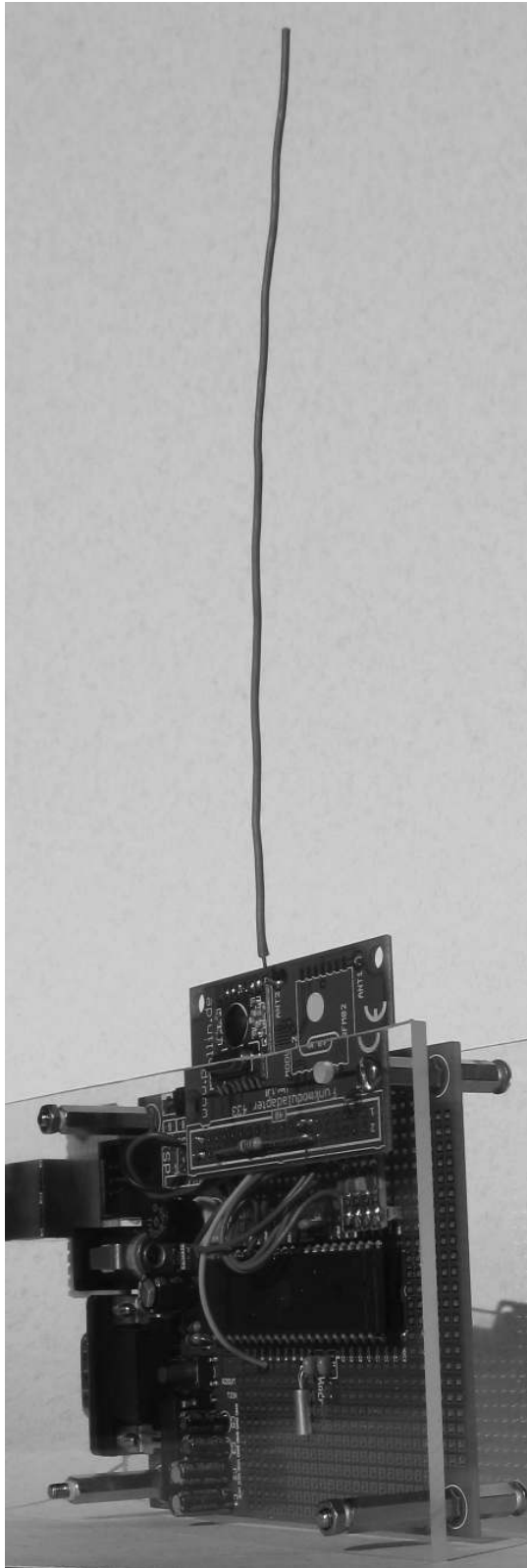




*für Wissenschaft und Technik, für kommerzielle EDV,
für MSR-Technik, für den interessierten Hobbyisten*



In dieser Ausgabe:



Forthbildung

Vereinsinternes

Funksensoren belauschen

EuroForth 2011 in Wien

Design Pattern und objekt-orientierte
Programmierung

Kontrollstrukturen als
Colon-Definitionen

Neues von VFX Forth

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.

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

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.

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“!

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

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.

Leserbriefe und Meldungen5
Forthbildung6
<i>Erich Wälde, Martin Bitter, David Kühling, Carsten Strotmann</i>	
Vereinsinternes13
<i>M. Anton Ertl, Ewald Rieger</i>	
Funksensoren belauschen16
<i>Erich Wälde und Martin Bitter</i>	
EuroForth 2011 in Wien25
<i>Bernd Paysan</i>	
Design Pattern und objekt-orientierte Programmierung28
<i>Bernd Paysan</i>	
Kontrollstrukturen als Colon-Definitionen35
<i>Fred Behringer</i>	
Neues von VFX Forth41
<i>Stephen Pelc</i>	



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

Manuskripte und Rechte

Berücksichtigt werden alle eingesandten Manuskripte. Leserbriefe können ohne Rücksprache wiedergegeben werden. Für die mit dem Namen des Verfassers gekennzeichneten Beiträge übernimmt die Redaktion lediglich die presserechtliche Verantwortung. Die in diesem Magazin veröffentlichten Beiträge sind urheberrechtlich geschützt. Übersetzung, Vervielfältigung, sowie Speicherung auf beliebigen Medien, ganz oder auszugsweise nur mit genauer Quellenangabe erlaubt. Die eingereichten Beiträge müssen frei von Ansprüchen Dritter sein. Veröffentlichte Programme gehen — soweit nichts anderes vermerkt ist — in die Public Domain über. Für Text, Schaltbilder oder Aufbauskizzen, die zum Nichtfunktionieren oder eventuellem Schadhafwerden von Bauelementen führen, kann keine Haftung übernommen werden. Sämtliche Veröffentlichungen erfolgen ohne Berücksichtigung eines eventuellen Patentschutzes. Warennamen werden ohne Gewährleistung einer freien Verwendung benutzt.

Liebe Leser,

als auf der Tagung der Forth-Gesellschaft Helfer für das Erstellen der Vierten Dimension gesucht wurden, sagte ich: *ich helfe mit*. Und ruck zuck hatte ich Ende August den Staffelstab, von Bernd Paysan kommend, auf dem Tisch. Die Aufgabe, vierzig Seiten zu füllen, erschien mir fast aussichtslos. Aber jetzt ist das Heft doch dick genug für den Druck. Eigentlich erstaunlich.

Das Editieren ist eine lehrreiche Angelegenheit: ich lernte, dass copy&paste aus Adobe-Reader schon mal eine Datei in einem merkwürdigen Encoding produziert: in-is13194-devanagari-unix, das kommt mir ziemlich Indisch vor [1]. Ich lernte, dass Fred Behringer der eifrigste Korrekturleser ist, den man sich wünschen kann. Ich lernte, dass Michael Kalus immer die Augen und Ohren aufsperrt und so eine Menge Information heranschafft. Dass Bernd Paysan ein TeX-Guru ist, wusste ich schon. Also ist auch dieses Heft eine gemeinsame Anstrengung. Vielen Dank an alle Autoren und Mithelfer!

Wann, liebe Leser, haben Sie zum letzten Mal einer anderen Person gezeigt, was Sie im (hoffentlich) stillen Kämmerlein treiben — sooft die Aufgaben des Alltags ein wenig Zeit übrig lassen? Ach, schon so lange her? Ich glaube, das ist ein kleines Problem. Bloß, weil Sie oder ich mehr oder weniger regelmäßig an einem elektronischen Lieblingsprojekt herumbasteln, ist das damit verbundene Wissen noch lange nicht für andere Menschen zugänglich.

Wissen verbreitet sich zwar sehr gut über gedruckte oder heutzutage digitale Medien. Aber noch besser verbreitet es sich durch *Vorleben*. Dem Nachbarn mal erklärt, wie die Messwerte der gekauften Technoline-Sensoren dauerhaft in eine Datenbank kommen? Der skeptischen Nachbarin gezeigt, dass es letztes Jahr im Oktober schon deutlich kühler war? Ich kann nur jedem empfehlen, anderen Leuten zu zeigen, was man im Bastelkämmerlein so treibt. Dabei ist es sogar erlaubt, praktisch funktionierende Lösungen zu betreiben, die programmieretechnisch oder stilistisch unausgereift sind. Ein Programm, das nicht laufen darf, weil es noch nicht perfekt ist, ist eine vertane Chance. Traut man sich aber mit den Unzulänglichkeiten ans Tageslicht, dann finden sich vielleicht Helfer oder Fans an ganz unerwarteten Stellen.

