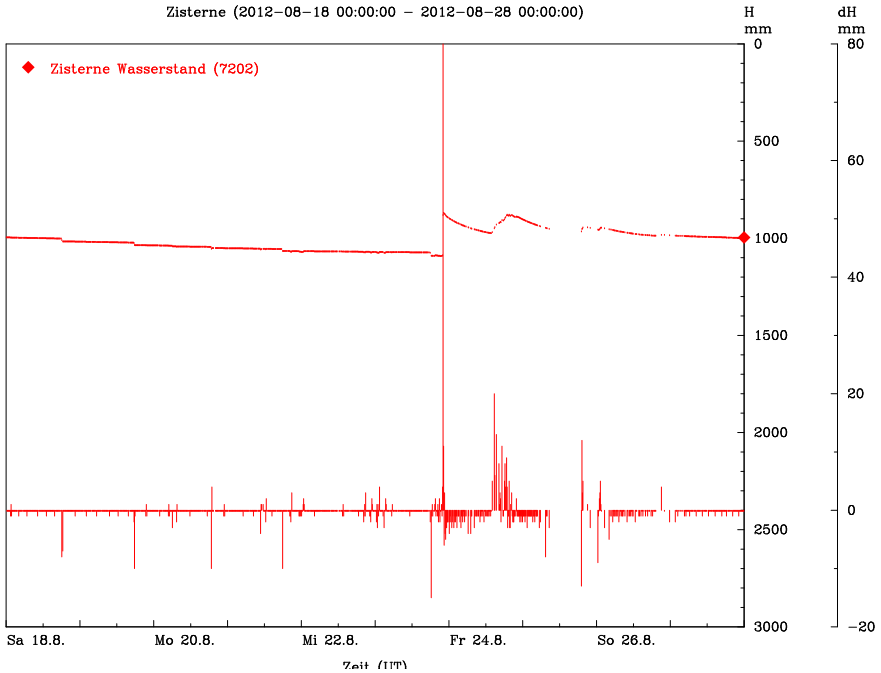


Abbildung 3: Daten für 10 Tage vom 18. bis 27. August 2012. Gezeigt ist der Wasserstand (obere Kurve) und die Änderung seit der letzten Messung in anderem Maßstab (untere Kurve). Auffälligste Änderung ist der sprunghafte Anstieg des Wasserstands in der Nacht vom 23./24.8. durch einen Gewitterregen. Am Freitag nachmittag regnete es etwas gemütlicher. Die nach unten gerichteten Zacken in der unteren Kurve an den Tagen 18. bis 21. und 23.8. wurden durch Entnahmen mit der Gartenpumpe verursacht. Die Kurven sind am 25.8. für etliche Stunden unterbrochen — Funk ist eben nicht immer zuverlässig.

Fortsetzung von S. 5

Und Dirk fügte aus eigenen Erfahrungen hinzu:

Deshalb habe ich auch immer im Forth–System selbst gearbeitet, und die Header dringelassen. Wenn ich abhandengekommen wäre, hätte jemand anders, der Forth kennt, Änderungen vornehmen können. Bin nicht abhandengekommen, und niemand wollte Änderungen haben. Es lief einfach gut.

Gruß,
Dirk.

Quelle:
Im englischen Originalbeitrag beginnt E. Rather so: *I remember once in the early 90's when Forth, Inc. got a call ...* Ihre Aussage ist zu finden in der Kette, die sich aus dem ursprünglichen Beitrag von Chris Hinsley *why bother with standards ?* am Montag, 19. November 2012 17:03:53 UTC+1 auf comp.lang.forth ergeben haben.

Links:
`http://www.forth.com` "Forth-based products and Services for real-time applications since 1973."

Michael Kalus

Forth-Assembler hilft, wenn keine Caps-Lock-LED eingebaut ist

Fred Behringer

Mit diesem Forth-Programm kann die Lock-Taste bei (Caps/Ins/Scroll/Num)-Lock daraufhin abgefragt werden, in welchem Zustand sie sich momentan befindet. Es wird alles über den BIOS-Interrupt 16h unter der Funktion 2 geregelt. Das Programm arbeitet mit Turbo-Forth, aber auch mit ZF — und sehr wahrscheinlich auch mit vielen anderen Forth-Systemen, deren Assembler so aufgebaut ist wie bei Turbo-Forth oder ZF.

Laptops ohne LEDs

Beim Googlen erfährt man u.a. von GigaGremlin (Nickname des folgenden Google-Autors): *Für alle, die, wie ich, an ihrem Note-/Netbook keine LED-Anzeige für ihre Feststelltaste mehr haben und sich darüber beim schnellen Tippen ärgern, habe ich eine kleine LED programmiert, die sich in die Taskleiste (rechts unten, bei der Uhr) einklinkt und bei eingeschalteter Feststelltaste grün/gelb aufleuchtet.*

Offensichtlich gibt es also Laptops ohne LEDs! Aber mehr noch: Was passiert denn, wenn man bei Windows (ich darf mich auf Windows 95 beschränken) eine Textdatei anzeigen lässt? Ich habe dazu beispielsweise das Listing des vorliegenden Artikels als Datei mit dem Namen `capslock.fth` abgespeichert und über den Windows-Explorer aufgerufen. Das Festgestelltsein der Caps-Lock-Taste wird dann überhaupt nicht angezeigt, obwohl dafür eine LED vorhanden ist, die unter DOS ja auch zur allgemeinen Zufriedenheit funktioniert! "Unter DOS" soll heißen, wenn ich Turbo-Forth (`forth.com`) aufrufe und zu `include capslock.fth` ergänze. Ähnlich bei Ins, Num und Scroll. Bei Ins ist auch in Desktop-Systemen normalerweise gar keine LED eingebaut. (Genaue Arbeitsweise unter Windows siehe weiter unten.) Und wenn man sich bei Wikipedia erkundigt, wundert man sich darüber, dass in den verschiedenen Systemen und Betriebssystemen ein vernünftiges Arbeiten überhaupt möglich ist: Da geht ja in den diversen real existierenden Systemen alles heillos durcheinander!

Wenn die Abfrage oder das Setzen nicht schon beim Hochfahren erledigt werden muss und wenn man sich für die Abfrage solange Zeit lassen kann, bis dass DOS oder Win-3.11 oder Win-95 oder Win-98 oder Win-ME geladen sind, dann kann das unten stehende Programm (siehe Listing) helfen. Ob es bei "höheren" Windowsen auch noch arbeitet, muss von Fall zu Fall erkundet werden. Ich habe in erster Linie DOS (DOS 6.2, FreeDOS etc.) vor Augen.

Aber ach!

Es ist nahezu unmöglich vorherzusagen, was beim Drücken welcher Taste (ich spreche von den Tasten, die nicht am Bildschirm angezeigt werden) geschieht oder nicht geschieht. Bei höheren Windows-Systemen schon gar nicht, aber auch schon bei Windows 95 nicht. Das hier entwickelte Programm könnte leicht auch dazu erweitert werden, schnell und bequem zu überprüfen, ob Shift- oder ähnliche Tasten gerade gedrückt wurden oder nicht. Ich verzichte auf eine solche Erweiterung und beschränke mich hier auf (Ins/Caps/Num/Scroll)-Lock.

Um es vorwegzunehmen: Bei Aufruf des Forth-Wortes `Anzeige` des vorliegenden Artikels wird auf dem Bildschirm beispielsweise folgende Information angezeigt:

```
   Ins-Lock (Einfuegen)           ist gesetzt
  Caps-Lock (Nur Grossbuchstaben) ist geloest
   Num-Lock (Ziffernblock)        ist gesetzt
Scroll-Lock (Bildschirm-Rollen)   ist geloest
```

Ob in den einzelnen Zeilen *gesetzt* oder *gelöst* steht, hängt natürlich davon ab, welche der dafür zuständigen Feststelltasten zuvor gedrückt wurde. Man beachte schon jetzt, dass das Programm auch in der Lage ist anzuzeigen, ob die Einfügetaste im Moment gesetzt oder gelöst ist, obwohl mein Computer (wohl alle Computer) dafür gar keine LED hat. In Wikipedia liest man über

> Caps-Lock und Shift-Lock
>
> Auf deutschen Tastaturen: Umschaltsperre (Shift-Lock), auf englischen Tastaturen: Feststelltaste (Caps-Lock).

Auf meiner Qwertz-Tastatur

gilt unter DOS, aber auch wenn Turbo-Forth oder ZF unter Windows 95 über den Explorer aufgerufen wird, Folgendes: Die Caps-Lock-Feststelltaste kann beliebig gesetzt und auch wieder gelöst werden. Ist sie gesetzt, dann leuchtet die mittlere LED und alle Buchstaben werden groß geschrieben. Bleibt es bei gesetzter Feststelltaste und man drückt auf die Umschalttaste, dann leuchtet die LED weiterhin und alle Buchstaben werden aber klein geschrieben - solange, bis die Umschalttaste wieder freigegeben wird. Die LED bleibt am Leuchten, solange die Feststelltaste gesetzt bleibt. Löst man die Feststelltaste wieder, indem man sie ein weiteres Mal drückt, dann erlischt die LED und alle Buchstaben werden bei gelöster Umschalttaste klein und bei gedrückter Umschalttaste groß geschrieben. Die LED bleibt bei gelöster Umschalttaste gelöscht. Ob Ziffern oder Sonderzeichen, hängt vom Zustand der Feststelltaste (nicht der Umschalttaste) ab. Mit anderen Worten: Über die Feststelltaste (CapsLock) kann man die Wirkung der Umschalttaste umkehren. Ich

habe das unter Turbo-Forth und auch unter ZF nachgeprüft.