Ganz in diesem Sinne waren im November ein paar Aktive unterwegs, um anderen Leuten von ihren Forth-Basteleien zu erzählen: Der Artikel *Forthbildung* auf Seite 6 erzählt ausführlich davon. Liebe Leser, wenn Sie in Ihrem Umfeld Personen kennen, die so etwas lernen wollen, dann fragen Sie uns. Wir können Ihnen helfen, eine *Forthbildung* durchzuführen. Wer mitmacht, wird feststellen, dass er oder sie genügend Wissen hat, um anderen zu zeigen, wie das mit Forth so funktioniert. Dabei kommt der Spaß auf keinen Fall zu kurz — trauen Sie sich!

Ich gebe hiermit den Staffelstab an Ulli Hoffmann weiter. Der Artikel über *fossil* ist leider in zu vielen Reisetunden für Carsten Strotmann stecken geblieben, aber er kommt! Viel Vergnügen beim Lesen,

Erich Wälde — *the guy who calls himself an Anfänger*

1. <http://de.wikipedia.org/wiki/Devanagari>

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web-Seite des Vereins herunterladen.

<http://www.forth-ev.de/repos/vd/2011-04/>

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

Ulrich Hoffmann Kontakt: Direktorium@Forth-ev.de

Bernd Paysan

Ewald Rieger



RaspberryPi

Auf `comp.lang.forth` wurde neulich gefragt: "Wie nahe kommt man ran an das blanke Metall (bare metal) einer MCU mit Hilfe von Forth?" Die Antwort war: "...kommt drauf an!" Klar kann man mit Forth theoretisch bis in die Bits und Bytes vordringen. Praktisch sieht das dann schon mal anders aus.

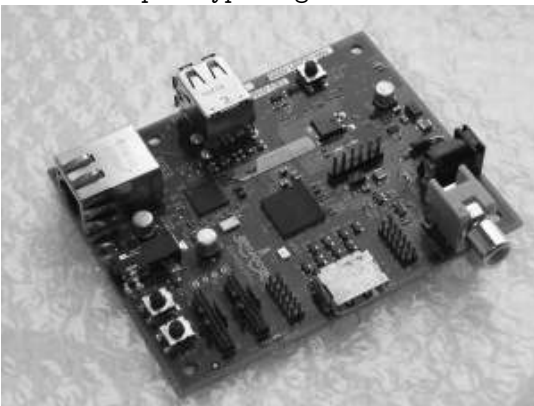
Der RaspberryPi wird einen *großen* (>18MB), rein binären und geheimen, mit Eigentumsvorbehalten versehenen GPU-Treiber haben, der für die Operationen im System verantwortlich ist, und Linux, oder was auch immer, wird den Zugriff auf die Maschinenebene nur durch APIs bekommen (also wie bei der Sony-Playstation 2&3). Das ist auch nicht so sehr nah dran an dem "Metall" selbst.

Also, wenn man erst mal dafür ein Forth übersetzt hat, mit einem Wort, das die API-Aufrufe kann, und den basalen API-Aufrufen selbst, und das läuft, ist das so nahe dran am "Metall", wie man nur kommen kann?

Nun, das wäre eine Art, das zu machen, aber noch nicht das, was ich "pures Metall" nennen würde. Wie Stephen schon sagte, technisch ist es möglich, das Linux-OS durch ein reines Forth-System (auf ARM-Basis) zu ersetzen. Sowohl VFX von MPE als auch SwiftX von FORTH Inc. leistet so eine Übersetzung (*cross-compilation*) auf diesen Zielprozessor. Um das zu machen, müsstest du aber auch deine eigenen Treiber für die Module der Peripherie schreiben. Die größte Herausforderung, solche Software zu ersetzen, liegt überhaupt nicht auf der Seite der verwendeten Werkzeuge, sondern ganz darin begründet, ob die Informationen darüber, wie die Hardware arbeitet und wie sie zu handhaben wäre, geheim gehalten wird und unzugänglich bleibt. Für Architekturen, für die kein Wall aus geheimer Software aufgebaut wurde, ist es generell einfacher, eigene native Treiber zu schreiben, als mit einem Betriebssystem zu ringen, das den Hausherrn spielt. Obschon so ein Betriebssystem dem Gerät auch eine allgemeinere Anwendbarkeit verleiht, was es schon mal lohnend machen kann, es auch zu benutzen.

Cheers, Elizabeth (2.9.2011)

Raspberry Pi – An ARM Linux box for \$25. Take a byte!
<http://www.raspberrypi.org/> mk



Zur Entwicklung von Forth

Im März 1993 erschien jener Beitrag in den ACM-SIGPLAN-Notizen zum ersten Mal. Elizabeth D. Rather, Donald R. Colburn und Charles H. Moore zeichneten darin die Entwicklungsgeschichte dieser Werkzeugsammlung

des Herrn Moore nach. Elizabeth war der zweite Mensch überhaupt, der etwas in Forth programmiert hatte. Und Don Colburn war einer der ersten Forth-Anwender, Mitbegründer der Forth Interest Group (FIG) und trug zur Entwicklung des ersten freien (public-domain) figForth bei. Nun, Chuck Moore, heute Vorsitzender und technischer Direktor von Green Arrays, Inc., und Erfinder von Forth, ist unseren alten Lesern ja bestens bekannt. Neue Leser und andere Interessierte finden die ganze Geschichte nun auch auf den Seiten der Forth Inc. vor: *The Evolution of Forth* <http://www.forth.com/resources/evolution/index.html> mk

Nachricht aus Übersee

Dear Fred,

vielen Dank für deine Nachricht. Wunderbar! Ihr strengt euch alle dermaßen stark an. Ich war seit Carstens Hiersein nicht mehr auf dem Treffen der Silicon Valley Forth Interest Group, aber ich bekomme regelmäßig (per E-Mail) Ankündigungen und sonstige interessante Neuigkeiten von der Gruppe. Die Zeit verfliegt so rasch und ich habe eine ganze Menge zu tun, um mich, wie man sagt, nicht unterkriegen zu lassen. Ich werde mir die neue digitale Version der Vierten Dimension auf dem Forth-ev-Webportal ansehen und bin darauf gespannt, was die diesmal 40 Seiten bringen. Meine allerbesten Wünsche dir und dem Rest der Gesellschaft!

Henry

(Übers.: Fred Behringer)

Neuigkeiten von amforth

Am 6.10.2011 wurde die Version 4.6 von amforth freigegeben. Neu sind drei Befehle zum Bearbeiten von einzelnen Bits: `bm-set`, `bm-clear` und `bm-toggle`. Sie sind in Assembler programmiert und arbeiten schneller als Forth Worte. Die drei neuen Worte benutzen eine Bit-Maske und die Speicheradresse als Argumente.

Für die Kommunikation mit SPI-Bausteinen gab es bislang das Wort `spirw`, welches ein Byte mit dem Baustein austauscht — für jedes geschriebene Byte erhält man gleichzeitig ein Antwortbyte zurück. Gelegentlich benötigt man aber den Austausch von zwei Bytes. In Anlehnung an die bekannten Worte `@` und `!` wurden daher zwei neue Worte definiert: `!@spi` und `c!@spi`, welche eine Zelle bzw. ein Byte mit einem SPI-Baustein austauschen.