Ruft man aber bei Windows 95 über den Explorer oder per NotePad eine Textdatei auf, dann ist die Prozedur etwas anders. (Ich berichte nur von meinen eigenen Anlagen. Über die Verhältnisse bei Fremd-Anlagen mache ich keine Aussage.) Die Feststellfunktion (Caps-Lock) arbeitet mit einer Umschaltsperre (Shift-Lock) Hand in Hand. Eine LED-Anzeige kommt nicht ins Spiel, weder bei Caps noch bei Ins oder Num oder Scroll. Ich konnte Folgendes beobachten:

- Nichts drücken: Kleinschreibung.
- Dann Umschalt drücken: Groß bei jedem Drücken, sonst klein.
- CapsLock drücken: Ab jetzt nur noch groß.
- Umschalt drücken und gedrückt lassen: Immer noch weiter nur groß.
- Rückflanke abwarten: Ab jetzt normal (groß bei Drücken, sonst klein).

Die Rückflanke der Umschalttaste entriegelt also eine schon gesetzte Feststelltaste. Mit anderen Worten: Will man nur groß schreiben, dann drücke man einmal die Feststelltaste. Will man nur klein schreiben, dann nehme man das Feststellen wieder zurück (indem man mindestens einmal die Umschalttaste drückt und wieder loslässt). - Die Ziffern und die Sonderzeichen verhalten sich genau so wie die Buchstaben.

Will man also abwechselnd einen Klein- und dann einen Groß-Buchstaben schreiben, dann lässt man ganz normal Buchstabe-Umschalt+Buchstabe-Buchstabe aufeinanderfolgen. Will man dagegen mit einem Großbuchstaben anfangen und ab dann abwechselnd einen Klein- und dann wieder einen Großbuchstaben schreiben, dann lässt man CapsLock-Buchstabe-Umschalt+Buchstabe aufeinanderfolgen.

Und was ist mit den LEDs?

Bei der Bearbeitung einer Textdatei unter Windows 95 zeigen die LEDs keine Reaktion (siehe oben). Sie verharren in demjenigen Zustand, den sie vor dem Einsprung in die Datei-Bearbeitung hatten. Mit anderen Worten: Sie verhalten sich so, als ob sie gar nicht vorhanden wären. Über den tatsächlichen Zustand der Feststelltasten bei Ins/Caps/Num/Scroll kann man sich dann aber durch geeignete Einbeziehung des hier besprochenen Forth-Programms Klarheit verschaffen. Ein Grund mehr für den vorliegenden Artikel: Korrektur eines unverständlichen Verhaltens der Win95-LEDs!

Auch bei Windows-XP?

Neugierdehalber habe ich meine Untersuchungen auch auf XP ausgedehnt. Da geht alles (wenn man sich auf das Aufrufen des Forth-Laufzeit-Systems mit dem eingebundenen Programm des vorliegenden Artikels beschränkt) genauso gut wie unter DOS oder unter Turbo-Forth in Win95: Die Caps-Lock-Feststelltaste kehrt Groß- und Kleinbuchstaben um. Ziffern und Sonderzeichen verhalten sich wie die Buchstaben. (Da sagt man immer, DOS-Programme können unter XP nicht aufgerufen werden!)

Eine Schwierigkeit bei ZF

ergibt sich wie folgt: Jedes Mal nach dem Drücken der Feststelltaste (beim Setzen wie auch beim Lösen) erscheint ein grafisches Zeichen auf dem Bildschirm, eine Art Doppelstrich-Cursor. Den muss man erst mit der Entferntaste weglöschen, bevor man das Forth-Wort anzeige (aus meinem Programm) aufrufen kann.

Stimmen aus dem Internet über XP

Googlet man die Foren-Meldungen im Internet durch, dann merkt man, dass doch nicht alles eitel Sonnenschein ist. Ich darf willkürlich eine solche Meldung zitieren, die mir ins Auge gefallen ist - zum Heraussuchen des genauen Ortes dieser Meldung kann man sich auch wieder auf Google stützen ;-), nämlich:

> Windows XP Prof startet nach Ruhezustand nur ab und zu einmal mit aktivierter Caps-Lock, aber LED auf der Tastatur bleibt aus. Das heißt, wenn ich nach dem Ruhezustand mein Logon-Kennwort wieder eingeben möchte, kommt ein falsches Kennwort. Sonst habe ich bei dieser Fehlermeldung immer ein Reboot durchgeführt, aber heute hatte ich so eine Eingebung: Einfach mal die Caps-Lock-Taste drücken und mit dieser das Kennwort einzugeben.
>
> WOW, und es funktionierte. Aber das Witzige dabei ist, dass jetzt die Caps-Lock-LED auf der Tastatur leuchtet und alles, was ich eintippe, klein geschrieben wird.
>
> Jetzt musste ich wieder ersteinmal die Shifttaste drücken und wieder Caps-Lock. Dann wieder Shift, - und erst jetzt funktionierte die Groß-Klein-Schreibung wieder korrekt.
>
> Kennt jemand das Problem und weiß vielleicht einer Abhilfe?

Soweit der Hilferuf des mir namentlich nicht bekannten Foren-Rufers. Nun ja, der geneigte Leser des vorliegenden Artikels *kennt* inzwischen *das Problem und weiß Abhilfe*. Für mich steht fest: Ich mache mir über die diversen Windowse keine Gedanken mehr. Im vorliegenden Artikel spreche ich über den BIOS-Interrupt 16h in der Funktion 2 - und der läuft über das BIOS — ob DOS oder nicht. Was Windows daraus macht, soll mir egal bleiben.

Das gilt für Turbo-Forth. Aber selbst bei ZF unter DOS (!), ich habe das nachgeprüft (siehe oben), wird da hineingefunkt! Es ist zum Verzweifeln! Und von sowas leben die Google-Foren!

Num-Lock und Scroll-Lock

Num-Lock schaltet den Ziffernblock ein oder aus, Scroll-Lock blockiert das Rollen des Bildschirms - oder löst es wieder.

Auf jedem PC anders

Die genaue Wirkungsweise der möglichen Umschaltungen und der genaue Zustand der vorhandenen oder nicht vorhandenen LEDs scheint in verschiedenen Systemen verschieden zu sein (siehe Wikipedia). Das hier gezeigte Programm (ein Programm zur Fingerübung in Forth-Assembler!) kümmert sich um die Feststelltasten. Und wie ist es, wenn man sich über das Gedrücktsein von (Shift left/Shift right/Alt/Ctrl) informieren möchte?

Gedrücktsein von Tasten anzeigen

Ob die linke oder rechte Shift-Taste im gegebenen Moment gedrückt ist, braucht am Bildschirm nicht angezeigt zu werden. Man sieht es ja an den Fingern, die die Tasten drücken. Man vergesse aber nicht, dass es auch Menschen gibt, die da Schwierigkeiten haben. Für sie könnte der Artikel ohne Weiteres ausgedehnt werden. Über zwei Daten-Bytes im BIOS-Datenbereich erfährt man alles Nötige. Eine der Tatsachen, die ich nachgeprüft habe, ist schon mal die folgende: Bei allen hier besprochenen Tasten weigern sich meine Anlagen, beim gleichzeitigen Drücken von mehr als einer Taste irgendeine Reaktion zu zeigen.

Tasten per Programm drücken?

Was den BIOS-Datenbereich betrifft, liest man generell die Empfehlung, Daten-Zellen nur auszulesen, nicht zu beschreiben. Na ja, es ist ja ohnehin schlecht vorstellbar, dass Keyboard-Tasten durch Beschreiben einer BIOS-Datenzelle gedrückt oder sonst wie beeinflusst werden könnten. Im BIOS-Datenbereich sind die (bitweise auszuwertenden) Bytes 0000:0417h und 0000:0418h für gedrückte Tasten (Bit=0/1) verantwortlich.

Keine LED für Ins

Die Einfügetaste (Ins) ist normalerweise (auch bei Desktops!) mit keiner LED zur Anzeige verknüpft. Das ist ein doppelter Grund für den vorliegenden Artikel. Schließlich möchte man ja auch schon bei einem unter Forth schnell zusammenprogrammierten Editor (für einen solchen braucht man ganz bestimmt keine Heerschar von professionellen Programmierern) wissen, ob und wann die Einfügevorrichtung eingeschaltet ist!

Und Shift-Lock?

Es gibt auch noch die Möglichkeit, bei (Right-Shift/Left-Shift/Alt/Ctrl) herauszubekommen, ob die betreffende Taste bei Eingabe des zuletzt eingegebenen Zeichens gedrückt war oder nicht. Das wird über die Funktion 2 des Interrupts 16h, und zwar über die Bits 0 bis 3 des Registers al, erledigt. Ich gehe darauf nicht ein, da bei Aufruf von *anzeige* in meinem Programm das *zuletzt eingegebene Zeichen* ja immer die Eingabetaste gewesen sein muss.

Für Turbo-Forth und ZF

Um unbequemen Diskussionen aus dem Weg zu gehen, lege ich mich auf Turbo-Forth in der 16-Bit-Version (siehe taygeta.com oder den Bremen-Mirror) fest. Das Programm habe ich auch unter ZF getestet. Die vom Forth-Assembler-Teil benötigten Befehle sind in Turbo-Forth und in ZF gleich. Und bei den Reaktionen des Programms und der LEDs habe ich keinen Unterschied zwischen ZF und Turbo-Forth feststellen können.

Einbinden per include (bei TF) oder fload (bei ZF)

Bei Turbo-Forth:

```
forth include capslock.fth
```

Bei ZF:

```
zf   fload capslock.fth
```

Endung `.fth` nicht vergessen!

Epilog

In ganz früheren Zeiten hat man solche außerhalb der eigentlichen Computer-Sprache liegenden Übungen per BASIC erledigt - und SYS-Befehle mit PEEK und POKE und Hex-Adressen zum Einsprung in den BASIC-Kernel verwendet. In Forth geht das genau so schön, und bequemer, schneller und viel einfacher nachzuvollziehen. Maschinen-Befehle und Interrupt-Aufrufe können in Forth spielend leicht in die Hochsprache eingebunden werden.

Das Problem herauszubekommen, ob CapsLock bei einem PC oder Laptop ohne LEDs gesetzt ist oder nicht, verdanke ich einer Aufgabe, die Reinhard Augustin am 3.11.2010 in einem Vortrag im Senioren-Computer-Kreis (SCK) in Vaterstetten gestellt hat.

Listings

Statusanzeige der Lock–Tasten

```
1   hex
2
3   code gss  ( -- byte )
4       al al xor  02 # ah mov  16 int    \ (g)et (s)hift (s)tatus
5       ah ah xor  ax push  next end-code \ al enthaelt Status (alle Bits)
6
7   : Anzeige ( -- )
8      cr             \ Anzeige, ob gesetzt oder nicht (d.h.: geloest)
9      gss 80 and
10         80 = if   ."    Ins-Lock (Einfuegen)           ist gesetzt" cr
11              else ."    Ins-Lock (Einfuegen)           ist geloest" cr
12              then
13     gss 40 and
14         40 = if   ."   Caps-Lock (Nur Grossbuchstaben) ist gesetzt" cr
15              else ."   Caps-Lock (Nur Grossbuchstaben) ist geloest" cr
16              then
17     gss 20 and
18         20 = if   ."    Num-Lock (Ziffernblock)        ist gesetzt" cr
19              else ."    Num-Lock (Ziffernblock)        ist geloest" cr
20              then
21     gss 10 and
22         10 = if   ." Scroll-Lock (Bildschirm-Rollen)   ist gesetzt" cr
23              else ." Scroll-Lock (Bildschirm-Rollen)   ist geloest" cr
24              then ;
```

Nachtrag

Und es gibt sie doch! Die per Programm ansteuerbaren PC-Tasten! Schaltet man sein Forth–System nach `hex` und gibt die Colon-Definition

```
: scroll-on  ( -- )  0 417 lc@ 10  or 0 417 lc! ;
```

ein, so lässt deren Aufruf die ScrollLock–LED aufleuchten und die ScrollLock–Wirkung einsetzen. Hat man dem System zusätzlich noch das Wort

```
: scroll-off ( -- )  0 417 lc@ ef and 0 417 lc! ;
```

einverleibt und ruft es danach auf, so verlöscht die LED wieder. Ich rede, konkret gesagt, von Turbo–Forth in der 16–Bit–Version. Bei ZF müssen `lc@` und `lc!` durch `c@l` und `c!l` ersetzt werden. Bei anderen PC–Kompatiblen entsprechend. In irgendeiner DOS–Box unter Windows natürlich nur dann, wenn das System nicht (schon) im Protected–Mode arbeitet und dadurch der Zugriff auf den BIOS–Datenbereich verwehrt bleibt.

Schreibt man dann im Interpreter–Modus hintereinanderweg:

```
scroll-on 2000 ms scroll-off
```

so wird die LED nur `on`–geschaltet, egal, ob sie vorher schon `on`–geschaltet war oder vom `Off`-Geschaltetsein herkommt. Sobald dann aber irgendeine (eine beliebige!) Taste auf dem Keyboard (selbst so ausgefallene Tasten wie Shift–Left oder Bild–Up) gedrückt wird, verlöscht die LED wieder, wenn sie vorher `on`–geschaltet war. (Wenn sie vorher schon `off`–geschaltet war, bleibt sie (natürlich?) `off`. Zwischendurch darf Vieles geschehen (ein ganzes Programm darf ablaufen), ohne dass die LED sich rührt. (Was im BIOS–Datenbereich an der Stelle 0000:0417 in der Zwischenzeit liegt, habe ich natürlich per `lc@` und `lc!` in allen Variationen untersucht.)

Sinngemäß gilt das eben Gesagte auch für die umgekehrte Wortfolge:

```
scroll-off 2000 ms scroll-on
```

Ähnliches gilt unabhängig voneinander auch für die Tasten NumLock und CapsLock. InsLock hat normalerweise keine LED. Wird eine der beiden obigen Forth–Wort–Folgen ausgeführt und danach eine beliebige Taste gedrückt, dann wird also der Tastatur–Kontroller in denjenigen Zustand versetzt, den er vor dem zusätzlichen Tastendruck innehatte.

Wo finde ich ein schnelles Rezept zur entsprechenden Rücksetzung des Tastatur–Kontrollers (ich akzeptiere nur Intel/AMD-Assembler-Befehle)? Ein Ersatz also für den *zusätzlichen Tastendruck*, eine brauchbare Tastendruck–Simulation? Die vielen guten Ratschläge im Internet bis hin zur fünften Google–Subebene konnten mir bisher nicht weiterhelfen. Ein schönes Wort aus dem Forth–Wortschatz — wie kompliziert sein Aufbau auch sein mag — würde mir auch schon gefallen. Durch Beschreiben des Tastatur–Puffers mit dem Scancode der Enter–Taste (oder irgendeiner anderen Taste) bin ich nicht weitergekommen.

Wave Engine (4)

Hannes Teich

Konzept-Überlegungen: Es sieht ganz so aus, als ob es sinnvoll sei, die bislang beschriebene Version des PC-Synthesizers Wave Engine *in eine weitere münden zu lassen. Dabei könnte einiges vereinfacht werden, und Code zum Experimentieren gäbe es endlich auch.*

Die aktuelle (zweite) Version ist nun schon ein paar Jahre alt. Es ist nicht Altruismus, der mich das Monstrum hier beschreiben lässt, sondern es scheint der einzige Weg zu sein, mich noch zu Lebzeiten am fertigen Produkt erfreuen zu können. Seit ich davon in der VD berichte (Übersicht am Ende), hat das Programm an Struktur gewonnen und sich von ferne dem Endziel genähert.

Version 1

Version 1 lief im Prinzip ganz ordentlich (man höre z. B. die Chopin-Skizze auf meiner Website), aber sie wurde mit einem Problem nicht fertig: Wenn ein neuer Ton einsetzt, noch ehe der Vorgänger verklungen ist, so will es der Zufall oft, dass die Kurve des alten Tons nach unten strebt, wenn die neue Kurve steil nach oben führt. Dadurch entsteht eine unschöne hörbare Spitze. Damals versuchte ich, mit dem Toneinsatz auf einen positiven Nulldurchgang des Vorgängers zu warten. Das ging recht gut mit Mono, nicht aber mit Stereo.

Die beiden Stereo-Kanäle wurden bislang nur für ein Phasen-Vibrato *missbraucht*. Dazu werden die Töne des einen Kanals um ein Quäntchen nach unten, die des anderen nach oben verstimmt, gerade so viel, dass eine langsame Schwebung entsteht. Das klingt recht gut, aber die Nulldurchgänge sind verschieden.

Die Version 1 lief übrigens mit Win32Forth, das Gforth unter Linux weichen musste.

Version 2

Hier ist das Problem so gelöst, dass der alte Ton kurz vor Einsatz des neuen so weit bedämpft wird, dass die Spitzen nicht mehr stören. Der Effekt wurde *Staccato* genannt. Zwei weitere Dämpfungen sind *Fading* (Ausklingen) und *Sustain* (kurzes Nachkingen eines beendeten Tons).

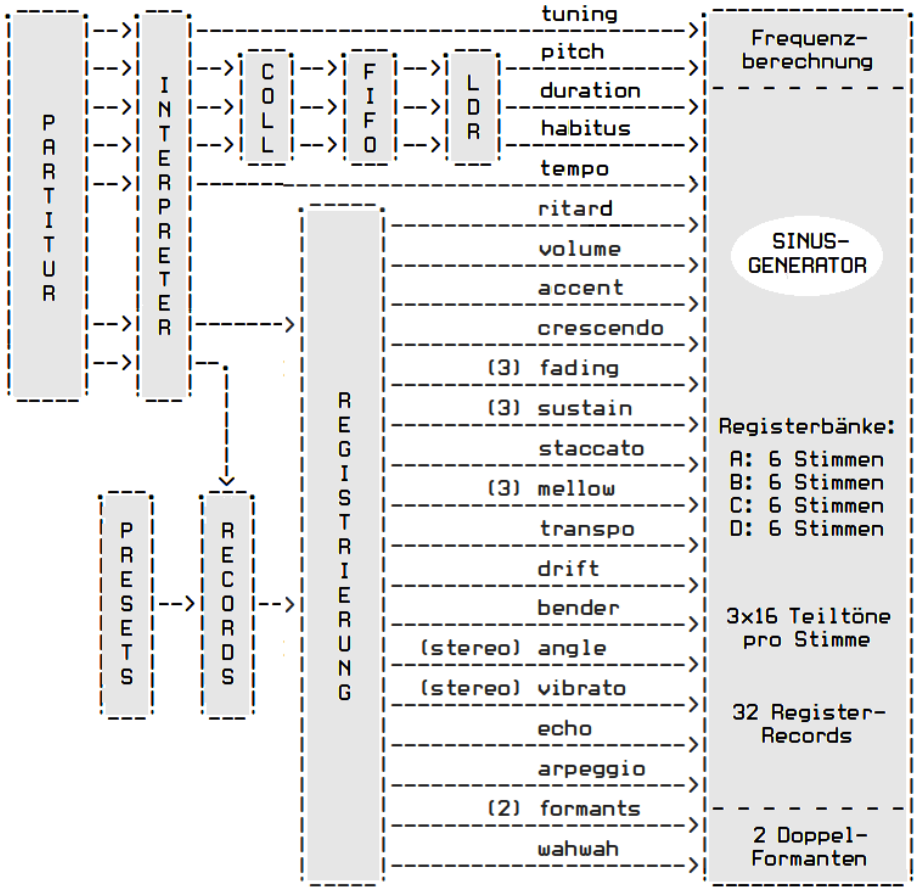


Abbildung 1: Blockbild der 2. Version

Zur Erinnerung zeigt Abbildung 1 nochmal das Blockbild. Es ist ein FIFO-Puffer zu erkennen, der Tonhöhe, Tondauer und Habitus transportiert. Er ist nötig, weil die vier Manuale gleichzeitig erklingen sollen, während der Computer die vier Zeilen in der Partitur nur nacheinander lesen kann. Es handelt sich um einen (4-fachen) Ringspeicher, der ständig Daten aufnehmen und abgeben kann. Zwei Zeiger wachen über die Adressierung der Daten sowie darüber, dass Unter- und Überlauf vermieden werden.