<http://amforth.sourceforge.net>

ew

Die Dunbar-Zahl und der Pfau

Der Neurobiologe M.Spitzer ging in *Nervenheilkunde*, 9/2011, der Frage nach, warum wir Menschen so ein großes Gehirn haben. Und es scheint, dass die Fähigkeit zu Tratsch und Klatsch und ein großes Tam-Tam zu machen, uns das beschert hat. Hat nichts mit Forth zu tun, ist aber unterhaltsame Wissenschaft, und vielleicht inspiriert es den einen oder anderen unter Euch, auch mal hier zu schreiben. Schließlich ist die Forth-Gesellschaft auch nicht viel größer als so eine Urhorde. mk

<http://www.schattauer.de/index.php?id=94&L=0>

Fortsetzung Leserbriefe auf Seite 15



Fort**h**bildung

Erich Wälde, Martin Bitter, David Kühling, Carsten Strotmann

In den vergangenen Wochen waren einige Forth-Aktive unterwegs in Sachen Forth für alle. Dazu boten der Brandenburger Linux-Infotag in Potsdam (5.11.) [1], die OpenRheinRuhr in Oberhausen (12./13.11.) [10] sowie eine Einladung an die Hochschule in Augsburg (7.11.) [13] hervorragende Gelegenheiten. In allen drei Workshops wurde die gleiche Aufgabe bearbeitet: Schreibe ein Programm, welches einen beliebigen Text als Morsecode ausgibt, via Piepser oder LED.

Brandenburger Linux-Infotag 2011



Samstag Morgen um halb neun kamen Gido und ich an das Institut für Informatik der Universität Potsdam, Campus Griebnitzsee. Die erste Überraschung erwartete uns schon: Carstens Flieger fiel aus, also waren wir am Vormittag erstmal nur zu zweit und Carsten

war ganz umsonst so früh aufgestanden.

Damit unser Stand ohne Carstens Forth-Poster nicht so kahl aussah, puzzelten wir uns erstmal ein Ersatzposter aus A4-Ausdrucken zusammen. Glücklicherweise hatte die BeLUG (Berlin Linux User Group) einen Farb-Laserdrucker zum BLIT mitgebracht.

Ich hatte zur Standdekoration und Demonstration von Forth einen alten TI-92 [3] Taschenrechner mit installiertem Forth92 [4] am Stand (ein Uralt-Projekt aus meiner Schulzeit), und dazu einen NanoNote aus der Neuzeit mit installiertem Gforth und der Mandelbrot-Fraktal-Demo, wie in der VD 4/2010 *Forth auf dem Ben NanoNote* beschrieben [5]. Auf meinem Laptop lief die ganze Zeit Gforth mit dem Schachprogramm Brainless [6], welches eifrig gegen sich selbst Schach spielte. Es sollte zeigen, dass Forth auch für rechenintensive Aufgaben geeignet ist; aber so ein normaler Laptop war offenbar ganz und gar ungeeignet, irgendetwas Aufmerksamkeit auf sich zu ziehen. Auf Gidos Forth-losem Rechner wurde noch schnell ein Win32Forth [7] installiert, aber leider blieb bis zum Ende keine Zeit mehr, etwas Sinnvolles damit anzufangen.

Der Vormittag begann sehr ruhig mit gefühlt maximal 2-stelliger Besucherzahl auf dem ganzen BLIT, eher 1-stellig am Forth-Stand. Auffallend viele Besucher interessierten sich für Forth auf Controllern, also Carstens Schiene, und mussten auf den Nachmittag vertröstet werden.

Carsten kam mit einer Kiste Arduinos, pünktlich eine halbe Stunde vor Beginn seines 2-stündigen Forth-Workshops [2], der sehr gut besucht war.

Nachmittags wurde es merklich voller, die meisten Besucher interessierten sich nach wie vor für Carstens Arduinos. Am lebenden Exemplar führte er später Besuchern vor, wie er einfach per Forth-Konsole ein paar LEDs blinken lassen konnte — sogar mit dem Multi-Tasker. Einige Besucher (Studenten!) wollten ihren Augen kaum trauen, offenbar ist man ein derartig einfaches Arbeiten in der

Welt monströser, sogenannter integrierter Entwicklungs-umgebungen mit JTAG-Debugger nicht mehr gewohnt.

Auch (Noch-)Nicht-Programmierer wurden vom Forth-Stand angezogen. Sie zeichneten sich dadurch aus, dass sie Forth gegenüber sehr offen waren und das auch blieben, nachdem sie die Forth-Code-Beispiele auf dem Poster betrachtet hatten.

Einem konnte ich auch schnell mal am TI-92 zeigen, wie diese *interaktive* Programmierung aussieht. Zu meiner Überraschung konnte er mir sogar folgen, als wir ein kleines Miniprogramm an der Forth-Konsole zusammenbastelten. Ausführung links nach rechts, wie man es eingibt, nur ein Datentyp (was ein Datentyp ist, wäre wahrscheinlich sowieso schwer zu erklären gewesen) — mehr muss man ja eigentlich nicht wissen um nachzuvollziehen, was passiert.

Gegen Ende kam ich auch noch mit einigen der Organisatoren ins Gespräch. Offenbar war der BLIT ehrenamtlich von Brandenburger und Berliner Linux-User-Groups organisiert. Einer der Organisatoren (wahrscheinlich von der Brandenburger Linux User Group [8]), packte am Ende noch seinen Laptop aus, und zeigte mir ein Gforth-Programm, das er geschrieben hatte, um Forth kennenzulernen: eine Implementierung von Conways Game of Life [9]. Ich durfte auch mal durch den Quellcode gucken, aber leider blieb keine Zeit, mir eine Kopie zu ziehen, denn wir mussten uns langsam ans Aufräumen und Abbauen machen.

Alles in allem ein interessanter Tag. Fazit: Forth weckt Interesse und hat einen erkennbaren Bekanntheitsgrad, zumindest in Linux-Geek/Nerd-Kreisen. David

OpenRheinRuhr



Kurzfristig sprang ich als Helfer bei der Standbetreuung der Forth-Gesellschaft e.V. bei der OpenRheinRuhr [10] in Oberhausen ein. Ich war sogar pünktlich! Abbildung 1 zeigt mein erstes Namensschild. Netterweise bekam ich einen verbesserten Ausweis. Am Stand

wurde ich sofort von Carsten und Gido gefragt, ob ich einen Rechner dabei hätte. Hatte ich nicht! Carsten hatte zwar einen, benötigte den aber für seinen Workshop *Arduino — Ausbruch aus dem Compile-Upload-Debug-Kreislauf* [11] und verschwand. Gido lernte an seinem Rechner (Fernstudium), Carstens Frau las papierne (!) Literatur. Ich lehnte mich zurück und dachte:

Mist... hätte ich mir doch was zu arbeiten mitgebracht!
Ich befürchtete ernsthaft Langeweile. Es kam anders.



Abbildung 1: Manchmal fühle ich mich wie ein Holzfäller.

Carsten hat ein schönes, sehr gut lesbares Textposter drucken lassen, das von einigen Menschen gelesen wurde. Blickkontakt — *Hast Du Fragen?* Und es ging los! Eine junge Dame: *Jahrelang begleite ich meinen Liebsten zu Opensource-Veranstaltungen, aber das hier ist das erste Programm, bei dem ich etwas verstehe. Dass hier eine Waschmaschine gesteuert wird, verstehe sogar ich.* 20 Minuten über die *Einfachheit* von Forth geredet. Wie die Zeit doch verfliegt! Der nächste Besucher hatte Zeit und Lust, sich die Sprache erklären zu lassen, Beispiele zu verfolgen und über Low-Level und High-Level zu diskutieren. Er *verquatscht* sich sogar und kommt zu seinem Termin zu spät.