Soweit war das noch übersichtlich und reizvoll. Komplizierter wurde es durch eine Funktion, die es erlaubt, *Phrasen* (Gruppen von Tönen) wiederholt erklingen zu lassen. Dazu muss mit Hilfe eines dritten Zeigers in den Pufferspeicher zurückgegriffen werden. Die Anweisung zu dieser Aktion wird selbst durch das FIFO transportiert und muss in geeigneter Weise aus der Partitur-Syntax gewonnen werden. Und das ist noch nicht alles. Es funktioniert zwar, hat aber, so meine ich, den Titel Krampflösung verdient.

Version 3

Es besteht der Verdacht, dass unter den noch zu realisierenden Funktionen zusätzlicher Bedarf entstehen könnte, in den Puffer einzugreifen. Deshalb suchte ich nach einer besseren Lösung, fand sie und bin zerknirscht, dass sie mir nicht eher eingefallen ist.

Daten für Töne, die zugleich erklingen sollen, müssen dem Generator zugleich angeboten werden. Das übernahmen bisher die Schieberegister, warum aber nicht eine entsprechend aufbereitete Datei? Die Interpretation der Partitur-Syntax könnte unangetastet bleiben. Es müsste nur der im Bild zu erkennende *Collector*, der die Schieberegister füttert, durch einen einfachen *Compiler* ersetzt werden, der die Aufgabe hat, eine maschinengerechte Zwischendatei zu schreiben. Damit könnte die aufwändige Schieberei entfallen. Wiederholungen von Phrasen würden einfach so oft geschrieben, wie sie erklingen sollen. Ich sehe nur Vorteile.

Den Konjunktiv möchte ich hiermit fallen lassen und die Compiler-Lösung anstreben. Und ich möchte nicht in der Version 2 herumbasteln, sondern eine Version 3 Stück für Stück hochziehen und jeweils den Code zugänglich machen.

Bleibt die Frage, warum solcher Code dieser Artikelfolge nicht schon beiliegt. Ja, da hatte ich ein seltsames Erlebnis: Nach einigen Wochen Pause funktionierte die Wave Engine nicht mehr so, wie ich sie verlassen hatte. Mit jeder nennenswerten Änderung war der letzte Stand in einem Silo hinterlegt worden, aber die letzten zehn Ausgaben funktionierten alle nicht mehr! Dasselbe Gforth wie zuvor, aber dem Betriebssystem (Ubuntu) war ein Update verpasst worden. Nach langem Suchen fand ich heraus, dass ich das Ende einer Tabelle übertreten hatte. Das war leicht zu korrigieren, aber es hängt offenbar eine Menge mehr daran. Da muss leider noch geforscht werden, ehe ich Code für die Version 3 abzweigen kann.

Referenzen

In folgenden VD-Heften wurde bisher über die Wave Engine berichtet:

- 2/2011 Top-One-Partitur
- 3/2011 Wave Engine (1)
- 1/2012 Wave Engine (2)
- 2/2012 Wave Engine (3)

Beachten Sie bitte den Dateibereich der Website der Forth-Gesellschaft unter `http://www.forth-ev.de/filemgmt/viewcat.php?cid=54` sowie die Site des Autors unter `http://www.stocket.de/WE`

Forth–Tagung 2013

Heinz Schnitter

Die Forth 2013 findet in der Zeit vom 19. bis 21. April 2013 in Garmisch–Partenkirchen statt. Als Zusatztag bieten wir Donnerstag, den 18. April 2013 an.

Die Organisation der Tagung machen Bernd Paysan und wir, meine Frau Ulrike und ich. Bernd übernimmt den „Call for Papers“ und organisiert die Vorträge, Ulrike und ich kümmern uns um das Tagungshotel und das Rahmenprogramm.

Beim Schreiben dieser Zeilen hatte ich ein Déjà–vu: Unsere Tagung findet ja vom 19. April bis 21. April 2013 im Forsthaus Graseck statt. Das ist das gleiche Datum und die gleichen Wochentage wie bei der Tagung vor 11 Jahren am selben Ort. Man könnte also den Kalender von 2002 nächstes Jahr wieder verwenden. Nun bin ich neugierig geworden und habe mich im Wiki schlau gemacht: *„Nach den Regeln des Gregorianischen Kalenders wiederholt sich in einer Periode von 400 Jahren exakt der Ablauf der Schalttage. Somit kehren mit der Tages- und Monatszählung auch die Wochentage wieder.“* Innerhalb dieser Periode findet man natürlich viele Jahre mit identischem Kalender. Häufige Abstände, von Jahren mit gleichem Kalender, sind 6, 11 oder 28 Jahre.

Das Alpenhotel, Forsthaus Graseck, hat eine hauseigene Seilbahn. Alle Zimmer sind mit Bad/Dusche, WC, Telefon und Balkon ausgestattet.

Die Anreise mit dem Auto oder der Bahn über München nach Garmisch–Partenkirchen ist bequem möglich. Mit dem Auto fährt man ab München auf die Autobahn A95 Richtung Garmisch–Partenkirchen, dort folgt man den

Hinweisschildern „Olympiastadion“, „Skistadion“, „Eckbauer“ oder „Partnachklamm“. Ab dem Skistadion dürfen Tagungsgäste mit dem Auto bis zur Talstation der hoteleigenen Bergbahn fahren.

Die Navi–Adresse des Hotels: 82467 Garmisch–Partenkirchen, Wildenau.

Je nach Witterung planen wir eine Bergwanderung (vom Eckbauer zum Tagungshotel) und eine Fackelwanderung durch die Partnachklamm. Möglich wäre auch eine Fahrt mit der Zahnradbahn oder der Seilbahn auf die Zugspitze oder eine Besichtigung von Schloss Linderhof, erbaut von König Ludwig II. Diese Aktivitäten bieten wir für den Freitag oder Samstag Vormittag an. Die Fackelwanderung ist für den Samstagabend geplant. Wer an unseren „Outdoor–Aktivitäten“ teilnehmen möchte, sollte unbedingt festes Schuhwerk und warme Kleidung mitbringen.

Bitte schicken Sie ihre Anmeldung sobald wie möglich an das Tagungsbüro, da wir das Zimmerkontingent bereits Ende Februar festlegen müssen. Alle Anmeldungen werden nach dem Prinzip „first come, first served“ bearbeitet. Auch 2013 gilt: **„Wer zu spät kommt, schläft im Tal!“**

Bei Fragen wegen der Tagungsanmeldung wenden Sie sich bitte an Heinz Schnitter, und bei Fragen zu Vorträgen und Workshops bitte an Bernd Paysan. Unsere gemeinsame Email–Adresse für Anregungen und Fragen: `tagung@forth-ev.de`.

Wir freuen uns auf euer Kommen!

Heinz und Ulrike Schnitter und Bernd Paysan

EuroForth 2012

Bernd Paysan

Exeter College

Die EuroForth organisiert dieses Jahr Nick und Janet Nelson im bezaubernden Exeter College im Zentrum von Oxford. Das College aus dem 17. Jahrhundert ist ein Ort, an dem man sich drehende Treppen vorstellen kann, und natürlich weder (Swap–)Drachen noch bärtige Programmierer fehl am Platz sind. Wie üblich, findet zwei Tage vor der eigentlichen Konferenz das Forth200x–Standardisierungstreffen statt, und dieses Mal haben wir auch etwas erreicht: Der Release Candidate ist beschlossene Sache [1], er steht mittlerweile zum Public Review bereit.

Zum Seitenprogramm der Konferenz gehören die abendlichen Gespräche. Leider hat sich Oxford — obwohl es eine Studentenstadt ist — nicht als ideal dafür erwiesen. Zum einen machen die Pubs um 11 Uhr abends zu, also für Nachteulen so gefühlt am frühen Nachmittag. Aus dem Konferenz–Raum werden wir auch frühzeitig herausgeschmissen, und die Common Rooms können nicht mit Hogwards mithalten — es gibt nur einen, und der ist eher ungemütlich. Am Freitag um halb zwölf fordert uns dann sogar der Portier auf, zumindest die Fenster zu schließen, weil wir zu laut seien.

Viel mehr gibt es aber nicht zu kritisieren — Mittagessen sind die landesüblichen Sandwiches, außer natürlich am Sonntag, da gibt es das landesübliche Beef, noch halb roh. Das Abendessen war ganz gut, und nur am Mittwoch Abend hat uns ein heftiger Regenschauer die Laune verdorben. Sonst war für englische Verhältnisse recht viel blauer Himmel zu sehen.

Forth 200x Meeting

Anton Ertl hat 2004 auf der EuroForth in Dagstuhl den Anstoß zu einer Überarbeitung des ANS–Forth–Standards gegeben, und dieses Meeting ist ein vorläufiger Höhepunkt: Das Standard–Komitee übergibt das aktuelle Dokument als „Release Candidate“ der öffentlichen Begutachtung — wie es sich für einen Standard gehört. Der Name ist Forth 2012.

Die Standard–Treffen laufen gemäß den ANSI–Regeln nach einer relativ starren und etwas bürokratischen Prozedur. Wir halten uns an diese bewährten Regeln, außerdem erlauben sie uns, den Standard etwa bei ANSI einzureichen. Auch wenn das eher nicht zur Diskussion steht, denn der Stempel von ANSI ist ziemlich teuer.

Zunächst (nach der Feststellung der Anwesenheit) wird das Protokoll des letzten Meetings abgesegnet, d.h. die Teilnehmer sind sich einig, dass es auch den Fakten entspricht. Danach gehen wir gemeinsam die Änderungen des Dokuments durch. Dabei kann sich durchaus herausstellen, dass gewisse Sachen nicht ganz durchdacht waren — insbesondere das Alignment der Parameter von xFIELD:–Definitionen mussten wir überarbeiten. Denn letztlich addiert so eine xFIELD:–Definition einfach nur einen Offset zu einer Adresse.

Danach wenden wir uns den CfVs zu, also den Änderungen am Standard, die durch einen Call for Votes gegangen sind. Dieses Mal gab’s nur noch einen: Der Vorschlag zur Aufnahme von S>F und F>S, damit man einfach–genaue Zahlen in Fließkommazahlen wandeln kann, und nicht nur, wie in ANS Forth vorgesehen, doppelt–genaue. Schließlich sind inzwischen recht viele 64–Bit–CPUs unterwegs, und es ist für diese CPUs etwas mühsam, eine 128–Bit–Zahl in eine 64–Bit–Fließkommazahl zu wandeln. Den Vorschlag haben wir dann gleich noch leicht geändert, um ihn an die tatsächliche Praxis anzupassen

(viele Forth–Systeme haben diese zwei Wörter seit langem drin) — dabei geht es hauptsächlich um das Verhalten bei Überlauf. Das überlässt man der FPU, und die hat seit dem 80387 einen Bug, der konsequent beibehalten wurde, und natürlich ist es deshalb eine „ambiguous condition“, wenn man einen Überlauf provoziert.

Es gibt dann noch ein paar Fast–Track–Proposals, die keine substantiellen Änderungen bedeuten, aber die Verständlichkeit des Textes erhöhen.

Der nächste größere Punkt ist die letzte Politur am Snapshot, damit daraus ein Release Candidate wird — dazu müssen wir uns auch mit Teilen des Standard–Dokuments befassen, die wir bisher weitgehend ignoriert haben. Wir haben 6 Punkte abzuarbeiten: Wir müssen den Standard, so er im Dokument selbst erwähnt ist, in „Forth 2012“ umbenennen (das ist natürlich ein LaTeX–Makro). Referenzen auf den vorhergehenden Standard, also ANS Forth, oder „Forth 94,“ wie wir es jetzt nennen; die also keine Selbst–Referenz sind, müssen dabei natürlich erkannt und entsprechend behandelt werden. Wir lassen das Kapitel 5 so, wie es ist. Die Bibliographie, die sehr viele ziemlich alte Werke enthält, bleibt auch erhalten, soll aber nochmal um die wenigen neuen Werke ergänzt werden. Annex C, die „Perspective,“ wird entfernt. Für Compatibility und den Portability Guide haben wir Änderungen, die eingepflegt werden.

Danach gucken wir uns noch kurz die offenen Baustellen an, als da wären:

Internationalisierung: Zwei der drei Teilstücke haben wir mit dem Xchar–Wordset und der Substitution schon eingebaut. Das dritte, die Lokalisierung, steht noch aus.

Cross–Compiler, die nochmal überarbeitet werden müssen, um an den Stand der Technik angepasst zu werden

IEEE Floating Point, die der neuen Revision des IEEE 754 folgen soll

FOUND, also die Frage nach dem Ersatz für FIND, für den es noch nicht mal ansatzweise eine gemeinsame Praxis gibt, weil noch nicht einmal völlig klar ist, was jetzt ein Xt ist (siehe [3])

Dateinamen und Verzeichnisse: Hier geht es um Pfad–Trenner wie / oder \ und Dateiendungen wie .fs/.fth/.4th

Verzeichnisse, bei denen mit der Aufnahme von SUBSTITUTE und ein paar gängigen Macros sich langsam eine brauchbare Grundlage einstellt

Memory Access, also Wörter, die 8/16/32/64 Bits in verschiedener Endianess aus dem Speicher holen

Da sich während des Meetings neue Fragen aufwerfen, werden diese natürlich auch noch bearbeitet. Der größte Block ist die Überarbeitung des Rationals. Zuletzt wird beschlossen, den Release Candidate 1 rechtzeitig zum Forth Day der SVFIG fertigzustellen, und dort dann das Public Review zu starten, und darüber per E–Mail abzustimmen. Mehr Details findet man im Protokoll von Peter Knaggs [2].

EuroForth Konferenz

Die eigentliche Konferenz fand im Salon des Rektors statt, dicht gepackt mit vielen Vorträgen [4]. Die erste Session begann recht theoretisch, aber der größte Teil der Vorträge bezieht sich auf Forth in der Praxis.

Bill Stoddard zeigte in „Forth semantics for Computer verification,“ dem Vortrag mit den Kringeln, wie man die Ausführung von Forth in einer für Informatiker üblichen formalen Semantik beschreibt. Es gibt da so Operationen, wie hintereinander Ausgeführtes, und aus einem Store in den Hauptspeicher wird eine damit an einen Wert gebundene Variable. Auch der Stack wird als Konkatenation von Werten aufgefasst; die Stack–Operatoren transformieren diesen Stack.

Mit diesen Werkzeugen kann man Forth einer formalen Verifikation unterziehen; in dem Forschungsprojekt ging es darum, eine reversible Programmiersprache, Ruth–R, auf das reversible Forth, das Bill Stoddard schon früher mal vorgestellt hat, zu übersetzen, und zu beweisen, dass Quelle und Ergebnis identisch sind.

Ian van Breda zeigte mit „Building an LR Parser Using Forth“, wie man einen Pascal–Compiler in Forth schreibt, und damit auch Leuten, die — aus welchen Gründen auch immer — sich weigern, Forth zu verwenden, eine „akzeptable Syntax“ an die Hand zu geben. Die Gründe, Forth nicht zu mögen, illustriert er u.a. am Beispiel eines Programmierers, dem Forth nicht gefallen hat, weil es die als Unterstriche eingegebenen Bindestriche nicht verstanden hat.

Ulrich Hoffmann hat mit „Applying Model checking techniques to Forth“ ein weiteres theoretisches Thema. Auch hier formuliert man Eigenschaften des Systems mit mathematischen Formeln, und versucht, diese Eigenschaften (des Models) zu beweisen.

Willi Stricker beendet den Ausflug in die Kringel und Pfeile mit „Strip Forth Prozessor presentation“, und zeigt seinen Forth–Prozessor im Actel–FPGA–Board, und zeigt dabei, wie er sich ein Umbilical System vorstellt, bei dem ein Host–System ein kleines Target fernsteuert.

Anton Ertl stellt am nächsten Morgen sein „Objects 2“ vor, einem System, das „Duck Typing“ kann, aber klassisch und performant über Tabellen implementiert wird, die dann halt teilweise leer sind. Da OOP–Anwendungen in Forth nicht exorbitant groß sind, sind diese Tabellen trotzdem nicht zu groß.

Klaus Schleisiek setzt das Thema Forth und OOP mit „Explaining simpleOOP“ fort. Nach wie vor versucht Klaus, eine neue Art des virtuellen Methoden–Dispatches zu erfinden, aber die Semantik nähert sich dem an, was andere Leute darunter verstehen. Ebenfalls erklärt er, wie die Parser–Erweiterung des Forth–Systems funktioniert, damit man das „information hiding“ implementieren kann.

Andrew Haley greift das „Standardize Forth OOP“ von Stephen Pelc vom Vorjahr auf, und schlägt vor, Method Dispatch und das Anlegen von Selektoren für Methoden einfach auch als Methode der virtuellen Tabelle zu implementieren, damit verschiedene Systeme mit verschiedenen Ansätzen zumindest zusammenarbeiten können.

Dirk Brühl präsentiert in „Forth for Education — 4€4th and 4€4th IDE“ die TI–Launchpads mit dem angepassten CamelForth und einer IDE in VisualForth. Ziel ist es dabei, jugendliche Bastler mit einfach zu programmierender Billighardware für Forth zu begeistern. Das geht dann nahtlos in einen von

Stephen Pelc initiierten Workshop zum Thema „Educational Forth“ über.

Andrew Read zeigt nach dem Mittagessen das Nr. 1–Highlight der Tagung, „The N.I.G.E. Machine: an FPGA based micro computer,“ ein Forth–Prozessor im FPGA, der so weit ausbaut ist, dass man ihn als richtigen Computer (mit Keyboard und VGA) verwenden kann.

Stephen Pelc hält seinen „Notation Matters“–Vortrag, den wir in der letzten VD bereits in deutscher Übersetzung lesen konnten.

Gerald Wodni hat mit MPE zusammen ein „Forth to .NET interface“ entwickelt, das die verschiedenen eher komplexen Möglichkeiten der Introspektion eines .NET–Systems nutzt, um Methoden einer .NET–Klasse von Forth aus aufzurufen.

Peter Knaggs implementiert mit „Java Forth“ ein Forth in Java, und experimentiert damit, wie man das Typsystem von Java von Forth aus nutzen kann.

Bernd Paysan schiebt den „Recognizers“–Evening Talk nach vorne, damit der geplante OOP–Workshop alle Fakten auf dem Tisch hat. Die Recognizers haben wir auch schon in der VD gehabt.

Bernd Paysan initiiert den Workshop zum Thema „Forth meta object protocol“, bei dem es darum geht, welche Konzepte man braucht, um damit verschiedene Forth–OOP–Systeme auf eine gemeinsame Basis zu setzen. Denn die Diskussion um Forth–OOP ist immer noch von „braucht man nicht“ und „mache ich anders“ geprägt. Die Recognizer für die Leute, die unbedingt den Parser ändern müssen, und die Verwendung von Methoden für Method dispatch und Erzeugung, die Andrew Haley vorschlägt, sollten aber die unterschiedlichen Konzepte zumindest auf eine gemeinsame Basis stellen, an der der Maintainer des Systems dann die Performance tunen kann. Dadurch können dann die eigentlichen OOP–Implementierungen schlank und einfach wartbar bleiben.