Dann jemand, der Forth aus seiner Jugend kennt und es toll findet, dass es immer noch Leute gibt, die sich damit beschäftigen. Danach ein Fachmann, der über Sicherheitsaspekte diskutiert. Ein Compiler bei anderen Sprachen fängt jede Menge Fehler ab. Das macht doch Programme sicherer! Diskussion darüber, dass Sicherheit beim Codieren vom Programmierer abhängt. Zwei Menschen, die es einfach nicht fassen können, dass eine Sprache ohne Variablentypen und Typwandlung auskommt. Schnell mit

```
variablenname @ dup . dup emit .bin
```

zeigt, dass es von meiner Entscheidung abhängt, was was ist. Langsames Begreifen in den Augen — ein Hauch von Freiheit! Hilfreich ist, dass Carsten mir seinen NanoNote dagelassen hat. Zum einen läuft darauf Gforth, zum anderen ist das Ding ein Blickfang. Man muss nicht wie früher aufpassen, dass man keinen Kaffee über die Tastatur schüttet, sondern man muss aufpassen, dass man das Ding nicht komplett in der Kaffeetasse versenkt. Falls es ein nächstes Mal gibt, werde ich mir das NanoNote wieder ausleihen und ein Arduino-Board dranhängen. Vielleicht auch einen kleinen Roboter?

Der Workshop war mit zeitweise 24 Teilnehmern bestens belegt. Die 3.5 Stunden waren genug, um das erste Morseprogramm im Detail zu besprechen. Die Teilnehmer waren sehr aufmerksam und stellten eine Menge guter Fragen.

Kurz vor 14 h wird es voll am Stand. Carsten kommt vom Workshop zurück und jede Menge Leute folgen ihm zum Stand. Einige Arduino-Boards wechseln den Besitzer. Einer kommt vorbei, sieht die mitgebrachten Exemplare der Vierten Dimension und erkennt: *Das ist mit L^AT_EX gemacht!* Später zeige ich die Vierte Dimension zwei Betreuern des DANTE-Projekts (Deutsche Anwendervereinigung T_EX). Große Freude in ihren Gesichtern, ein Beispiel für L^AT_EX in Aktion zu sehen. Besonders freut der Wechsel von ein- und zweispaltigem Satz. Auch sie sagen: *Wir erkennen L^AT_EX an den Schrifttypen.* (Anmerkung der Redaktion: Wir benutzen zum Satz der Vierten Dimension sogar eine abgewandelte Version der L^AT_EX-Makros, mit denen die DANTE-Zeitschrift *Die T_EXnische Komödie* hergestellt wird — ohne das Konzept Free Software wäre das nicht möglich!)

Beim fünften Gespräch merke ich, dass sich einige meiner Argumente wiederholen:

- Forth ist anfangs leicht zu lernen, man kann sich steigern bis zu hochkomplexen Anwendungen.
- Forth läuft auf Raumsonden (Near), Gepäcksystemen (Dubai), Überwachungssystemen (Atomenergieüberwachung), Prüfständen.
- Unternehmerargumente: Forth passt in einen Mikroprozessor und dokumentiert sich selbst. Das erspart (erleichtert) Versionspflege. Die Hardware vor Ort hat eben ihr eigenes System an Bord. Forth macht meine Entwicklungsabteilung so schnell, dass ich den Mitbewerbern mindestens ein halbes Jahr voraus bin. Patente kann ich mir deshalb sparen.
- Für mich persönlich hat Forth fast Suchtcharakter, es macht einfach Spaß.

Gegen 15 h muss Carsten weg nach Rom. Gegen 16 h fährt Guido zurück nach Leipzig. Schade. Guido wollte mir doch noch etwas über Bayes-Filter zeigen — keine Zeit gehabt! Kurz nach 17 h endet der offizielle Teil der Veranstaltung. Ich packe meine Sachen und das, was Carsten dagelassen hat, und fahre. Fazit: Mit einer Handvoll Menschen über Forth geredet, ein NanoNote kennen gelernt und absolut keine Pause gehabt. Von Langeweile weit und breit keine Spur!

Martin

Hochschule Augsburg



Auf dem Vintage Computer Festival Europe [14] in München traf Carsten auf Prof. Dr.-Ing. Thorsten Schöler von der Hochschule Augsburg [13].

Dieser hatte seine Nase in den Forth-Benchmarking-Contest [15] gesteckt und war so begeistert, dass er beschloss, seine Studierenden mit Forth bekannt zu machen. Und so nahm die Forthbildung ihren Lauf.

Am Sonntagabend reisten Carsten und ich aus getrennten Richtungen nach Augsburg. Neben einem ordentlichen Abendessen (*Wer nichts isst, soll auch nicht arbeiten!*) haben wir die vom BLIT kommenden Arduino-Boards mit neuestem amforth (Version 4.6) [19] bespielt



und getestet, und viel und lange über Forth, die Welt und den ganzen Rest philosophiert.

Am Montagmorgen fuhren wir zur Hochschule, wo wir von Thorsten herzlich empfangen wurden. Alles war vorbildlich organisiert: Raum, Projektor, Strom, Kaffee — keine unerwarteten Klimmzüge in letzter Minute. Lediglich eine Tafel mit Kreide gab es in diesem Raum nicht — also auch keine Kreidestauberinnerungen.

Gegen 10h kamen die Studierenden mit erwartungsvollen Gesichtern und machten sich an den Tischen breit. Neben Thorsten hatten sich 12 Studierende der Fachrichtungen Informatik und Technische Informatik eingefunden. Jeder erhielt ein paar Blätter und ein Exemplar der Vierten Dimension (AVR Sonderheft), welche uns dankenswerterweise vom Vorstand zugesandt worden waren. Dazu verteilten wir die inzwischen wohlbekannten Arduino-*Duemilanove*-Platinen [16] mit dem knallroten Danger-Shield [17] und USB-Kabel. Ein USB-Stick machte die Runde. Jeder erhielt ein Archiv mit amforth-4.6 [19], den verwendeten Demo-Programmen und einigen, nützlichen Progrämmchen und Dateien. Dann konnte es losgehen.

Wir stellten kurz uns selbst vor, dann die Sprache Forth und wo sie herkommt. Als Nächstes zeigte ich mein Lieblingsthema: *Interaktiv* auf dem Kontroller rumfuhrwerken und mit den LEDs blinzeln. Das war der richtige Zeitpunkt, um alle mitgebrachten Notebooks mit den Arduinos zu verbinden. Es wurden serielle Konsolen (minicom, screen) und USB-RS232-Treiber von FTDI [18] installiert, die Information von 8N1, none enträtselt und Reset gedrückt. Nur die Sache mit der unwilligen Korrekturtaste (backspace) in screen wurde nicht gelöst¹. Am Ende bekamen alle den ersehnten ok-Prompt zu sehen.

Als Übung zum Warmwerden und Mittippen führten wir die Benutzung des Stacks vor: *dup*, *drop*, *over*, *rot*, *.s*, *.*, das Wort Stack-Effekt und eine unbekümmerte Heiterkeit machten die Runde. Merke: die Stack-Effekt-Kommentare sehen zwar aus wie eine Argumentliste, sind aber keine!

Die zweite Übung galt den (schon aufgespielten) Worten *portpin:*, *pin_output*, *high* und *low*. Damit lässt es sich prächtig mit den beiden LEDs blinken. Mit Hilfe von *ms* und ein paar neu definierten Worten lässt es sich nicht nur blinken, sondern auch piepsen — der Lärm und die Heiterkeit stiegen um ein paar Dezibel.

Wir hatten eine Aufgabe mitgebracht: Schreibe ein Programm, welches einen Text als Morsecode ausgibt, piepsend oder blinkend. Das Problem kann sehr schön vom kleinen her gelöst und in übersichtliche Funktionen zerlegt werden. Morsezeichen bestehen aus langen und kurzen Zeichen (Strich und Punkt), kurzen und langen Pausen.

```
\ 2 ms T_period ~= 500 Hz
: buzz ( cycles -- )
  0 ?do bz low 1ms bz high 1ms loop
;
: gap ( cycles -- )
  0 ?do bz high 1ms bz high 1ms loop
```

¹ Ctrl-h funktioniert.

;

Dass man eine Pause (*gap*) so umständlich gestaltet, habe ich damit begründet, dass ich damit eben genau gleich lange Zeichen und Pausen generieren kann. Und ganz klar kann man das auch anders lösen. Kurze und lange Zeichen/Pausen werden durch unterschiedliche Parameter erzeugt.

Um ein weiteres Forth-Konzept zu demonstrieren, führten wir vor, dass man den Namen einer Funktion gar nicht braucht, wenn man die Adresse der *Einsprungadresse* XT und das Wort *execute* kennt. Gepaart mit *Edefer* entstanden zwei Worte, die zur Laufzeit zwischen Blinken und Piepsen umschalten können:

```
Edefer transmit
: piepser ['] buzz is transmit ;
: blinker ['] blink is transmit ;
```

Es zeigte sich aber, dass Blinken zu langweilig ist. Piepsen blieb verbreitet. Die Funktionen für kurze und lange Zeichen und Pausen werden nach diesen Vorarbeiten sehr übersichtlich:

```
decimal
: kurz 50 transmit 50 gap ; \ Punkt
: lang 150 transmit 50 gap ; \ Strich
: Zend 100 gap ; \ Pause zwischen Zeichen
: Wend 300 gap ; \ Pause zwischen Worten
```

Solchermaßen ausgerüstet kann man Worte definieren, die einzelne Zeichen ausgeben:

```
: _S kurz kurz kurz Zend ;
: _0 lang lang lang Zend ;
```

Allerdings ist das Definieren aller Zeichen und das Ausgeben eines beliebigen Textes sehr umständlich. Bequemere Lösungen sind erwünscht. Ungefähr zu dieser Zeit sind wir auch in die Mensa essen gegangen. Für mich ein etwas nostalgischer Moment ca. 20 Jahre nach dem letzten Mensabesuch.

Ein weiterer Baustein, um die Aufgabe mit dem Morsen zu lösen, ist das Wort *emit*. Wir argumentieren, dass wir am ok-Prompt gerne so etwas schreiben wollen:

```
> morse s" hier kommt das sos" type endmorse
ok
```

Und dann soll alles richtig funktionieren. Den Rest der Zeit verbrachten wir damit, eine bessere Repräsentation der Daten (einzelner Morsecode für ein Zeichen) und eine schlaue Funktion *morseemit* zu produzieren.

Die eine Lösung besteht aus einer Tabelle, deren Index (0–255) dem ASCII-Wert des zu übertragenden Zeichens entspricht. Der Wert zu diesem Index ist entweder null (nichts zu tun) oder enthält das XT des Wortes (2 Byte), welches dieses eine Zeichen überträgt. Die Werte der Tabelle müssen also vorher ordentlich gelöscht werden, alle Worte, die ein Zeichen ausgeben, müssen ebenfalls generiert werden. Erst definiert man für jedes Morsezeichen ein Wort:

```
: _A kurz lang Zend ;
: _B lang kurz kurz kurz Zend ;
: _C lang kurz lang kurz Zend ;
...
```


Dann erstellt man die Tabelle, löscht ordentlich den Inhalt, und füllt die Tabelle unter Verwendung des Hilfswortes `>mtable`. Das Leerzeichen wird als Wortendepause befüllt.

```
variable mtable 256 2 * allot
```

```
mtable 256 2 * erase
```

```
: >mtable ( xt c -- )
  2 * mtable + !
;
```

\ Tabelle zur Kompilierzeit füllen

```
' _A char a >mtable
' _B char b >mtable
' _C char c >mtable
...
' Wend bl >mtable
```

`morseemit` holt also den Wert aus der Tabelle und ruft damit `execute` auf.

```
variable o-emit \ XT vom normalen "emit"
```

```
: morseemit ( key -- )
  \ altes emit ausführen
  dup o-emit @ execute
  \ Argument auf 0-255 begrenzen
  255 and
  \ XT aus Tabelle lesen
  2 * mtable + @
  \ XT > 0 ?
  dup if
    \ ausführen
    execute
  else
    \ sonst wegwerfen
    drop
  then
;
```

Testen kann man diese Worte, indem man beliebige Zeichen mit `morseemit` ausgibt:

```
> char s morseemit char o morseemit
...
```

Die andere Lösung besteht aus einer ähnlichen Tabelle. Die Bedeutung des Indexes ist gleich, der Wert ist aber eine gepackte Information: 3 Bit enthalten die Anzahl der Morsezeichen (Punkt oder Strich), die übrigen 5 Bit enthalten Null für Punkt und Eins für Strich. Diese Tabelle hat nur ein Byte pro Eintrag und ist damit nur noch halb so groß. `morseemit` holt den Wert aus der Tabelle, entpackt ihn und ruft in einer Schleife die Funktionen `kurz` und `lang` auf, und spendiert eine `Zend`-Pause am Ende. Die Werte der Tabelle müssen hier ebenso aufgebaut und gepackt werden, aber wir sparen alle Worte, die einen einzelnen Morsecode ausgeben.

Die Tabelle wird angelegt und gelöscht. Zwei Worte `pack` und `unpack` zaubern Länge und Code Bits zusammen und wieder auseinander. Danach wird die Tabelle gefüllt.

```
variable mtable 256 allot
mtable 256 erase
```

```
: >mtable ( gepackter-morsecode c -- )
  mtable + c!
```

```
;
```

```
\ Hilfs Worte zum Ent-/Packen von Morse Code
: pack ( #zeichen code -- pcode )
  3 lshift swap 7 and or
;
```

```
: unpack ( pcode -- #zeichen code )
  dup 7 and swap 3 rshift
;
```

```
\ Zahlenbasis auf "2" setzen
: binary ( -- ) 2 base ! ;
```

\ Tabelle zur Kompilierzeit füllen

```
binary 00010 decimal 2 pack char a >mtable
binary 00001 decimal 4 pack char b >mtable
binary 01010 decimal 4 pack char c >mtable
...
```

`domorse` gibt ein einzelnes Morsezeichen aus inclusive der Pause am Zeichenende. `morseemit` holt das gepackte Morsezeichen aus der Tabelle und ruft dann `domorse` auf.

```
variable o-emit \ XT des Original "emit"
```

```
: domorse ( code #zeichen -- )
  \ Schleife Anzahl der Signale
  0 ?do
    dup          \ Kopie anlegen
    1 and        \ erstes Bit maskieren
    if           \ ist Bit gesetzt? dann
      lang       \ Strich
    else         \ sonst
      kurz       \ Punkt
    then         \
    \ verbleibende Bits nach rechts schieben
    2/
  loop
  drop          \ Argument löschen
  Zend         \ Pause Zeichenende
;
```

```
: morseemit ( key -- )
  \ altes emit ausführen
  dup o-emit @ execute
  \ Argument auf 0-255 begrenzen
  255 and
  dup bl = if    \ Leerzeichen? dann
    Wend        \ Pause Wortende
  then

  mtable + c@    \ morse code holen
  unpack         \ entpacken
  domorse        \ Signale ausgeben
;
```

Wenn das alles soweit funktioniert, dann kann man das auch in Aktion erleben. Ein unbedachtes

```
piepser
' morseemit is emit
```

sorgt dafür, dass alle Tastendrucke und Ausgaben des Controllers eifrig mit Piepsen kommentiert werden, - - - . - verkündet den ok-Prompt!