Bernd Paysan berichtet am nächsten Morgen über den Stand der Dinge bei „net2o — from Fiction to Reality.“ Dabei wird sowohl ein praktikabler Lösungsansatz zur Flusskontrolle und ein performanter Ansatz zur Kryptographie thematisiert.

Bill Stoddard hat mit „Forth Local Variable Semantics“ noch einen Beitrag der eher theoretischen Natur.

Ian van Breda möchte noch einige Kommentare zum Forth200x–Standard loswerden. Da der Standard jetzt fertig ist, kommt er ein bisschen spät. Aber es gibt ja noch einen nächsten Standard... und eine Public–Review–Periode, in der man seine Kommentare zum Standard loswerden kann.

Awards Am Schluss gibt's dann noch Preise für die Vorträge. Bronze geht an Bernd Paysan, für die Neuerfindung des Internets, Silber an Dirk Brühl, für seine 4€4th–Demonstration, und Gold an Andrew Read, für seinen N.I.G.E.–Prozessor. Alle drei Medaillen sind allerdings in Wahrheit aus Schokolade.

Literaturverzeichnis

[1] Forth 200x Standards Committee #8, *Forth 2012 RC 1,* http://www.forth200x.org/documents/forth-rc1.pdf

[2] Forth 200x Standards Committee #8, *Draft Minutes 2012,* http://www.forth200x.org/meetings/minutes2012.pdf

[3] Bernd Paysan, *Semantics,* VD 2012/02

[4] *EuroForth 2012 Proceedings,* http://www.complang.tuwien.ac.at/anton/euroforth/ef12/papers/

net2o — Das Internet neu erfinden, Teil 1

Bernd Paysan

net2o, kurz für „Internet 2.0“, ist ein Netzwerkprotokoll, das verschiedene Schwächen des aktuellen, und zum Teil 30 Jahren alten Internets behebt. Ich will in einer Artikelserie dazu mehr Details veröffentlichen. In dieser Folge geht es um das Paket–Format, die Kommando–Sprache und die Flusskontrolle.

Einleitung

Das Internet neu erfinden — geht das überhaupt? Ja, weil das Internet by Design ein Netz von Netzen ist, und deshalb offen für Neuerungen und Veränderungen ist. Damit kann man die beiden Hauptbedingungen für Änderungen erfüllen: Es muss ohne Investitionen funktionieren, und es muss jedem unmittelbar einen Vorteil bieten. Nicht kompatible Änderungen, wie schon IPv6 zeigt, die beide Bedingungen nicht erfüllen, setzen sich nicht durch, selbst wenn sie grundsätzlich notwendig sind.

Wie man forthige Netzwerke macht, hat Heinz Schnitter schon Mitte der 80er mit ONF gezeigt [1]; net2o greift einige wesentliche Ideen aus ONF auf, und verbindet sie mit anderen Konzepten, die in einem geschlossenen Netzwerk wie dem Beschleunigerlabor natürlich unnötig sind, wie etwa Verschlüsselung.

Was ist also kaputt am Internet, was muss besser gemacht werden? Dazu zunächst mal das, was erhalten werden muss am Internet, weil es nämlich eine gute Idee war:

- Frei und offen — das steht überhaupt nicht zur Debatte
- Paket–Orientierung: Die Daten werden in einzelne Pakete zerlegt, und so an den Empfänger geschickt
- Verbindungsloses Netzwerk: Jedes Paket enthält eine Zieladresse; die Verbindung muss nicht vorher angemeldet werden, sondern Pakete können ad hoc versendet werden. Das Herausfinden der Zieladresse ist ein davon getrennter Schritt.

Diese drei Prinzipien sind wesentlich für den Erfolg des Internets. Doch das Internet wurde in den 70ern und 80ern entwickelt, und viele Annahmen von damals sind heute einfach nicht mehr richtig. Allerdings: ONF ist auch aus den 80ern, und trotzdem wert, Betrachtung zu finden. Was ist also falsch oder verbesserungswürdig?

- Path–Switching statt Routing: Aktuell muss für jedes Paket dynamisch neu eine Route berechnet werden. Das kostet jedes Mal Zeit oder Geld (das man in teure Router–Hardware investiert), und die Routing–Tabellen werden auch nicht kürzer. Da die meisten Verbindungen viele Pakete austauschen, ist es sinnvoll, diese Berechnung nur einmal durchzuführen, und die eigentlichen Pakete zu switchen. Da das netzwerkübergreifend ist, also über einen Pfad.
- Die TCP–Flusskontrolle hat ein Problem, das „Buffer Bloat“ genannt wird. Der Algorithmus hat noch weitere Probleme mit nicht sonderlich zuverlässigen Netzwerken (den berühmt–berüchtigten WLAN–Kabeln), die heute sehr populär sind. Es muss also ein neuer her. Dazu ist TCP auch nicht im Ansatz echtzeitfähig, erschwert also heute populäre Medien–Anwendungen.
- Es gibt mit UDP einen leichtgewichtigen Zugang zu IP, den man generell für alle alternativen Protokolle nutzt, weil er durch alle Router geht; UDP ist aber kein voll funktionales Protokoll im eigentlichen Sinne. Auch ich verwende UDP, um über existierende IP–Netzwerke zu gehen.
- Verschlüsselung gilt als „zu schwierig, zu kompliziert,“ wird bei den meisten Protokollen als optionales Feature hinzugefügt, und funktioniert, selbst wenn es gemacht wird, eher selten so zuverlässig wie gewünscht.
- Der Versuch einer Public–Key–Infrastruktur, der mit SSL unternommen wurde, kann als gescheitert angesehen werden. Es muss also ein neuer Versuch her.

Die ganze Herangehensweise läuft mit dem Forth–Ansatz, das Problem im Wesenskern zu verstehen, sodass es einfach implementiert werden kann, und dabei traditionelle Herangehensweisen über den Haufen zu werfen. In der Regel reicht es, einfach alles genau „verkehrt herum“ zu machen, dann passt es.

Net2o soll natürlich skalieren, also von ziemlich klein (was heute schon größer ist als zu den Anfangszeiten von TCP/IP) zu ganz groß, und es soll einfach implementierbar sein, auch in Hardware, wenn das zum Beschleunigen sinnvoll ist.

Paket–Format

Das Format eines net2o–Pakets ist recht einfach:

Die Abstraktionsebene, die ich hier verwende, ist die eines Speichers, d.h. das Paket hat zwangsläufig eine Speicheradresse (64 Bit, auf seine Größe aligned).

Damit es seinen Weg durchs Netz findet, ist davor ein Switch–Pfad, mit 128 Bits. Das Switching ist einfach: Jeder Hop guckt sich den vordersten Teil des Pfads an, entscheidet danach, wie's weitergeht, nimmt dieses Teilstück weg, und hängt hinten bit–reverse einen Rückwärtspfad an. Am Empfänger angekommen, muss der nur den ganzen Pfad umdrehen, und kann so die Antwort zurückschicken.

Und davor gibt es noch zwei Bytes mit Flags und einer Längenangabe. Die Länge der Daten ist $64 * 2^n$, $n \in [0..15]$, also von 64 Bytes bis 2 MB in Zweierpotenzen. Sinnvoll ist heute maximal 8 kB, in LANs mit Jumbo–Frames, aber das Protokoll soll ja auch für die nächsten 30 Jahre halten. Byteweise Granularität der Paketgröße ist heute IMHO nicht mehr nötig, sie stört auch bei Verschlüsselung, da man aus der exakten Länge einer Datenübertragung durchaus Rückschlüsse ziehen kann, und es gibt bereits Attacken, die das ausnutzen. Angesichts der eher zögerlichen Verlängerung von Paketen über die letzten 30 Jahre, und der Tatsache, dass das Switching von net2o wenig Aufwand erfordert, also ein hoher Durchsatz an Paketen möglich ist, halte ich die Maximalgröße von 2 MB für mehr als ausreichend. Eine Nutzlast von weniger als 64 Bytes geht wegen des verwendeten Verschlüsselungsalgorithmus nicht.

Der net2o–Header sieht als Forth200x–Struktur dann wie folgt aus

```
begin-structure net2o-header
    2 +field flags
   16 +field destination
    8 +field addr
end-structure
```

Am Ende eines jeden Pakets ist noch eine kryptographische Checksumme, die 16 Bytes (128 Bits) groß ist. Das reicht, um die Authentizität und Integrität eines Pakets zweifelsfrei zu belegen. Der gesamte Overhead des Protokolls selbst (ohne zusätzliche Verpackung in UDP und IP) sind also 42 Bytes. Das Alignment ist so gewählt, dass man einen 14–Bytes–Ethernet–Header davorsetzen kann; etwas, was in eingebetteten Implementierungen sicher vorkommen wird.

Kommandos

Jede net2o–Verbindung baut zwei Speicherbereiche (mappings) auf: Eine für Kommandos, eine für Daten. Das Ausführen entfernter Kommandos ist nichts Ungewöhnliches, nur im aktuellen Internet hat jedes Protokoll seine eigene Kommandosprache, und in der Regel wird pro Paket genau *ein* Kommando verschickt — auch wenn das dann auf beliebig komplexen Daten operieren darf. Ich mache das anders, die Operationen sind grundsätzlich auf sehr einfachen Datentypen (ein paar wenige Zahlen oder Strings) und dafür darf man relativ viele in ein Kommando–Paket packen.

Net2o verwendet Kommandos auch für weiter unten liegende Protokoll–Layers, wenn man der Nummerierung des ISO–OSI–Stacks folgt; die Kommandos sind tokenisiert für eine Stack–Maschine. Ich implementiere ad hoc die Kommandos, die ich gerade brauche; die Sprache ist im Moment also nicht vollständig. Es ist auch kein Forth im eigentlichen Sinne, es gibt weder Stack–Operationen, noch kann man on–the–fly neue Definitionen machen. Das kommt vielleicht noch, grundsätzliche Bedenken dagegen habe ich nicht.