Nachtrag: Diese beiden Lösungen haben einen störenden Makel, denn die mühselig aufgebaute Tabelle mit den Morse-Zeichen verschwindet ins Daten-Nirwana, wenn



der Strom unterbrochen wird. Das ist schade. Um die Tabelle dauerhaft zu speichern, muss sie im Flash-Speicher abgelegt werden. Die Worte , (Komma) und @i (fetch from instruction memory) schreiben bzw. lesen 2 Byte im Flash-Speicher. Eine solche, etwas eingeschränkte Lösung findet der interessierte Leser auf dem Webserver der Forth-Gesellschaft unter [20].

Die Zeit ging viel zu schnell herum. Um etwa 16 h endete die Veranstaltung. Mit einem kleinen Präsent (u. a. ein USB-Stick, der als Flaschenöffner taugt!) wurden wir verabschiedet. Die Teilnehmer zogen diskutierend und mit zufriedenen Grinsen von dannen. Wir sammelten unsere

Sachen wieder ein und beantworteten letzte Fragen. Einer fragte tatsächlich, was er davon hätte, in die Forth-Gesellschaft einzutreten — wir hatten ganz listig Aufnahmeformulare in die Hefte gelegt. Meine Antwort: *Du bekommst das Heft auf Papier, es gibt einmal im Jahr eine Tagung und wenn Du willst, lernst Du Leute wie Carsten und mich kennen.*

Fazit: So eine Forthbildung ist anstrengend und etwas aufwändig. Aber es macht eine Menge Spaß. Die Wissbegierde der Teilnehmer treibt mich ständig voran. Antworten auf Fragen zu geben, bringt mich dazu, Dinge nachzulesen und besser zu verstehen. Deswegen gilt unser Dank den Teilnehmern, ohne die der Ausflug nach Augsburg wahrscheinlich nicht stattgefunden hätte.

Siehe auch

1. blit.org Brandenburger Linux-Infotag
2. <http://www.blit.org/2011/zeitplan/events/151.de.html> Forth Workshop
3. en.wikipedia.org/wiki/TI-92_series
4. sourceforge.net/projects/forth92
5. [VD 2010/04 forth-ev.de/filemgmt/visit.php?lid=337](http://vd.2010/04/forth-ev.de/filemgmt/visit.php?lid=337)
6. sourceforge.net/projects/forth-brainless
7. win32forth.sourceforge.net/
8. [BralUG bralug.de](http://BralUG.bralug.de)
9. de.wikipedia.org/wiki/Conways_Spiel_des_Lebens
10. openrheinruhr.org Ein Pott voll Software
11. <http://programm.openrheinruhr.de/2011/track/OpenHardware/83.de.html> Forth Workshop
12. dante.de Deutsche Anwendervereinigung T_EX
13. hs-augsburg.de Hochschule Augsburg
14. vcfe.org Vintage Computer Festival Europe
15. theultimatebenchmark.org
16. arduino.cc/en/Main/ArduinoBoardDuemilanove
17. www.sparkfun.com/products/10428 Danger Shield
18. ftdi-chip.com/
19. amforth.sourceforge.net
20. www.forth-ev.de/repos/vd/2011-04/Forthbildung/morse4.fs

Listing

```
1 \ --- morse/base.fs ----- 20
2 \ 2011-10-26 EW 21 PORTD 5 portpin: led1
3 \ 2011-11-02 CS 22 PORTD 6 portpin: led2
4 \ arduino duemilanove + danger shield 23
5 \ morse code stuff 24 PORTD 3 portpin: bz
6 \ (Potsdam/Augsburg/Oberhausen) 25
7 26 PORTC 2 portpin: sl1
8 \ make marker loads: 27 PORTC 1 portpin: sl2
9 \ lib/misc.frt 28 PORTC 0 portpin: sl3
10 \ lib/bitnames.frt 29
11 \ lib/ans94/marker.frt 30 PORTC 3 portpin: photocell
12 \ ../devices/atmega328p/atmega328p.frt 31 PORTC 4 portpin: thermometer
13 32 PORTC 5 portpin: knocksensor
14 marker --base-- 33
15 decimal 34 PORTD 4 portpin: sr_in
16 35 PORTD 7 portpin: sr_oe \ output enable
17 PORTB 2 portpin: sw1 36 PORTB 0 portpin: sr_cl
18 PORTB 3 portpin: sw2 37
19 PORTB 4 portpin: sw3 38 \ Piepser
```

```

39 \ 2 ms T_period ~= 500 Hz
40 : buzz ( cycles -- )
41   0 ?do bz low 1ms bz high 1ms loop
42 ;
43 : gap ( cycles -- )
44   0 ?do bz high 1ms bz high 1ms loop
45 ;
46 : blink ( cycles -- )
47   led1 high
48   0 ?do 1ms 1ms 1ms loop
49   led1 low
50 ;
51
52 Edefer transmit
53 : piepser ['] buzz is transmit ;
54 : blinker ['] blink is transmit ;
55
56 decimal
57 : kurz 50 transmit 50 gap ;
58 : lang 150 transmit 50 gap ;
59 : Zend 100 gap ; \ Pause zwischen Zeichen
60 : Wend 300 gap ; \ Pause zwischen Worten
61
62 : init
63   led1 pin_output led1 low
64   led2 pin_output led2 low
65   bz pin_output
66
67   piepser
68 ;

1 \ --- morse/morse2.fs -----
2 \ 2011-10-26 EW
3 \ 2011-11-02 CS
4 \ arduino duemilanove + danger shield
5 \ morse code stuff
6 \ (Potsdam/Augsburg/Oberhausen)
7 \ 2nd version
8
9 marker --morse--
10
11 : _A kurz lang Zend ;
12 : _B lang kurz kurz kurz Zend ;
13 : _C lang kurz lang kurz Zend ;
14 : _D lang kurz kurz Zend ;
15 : _E kurz Zend ;
16 : _F kurz kurz lang kurz Zend ;
17 : _G lang lang kurz Zend ;
18 : _H kurz kurz kurz kurz Zend ;
19 : _I kurz kurz Zend ;
20 : _J kurz lang lang lang Zend ;
21 : _K lang kurz lang Zend ;
22 : _L kurz lang kurz kurz Zend ;
23 : _M lang lang Zend ;
24 : _N lang kurz Zend ;
25 : _O lang lang lang Zend ;
26 : _P kurz lang lang kurz Zend ;
27 : _Q lang lang kurz lang Zend ;
28 : _R kurz lang kurz Zend ;
29 : _S kurz kurz kurz Zend ;
30 : _T lang Zend ;
31 : _U kurz kurz lang Zend ;
32 : _V kurz kurz kurz lang Zend ;
33 : _W kurz lang lang Zend ;
34 : _X lang kurz kurz lang Zend ;
35 : _Y lang kurz lang lang Zend ;

36 : _Z lang lang kurz kurz Zend ;
37
38 \ erstelle Tabelle fuer execution token
39 variable mtable 256 2 * allot
40
41 \ loesche Tabelle
42 mtable 256 2 * erase
43
44 \ Hilfswort zum Fuellen der Tabelle
45 : >mtable ( xt c -- )
46   2 * mtable + !
47 ;
48
49 \ Tabelle zur Kompilier-Zeit fuellen
50 ' _A char a >mtable
51 ' _B char b >mtable
52 ' _C char c >mtable
53 ' _D char d >mtable
54 ' _E char e >mtable
55 ' _F char f >mtable
56 ' _G char g >mtable
57 ' _H char h >mtable
58 ' _I char i >mtable
59 ' _J char j >mtable
60 ' _K char k >mtable
61 ' _L char l >mtable
62 ' _M char m >mtable
63 ' _N char n >mtable
64 ' _O char o >mtable
65 ' _P char p >mtable
66 ' _Q char q >mtable
67 ' _R char r >mtable
68 ' _S char s >mtable
69 ' _T char t >mtable
70 ' _U char u >mtable
71 ' _V char v >mtable
72 ' _W char w >mtable
73 ' _X char x >mtable
74 ' _Y char y >mtable
75 ' _Z char z >mtable
76 ' Wend bl >mtable
77
78 : SOS _S _O _S Wend ;
79
80 variable o-emit
81
82 : morseemit ( key -- )
83   \ altes emit ausfuehren
84   dup o-emit @ execute
85   \ Argument auf 0-255 begrenzen
86   255 and
87   \ XT aus Tabelle lesen
88   2 * mtable + @
89   \ XT > 0 ?
90   dup if
91     \ ausfuehren
92     execute
93   else
94     \ sonst wegwerfen
95     drop
96   then
97 ;
98
99 : morse
100   ['] emit defer@ o-emit !
101   ['] morseemit is emit