Da das eine Sprache ist, deren Ausführung von einem entfernten Rechner angestoßen wird, muss jeder Befehl entsprechend überprüft werden, und zwar bezüglich einer recht restriktiven Semantik. Man darf nicht Beliebiges machen, sondern nur, was im Kontext der Verbindung erlaubt ist.

Die Kommandos sind tokenisiert — Heinz hat auf seinem ONF einfach Quelltext herumgeschickt, aber da viele net2o–Kommandos von Programmen erzeugt werden, ist das eher unpraktisch. Zum Debuggen gibt es einen De–Tokenizer, der ist unverzichtbar. Ich verwende zur Codierung ein von Googles protobuf [2] abgeleitetes Format. Google verwendet protobuf zum Austausch strukturierter Daten, ich verwende es zum Austausch tokenisierten Codes, also ziemlich anders. Erhalten bleibt, wie Zahlen codiert werden: Sie werden dazu in 7–Bit–Stücke zerlegt, und der Reihe nach vom höchstwertigen beginnend als Bytes in den Speicher gelegt. Das Bit 7 ist 0, wenn es sich um das niederwertigste Teilstück handelt (das markiert dann das Ende der Zahl), ansonsten 1. Das Format ist etwas einfacher zu codieren und decodieren als UTF–8, kann prinzipiell beliebig große Zahlen darstellen (in der Praxis beschränke ich mich auf 64 Bits), und ist auch etwas kompakter. Codieren und decodieren in Forth ist recht einfach (hier nur der Code für 64–Bit–Maschinen):

```
: p@+ ( addr -- u64 addr' )  >r 0
  BEGIN  7 lshift r@ c@ $7F and or
         r@ c@ $80 and  WHILE
         r> 1+ >r  REPEAT  r> 1+ ;
: p-size ( u64 -- n )
\ to speed up: binary tree comparison
\ flag IF  1  ELSE  2  THEN  equals  flag 2 +
  dup    $FFFFFFFFFFFFFF u<= IF
      dup       $FFFFFFF u<= IF
          dup      $3FFF u<= IF
              $00000007F u<= 2 +  EXIT  THEN
          $00000001FFFFF u<= 4 +  EXIT  THEN
      dup   $3FFFFFFFFF u<= IF
          $00007FFFFFFF u<= 6 +  EXIT  THEN
      $00001FFFFFFFFFF u<= 8 +  EXIT  THEN
  $000007FFFFFFFFFFFFFF u<= 10 + ;
: p!+ ( u64 addr -- addr' )
  over p-size + dup >r >r
  dup $7F and r> 1- dup >r c!  7 rshift
  BEGIN  dup  WHILE  dup $7F and $80 or
         r> 1- dup >r c! 7 rshift  REPEAT
  drop rdrop r> ;
```

Vorzeichenbehaftete Zahlen werden nicht im Zweierkomplement übertragen, sondern mit dem LSB als Vorzeichenbit. Die Umwandlung läuft also so (n hier die vorzeichenbehaftete Zahl, u die vorzeichenlose, aber codierte):

```
: u>n ( 64u -- 64n )
  dup 2/ swap 1 and IF negate THEN ;
: n>u ( 64n -- 64u )
  dup 0< 1 and swap abs 2* or ;
```

Bei vielen Operationen gibt es sowohl lokal als auch remote etwas zu tun, weshalb man beim Erzeugen von

Kommando–Paketen Wörter benutzt, die beide Aktionen sinnvoll kontrollieren. Ich will das am Beispiel des Copy–Befehls erklären, der das Kopieren einer entfernten Datei auf den lokalen Rechner anstößt (das ist nur ein Teil der gesamten Aktion)

```
: n2o:copy ( addrsrc us addrdest ud -- )
  2swap $, r/o ulit, file-reg# @ ulit,
  open-tracked-file
  file-reg# @ save-to
  1 file-reg# +! ;
```

`$,` und `ulit,` legen Literals ab. `open-tracked-file` öffnet eine Datei und übermittelt anschließend ihre Größe an den Client. Der legt mit `save-to` eine lokale Datei an, die alle zu diesem File–Handle gehörenden Datenblöcke aufnehmen wird. Damit da auch was passiert, muss man nur noch `slurp-all-tracked-blocks` sagen.

Vielleicht noch eine Anmerkung, wie man Kommandos so konstruiert, dass sie gebündelt übermittelt werden können und man keine unnötigen Round–Trip–Delays hat:

`open-file`, egal ob in Forth oder C, nimmt einen String und gibt ein File–Handle, das man für weitere Zugriffe verwenden kann, zurück. Das ist schlecht, denn wenn wir das Paket bauen, wissen wir dieses File–Handle ja noch nicht. Also machen wir es anders: `open-tracked-file` nimmt einen String und eine Nummer, und man kann diese Nummer dann als File–Handle verwenden. Die Nummer ist beim Zusammenbau des Pakets schon bekannt.

Die Namen der Funktionen werden sich wahrscheinlich noch ändern, das Tracking im Namen bezieht sich darauf, dass Änderungen der Dateiposition übermittelt werden, also beide Seiten Bescheid wissen.

Flusskontrolle

In einem Netzwerk darf man Daten nur so schnell senden, wie sie die Gegenstelle aufnehmen kann (inklusive aller Flaschenhälse dazwischen). Wenn man zu viele Daten schickt, füllen sich die Puffer...

Die ursprüngliche Annahme von TCP war, dass RAM knapp und teuer ist, und deshalb Router nur sehr wenig davon besitzen. TCP ermittelt durch Vergrößern des TCP–Windows, wie viel das ist, und reduziert die Datenrate, wenn Pakete verloren gehen (weil mutmaßlich der Puffer voll ist). Tatsächlich ist heute RAM natürlich spottbillig, und viele Router haben Puffer für mehrere Sekunden, die dann auch von TCP bis an den Rand gefüllt werden (siehe Abbildung 1). Das erhöht die Latenz und sorgt dafür, dass parallele Verbindungen (etwa zu einem großen Download) sehr langsam und träge sind.

Auf der anderen Seite nutzen wir heute viele drahtlose Netzwerke, deren Zuverlässigkeit nicht sonderlich hoch ist. Da geht schon mal ein Paket verloren, ohne dass es der Sender bemerkt. Zur Erinnerung: In den dicken gelben Koax–Kabeln gab es auch Kollisionen, denn sie waren auch ein geteiltes Medium. Aber die bekam der Ethernet–Adapter mit, und man konnte die Sendung wiederholen. Die Störungen, die ein WLAN–Paket so weit zerstören, dass es nicht mehr zusammengesetzt werden kann (trotz massiven Einsatzes von Vorwärts–Fehler–Korrektur FEC) bleiben dem Sender aber verborgen.

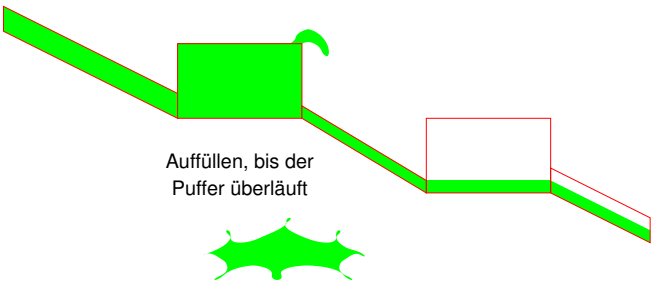


Abbildung 1: Buffer Bloat

Darauf reagiert der TCP–Algorithmus erst mal mit der Halbierung der Übertragunsrate, bzw. dem TCP–Window, also der Menge Daten, die übertragen werden, bevor man anfängt, auf ein Acknowledge zu warten. Die heute übliche Kombination aus drahtlosem Netzwerk und Geräten mit großen Puffern ist also alles andere als optimal für TCP — bei Paketverlusten wird die Verbindung unnötig langsam, gehen die Pakete nicht verloren, erhöht sich die Latenz auf unerträgliche Werte.

Es muss also ein neuer Algorithmus her, der die tatsächlich verfügbare Datenrate anders ermittelt, denn Ziel ist es, die Puffer leer zu lassen und die verfügbare Bandbreite maximal zu nutzen (siehe Abbildung 2). Ansätze gibt es schon, das vom µTorrent–Protokoll verwendete LEDBAT z.B., oder CurveCP von Dan Bernstein. Leider funktionieren beide meiner Meinung nach nicht gut genug.

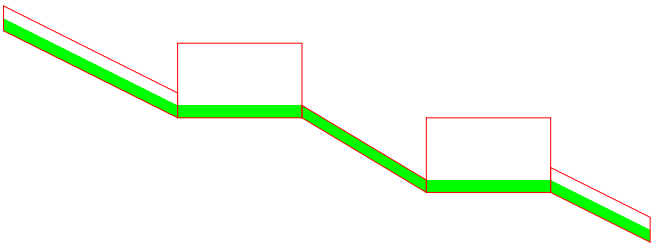


Abbildung 2: Richtig kalkulierte Datenrate

Wie misst man die erreichbare Bandbreite aus? Ich verwende dazu ein paar kurze Bursts, und sende dabei jeweils 8 Pakete so schnell es geht an den Empfänger (siehe Abbildung 3). Der misst die Ankunftszeit der Pakete, und rechnet daraus die erreichbare Datenrate aus. Dabei gehe ich davon aus, dass die Pakete normalerweise in der gleichen Reihenfolge ankommen, wie sie versandt wurden; diese Annahme trifft zumindest meistens zu.

Verloren gegangene Pakete werden in die Rechnung einbezogen, es wird gemutmaßt, dass eine um den Faktor der Paketverluste geringere Datenrate zu keinem Paketverlust führen würde. So sieht also die Formel zur Berechnung der Datenrate wie folgt aus:

```
: calc-rate ( -- )
  delta-ticks @ tick-init 1+ acks @ */
  lit, set-rate ;
```

Und der Server implementierte `set-rate` einfach als

```
: set-rate ( rate -- )  ns/burst ! ;
```

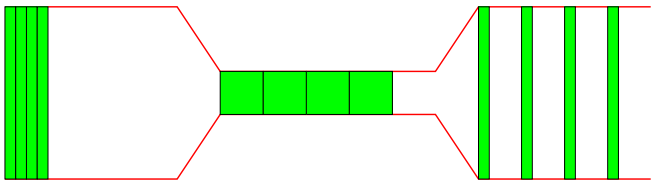


Abbildung 3: Messe den Flaschenhals mit einem Paket–Burst

Denn die Datenrate wird als Zeit für das Senden eines Bursts gespeichert, Einheit: Nanosekunden.