```



```

102 ;
103 : endmorse
104   o-emit @ is emit
105 ;

1 \ --- morse/morse3.fs -----
2 \ 2011-10-26 EW
3 \ 2011-11-04 CS
4 \ arduino duemilanove + danger shield
5 \ morse code stuff
6 \ (Potsdam/Augsburg/Oberhausen)
7 \ 3rd version
8
9 marker --morse--
10
11 \ erstelle Tabelle fuer gepackte Morse-Daten
12 variable mtable 256 allot
13
14 \ loesche Tabelle
15 mtable 256 erase
16
17 \ Hilfswort zum Fuellen der Tabelle
18 : >mtable ( gepackter-morsecode c -- )
19   mtable + c!
20 ;
21
22 \ Hilfsworte zum Ent-/Packen von Morse-Codes
23 : pack ( #zeichen code -- pcode )
24   3 lshift swap 7 and or
25 ;
26 : unpack ( pcode -- #zeichen code )
27   dup 7 and swap 3 rshift
28 ;
29
30 \ Zahlenbasis auf "2" setzen
31 : binary 2 base ! ;
32
33 \ Tabelle zur Kompilierzeit fuellen
34 binary 00010 decimal 2 pack char a >mtable
35 binary 00001 decimal 4 pack char b >mtable
36 binary 01010 decimal 4 pack char c >mtable
37 binary 00001 decimal 3 pack char d >mtable
38 binary 00000 decimal 1 pack char e >mtable
39 binary 00100 decimal 4 pack char f >mtable
40 binary 00011 decimal 3 pack char g >mtable
41 binary 00000 decimal 4 pack char h >mtable
42 binary 00000 decimal 2 pack char i >mtable
43 binary 01110 decimal 4 pack char j >mtable
44 binary 00101 decimal 3 pack char k >mtable
45 binary 00010 decimal 4 pack char l >mtable
46 binary 00011 decimal 2 pack char m >mtable
47 binary 00001 decimal 2 pack char n >mtable
48 binary 00111 decimal 3 pack char o >mtable
49 binary 00110 decimal 4 pack char p >mtable
50 binary 01101 decimal 4 pack char q >mtable
51 binary 00010 decimal 3 pack char r >mtable
52 binary 00000 decimal 3 pack char s >mtable
53 binary 00001 decimal 1 pack char t >mtable
54 binary 00001 decimal 3 pack char u >mtable
55 binary 00001 decimal 4 pack char v >mtable
56 binary 00011 decimal 3 pack char w >mtable
57 binary 01001 decimal 4 pack char x >mtable
58 binary 01011 decimal 4 pack char y >mtable
59 binary 00011 decimal 4 pack char z >mtable
60
61 variable o-emit
62
63 : domorse ( code #zeichen -- )
64   \ Schleife Anzahl der Signale
65   0 ?do
66     dup          \ Kopie anlegen
67     1 and        \ erstes Bit maskieren
68     if           \ ist Bit gesetzt? dann
69       lang       \ Strich
70     else         \ sonst
71       kurz       \ Punkt
72     then         \
73     \ verbleibende Bits nach rechts schieben
74     2/
75   loop
76   drop          \ Argument loeschen
77   Zend         \ Pause Zeichenende
78 ;
79 : morseemit ( key -- )
80   \ altes emit ausfuehren
81   dup o-emit @ execute
82   \ Argument auf 0-255 begrenzen
83   255 and
84   dup bl = if   \ Leerzeichen? dann
85     Wend       \ Pause Wortende
86   then
87
88   mtable + c@   \ Morse-Code holen
89   unpack        \ entpacken
90   domorse       \ Signale ausgeben
91 ;
92
93 : morse
94   ['] emit defer@ o-emit !
95   ['] morseemit is emit
96 ;
97 : endmorse
98   o-emit @ is emit
99 ;

```



Vereinsinternes aus der Forthgesellschaft e.V.

M. Anton Ertl, Ewald Rieger

Protokoll der ordentlichen Mitgliederversammlung 2011 der Forth-Gesellschaft e.V. am 17.04.2011 im Zeppelinhaus Goslar