Leider funktioniert das in der Praxis dann doch nicht so einfach. Denn die Router haben eine einfache FIFO–Policy, senden eingehende Bursts also so weiter, wie empfangen. Sind also gleichzeitig zwei net2o–Datenströme unterwegs, ergibt sich folgendes Bild (Abbildung 4):

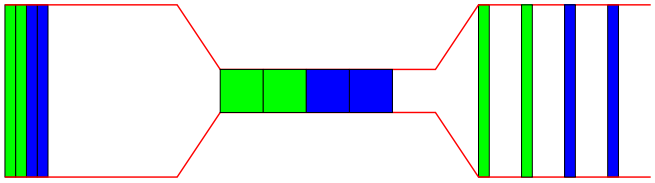


Abbildung 4: Ein FIFO–Router liefert Pakete so aus, dass das Doppelte der verfügbaren Datenrate berechnet wird

An der Stelle ist ein Ansatzpunkt, wie man das Problem lösen kann, indem man das Verhalten der Router ändert, nämlich, dass sie eine faire Queue für alle Verbindungen bilden, also jeweils innerhalb der Verbindung FIFO–Charakteristik verwenden, und zwischen den Verbindungen ein Round–Robin–Verfahren. Das würde auch die Buffer–Bloat–Problematik in aktuellen Netzen entschärfen, da richtig implementierte Protokolle um die Staus herumgeroutet würden, siehe Abbildung 5.

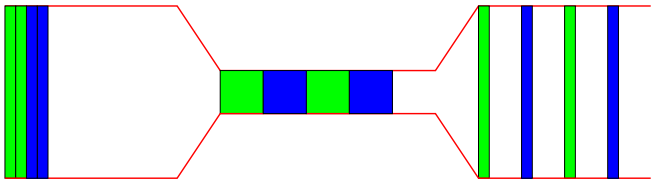


Abbildung 5: Ein fairer Router ermöglicht das korrekte Messen der verfügbaren Bandbreite

Da wir diese verbesserten Router noch nicht haben, muss mein Algorithmus etwas aufgebohrt werden. Er soll primär weiterhin die erzielbare Datenrate ermitteln, aber anhand der Zeit zwischen Senden und Empfang ermitteln, ob es dazwischen eine unbemerkte Stau–Situation gibt. Ich messe also, wie lange die ersten Pakete eines Bursts zum Empfänger brauchen, und beobachte das. Ich verwende für diese zusätzliche Zeit den Begriff „Schlupf“ (slack), der eigentlich nicht entstehen sollte.

Berücksichtigt werden sowohl ein Anwachsen des Schlupfs als auch die absolute Größe. Ersteres geht linear in die Datenrate ein, und zwar unter der Annahme, dass alle Pakete, die unterwegs sind, zu schnell abgeschickt wurden, und entsprechend korrigiert werden muss, um ein Anwachsen des Schlupfs zu verhindern — und dass man diesen Wert langsam abklingen lassen muss, damit er seine Wirkung erzielt. Letztere geht exponentiell ein, wobei ich für Faktor 2 entweder 20 ms oder ein Viertel des bisher gemessenen maximalen Schlupfs verwende (was immer größer ist).

Damit war nach langem Experimentieren eine Lösung da, die sowohl sehr schnell auf die tatsächlich verfügbare Bandbreite konvergiert (nach einem Round Trip Delay plus der Übertragungsdauer des ersten Bursts), und die auch bei Engpässen schnell adaptiert, und den Puffer leer hält, siehe Abbildung 6.

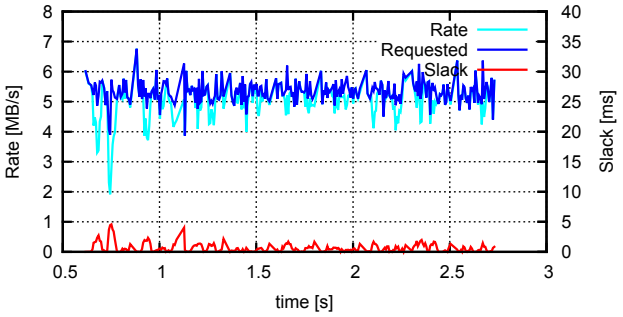


Abbildung 6: Eine Verbindung über VDSL

Nicht ganz so schön, aber immer noch brauchbar, sieht das bei vier parallelen Verbindungen über ein WLAN aus (Abbildung 7, gezeigt wird genau eine Verbindung):

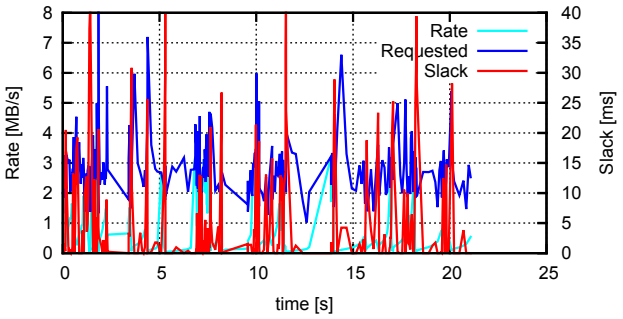


Abbildung 7: Eine von vier Verbindungen über WLAN

Im nächsten Teil geht es dann darum, wie man fehlende Pakete nochmal überträgt, und wie die Verschlüsselung aussieht. Den wachsenden Code gibt's in einem Fossil–Repository [3].

Literaturverzeichnis

[1] Heinz Schnitter, *Open Network Forth,* Forth–Tagung 2009

[2] Kenton Varda et al, *Protocol Buffers,* http://code.google.com/p/protobuf/

[3] Bernd Paysan, *net2o fossil repository,* https://fossil.net2o.de/net2o

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 (p)
`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 Küstenforth
Klaus Schleisiek
Tel.: (0 40) – 37 50 08 03 (g)
`kschleisiek@send.de`
Treffen 1 Mal im Quartal
Ort und Zeit nach Vereinbarung (bitte erfragen)

Mainz Rolf Lauer möchte im Raum Frankfurt, Mainz, Bad Kreuznach eine lokale Gruppe einrichten.
Mail an rowila@t-online.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`

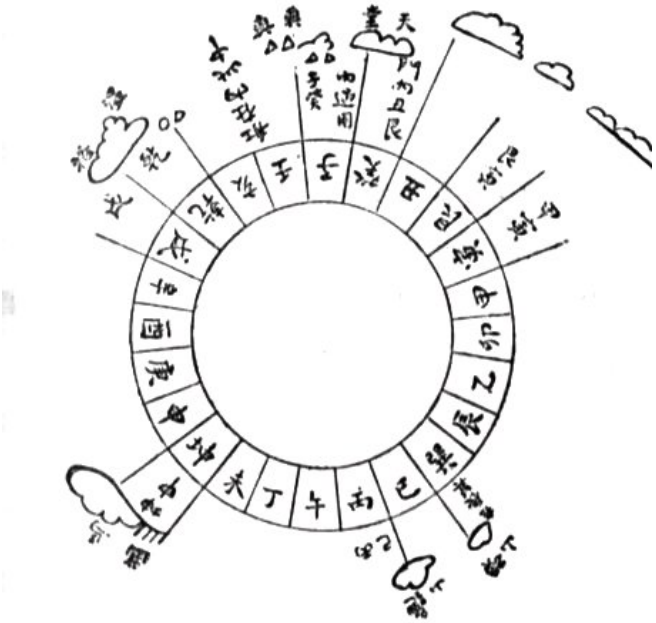
Spezielle Fachgebiete

FORTHchips (FRP 1600, RTX, Novix)	**Klaus Schleisiek–Kern** Tel.: **(0 40) – 37 50 08 03** (g)
KI, Object Oriented Forth, Sicherheitskritische Systeme	**Ulrich Hoffmann** Tel.: **(0 43 51) – 71 22 17** (p) Fax: **– 71 22 16**
Forth–Vertrieb volksFORTH ultraFORTH RTX / FG / Super8 KK–FORTH	Ingenieurbüro **Klaus Kohl–Schöpe** Tel.: **(0 70 44)–90 87 89** (p)

Termine

Mittwochs ab 20:00 Uhr
Forth–Chat IRC `#forth-ev`

02.–03. Februar 2013 FOSDEM, Brüssel
16.–17. März 2013, Chemnitzer Linuxtage
19.–21. April 2013 Forth–Tagung
22.–25. Mai 2013 Linuxtag, Berlin
09.–10. November 2013 OpenRheinRuhr, Oberhausen



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

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

Forth–Tagung 2013

Forsthaus Graseck, Graseck 4, 82467 Garmisch–Partenkirchen
vom 19. bis 21. April 2013

Organisator:

Heinz und Ulrike Schnitter, Bernd Paysan, Email: `tagung@forth-ev.de`

Termine:

31.1.2013 Frühanmeldeschluss
28.2.2013 Anmeldeschluss für Spätanmelder
1.3.2013 Redaktionsschluss für Konferenz–Paper, die in der VD 2013/01 veröffentlicht werden

Programm:

Donnerstag, 18.4.2013 Treffen der Frühankommer
Freitag, 19.4.2013 15 Uhr Beginn der Tagung
Vorträge & Workshops
Samstag, 20.4.2013 9 Uhr Vorträge & Workshops
Exkursion Partnachklamm
Sonntag, 21.4.2013 9 Uhr Jahresversammlung
13 Uhr Ende der Tagung