M. Anton Ertl

1. Begrüßung der Mitglieder
Ulrich Hoffmann begrüßt um 9:10h im Namen des Direktoriums die versammelten Mitglieder.
2. Feststellung der Beschlussfähigkeit
24 von 115 Mitgliedern sind anwesend. Damit ist die Mitgliederversammlung laut Satzung beschlussfähig.
3. Wahl des Schriftführers
Zum Schriftführer wurde einstimmig M. Anton Ertl gewählt.
4. Wahl des Versammlungsleiters
Martin Bitter wird einstimmig zum Versammlungsleiter gewählt.
5. Ergänzungen der Tagesordnung
Carsten Strotmann schlägt einige Themen für Verschiedenes vor (siehe unten)
6. Bericht des Direktoriums
 - (a) Bericht der Verwaltung (Ewald Rieger)
 - i. Mitgliederentwicklung
in 2010 gab es 4 Austritte und 2 Eintritte. Damit ergab sich ein Mitgliederstand von 117 zum 31.12.2010. In 2011 sind bis heute bereits 2 Mitglieder ausgetreten, 5 Mitglieder haben ihren Mitgliedsbeitrag 2011 noch nicht bezahlt.
 - ii. Information zum Stand der in 2010 beschlossenen Satzungsänderung
Die Satzungsänderung §16 Auflösungsteil konnte bis zur heutigen Versammlung noch nicht in das Vereinsregister Hamburg eingetragen werden. In mehreren Durchgängen wurden diverse Fehler im Protokoll und in der Satzung beseitigt und jeweils dem Notar und Amtsgericht Hamburg zur Eintragung vorgelegt.
 - iii. Kassenbericht 2010 und Wirtschaftsplan 2011
Ewald Rieger berichtet über das Wirtschaftsjahr 2010 und erläutert die geplanten Ein- und Ausgaben für das Jahr 2011.
Der Kassenbericht wird als separates Dokument dem Protokoll angehängt.
 - i. Kurzbericht des Kassenprüfers
Egmont Woitzel prüfte am Vorabend die Unterlagen und Aufzeichnungen der Kasse und bestätigt, dass alles in Ordnung ist.
 - (b) Rund um das Forth-Magazin
Ulrich Hoffmann beklagt wie auch in den vergangenen Jahren den Mangel an Artikeln für die Vereinszeitschrift VD (der Speicher ist inzwischen vollkommen leer). Daraus entsteht ein hoher Arbeitsaufwand beim Erstellen der VD. Er schlägt vor, die Arbeit zukünftig rollierend auf mehrere Personen zu verteilen. Die nächsten Hefte werden nun von den nachfolgend genannten Mitgliedern erstellt:
Michael Kalus
Carsten Strotmann
Bernd Paysan
Thomas Beierlein
Erich Wälde (Korrekturlesen Formatierung)
Ulrich Hoffmann
- (c) Internet-Präsenz (Ulrich Hoffmann)
Nach einem Festplattenschaden wurde Mitte des Jahres 2010 der Rootserver bei Strato aufgegeben und ein virtueller Server bei Strato angemietet.
- (d) Außendarstellung und Projekte (Bernd Paysan)
der Vortragende lobte die Arbeit der Bildungsgruppe (E. Wälde, C. Strotmann u. M. Bitter). Sie nahmen an verschiedenen Veranstaltungen und Messen teil, um Einführungskurse in Forth (am-Forth) zu halten. Ein ausführlicher Vortrag zu diesem Thema wurde bereits im Rahmen des Tagesprogramms von Erich Wälde gehalten.
7. Entlastung des Direktoriums
Carsten Strotmann stellt den Antrag, das Direktorium zu entlasten.
Das Direktorium wird einstimmig entlastet. (24 Ja, 0 Nein, 0 Enthaltungen)
8. Wahl des Direktoriums
Antrag Michael Kalus: Bestehendes Direktorium wiederwählen.
Der Antrag wird angenommen
Die Direktoren Ulrich Hoffmann, Bernd Paysan und Ewald Rieger wurden einstimmig mit 24 Ja, 0 Nein, 0 Enthaltungen wieder gewählt.
Alle drei nehmen die Wahl an.
Außerhalb der Versammlung: Verleihung des Swap-Drachens an Friedel Amend (Zeremonienmeister: Friedrich Prinz).
9. Projekte
 - Schulung in Factor
ca. 10 Interessierte unter den Teilnehmern der Versammlung
Lehrender muss noch organisiert werden
Abhaltung möglicherweise im Linux-Hotel



- Vintage Wettbewerb
Wer bringt mehr Forths (inkl. Benchmark) auf altem System zum Laufen
Spendiert die FG dafür Preise?
Stoffdrache, T-Shirts, Anstecknadeln, VDs, alte Bücher, Arduino/Dangershield
- Vorschläge für Hingucker
Roboter (Ball, Insekt, Pioneer, Tri-Mühle)
Optik: LED-Würfel (Game of Life)
Augen/Vorhang
- T-Shirts mit Logo etc. (v.a. für Standbesatzung)
Carsten Strotmann meldet sich freiwillig
- PIC Demoboard

Carsten Roederer will ein Demoboard auf Basis des PIC18 entwerfen und herstellen lassen. Geringe Resonanz

- Carsten Strotmann bittet um Vorschläge für moderne Boards für den Microcontroller-Verleih.

10. Verschiedenes

Diskussion über den Tagungsort 2012

Linux-Hotel nicht für 2012

Martin Bitter und Carsten Strotmann organisieren die Tagung, wahrscheinlich in den Niederlanden

EuroForth 2011

Die Versammlung endet um 12:30 h.

Protokollführer

Für das Direktorium

M. Anton Ertl

Ewald Rieger

Einnahmenüberschussrechnung in Euro vom 01.01.2010 - 31.12.2010

Ewald Rieger

Forth-Gesellschaft e.V. Postfach 32 01 24 68273 Mannheim

Ideeller Bereich

Einnahmen aus ideellem Bereich

02110	Echte Mitgliedsbeiträge bis 256 €	2.144,00	
	Summe Beiträge		2.144,00
03220	Erhaltene Spenden/Zuwendungen	183,00	
03221	Geldspenden/-zuwendungen gg. Quittung	1.216,69	
	Summe Spenden		1.399,69

Einnahmen aus ideellem Bereich

3.543,69

Kosten ideeller Bereich

02700	Kosten der Mitgliederverwaltung	-761,26	
02702	Porto, Telefon	-60,00	
02703	Einzugskosten	-3,00	
	Summe Kosten Mitgliederpflege		-824,26
02812	Forth Projekte Unterstützung	-1.588,80	
	Summe Unterstützung Projekte		-1.588,80
02560	Reisekostenerstattungen	-1.105,17	
02810	Repräsentationskosten	-67,29	
	Summe Veranstaltungen		-1.172,46
02802	Geschenke, Jubiläen, Ehrungen	-44,30	
02811	Internetkosten	-511,81	
	Summe Sonstige Kosten ideeller Bereich		-556,11

Summe Kosten ideeller Bereich

-4.141,63

Summe ideeller Bereich

-597,94

Vermögensverwaltung			
Einnahmen der Vermögensverwaltung			
04150	Zinserträge 0 % USt	18,36	
	Summe Einnahmen der Vermögensverwaltung		18,36
	Summe Einnahmen der Vermögensverwaltung		18,36
Kosten der Vermögensverwaltung			
04712	Nebenkosten des Geldverkehrs	-190,84	
	Summe Kosten der Vermögensverwaltung		-190,84
	Summe Kosten der Vermögensverwaltung		-190,84
	Summe Vermögensverwaltung		-172,48
Zweckbetrieb			
Einnahmen aus Zweckbetrieben			
06006	Einnahmen Zeitschrift VD 7% USt.	1.768,77	
	Summe Zeitschrift Vierte Dimension	1.768,77	
01845	Umsatzsteuer 7 %	124,23	
01910	Sammelkonto USt-Vorauszahlung/-Erstattung	189,41	
	Summe Umsatzsteuer		313,64
	Summe Einnahmen aus Zweckbetrieben		2.082,41
Kosten des Zweckbetriebes			
06180	Aufwendungen für bezogene Leistungen	-1.215,40	
06303	Transportkosten	-640,70	
06304	Verpackungskosten Zeitschrift	-39,06	
	Summe Zeitschrift Vierte Dimension		-1.895,16
06340	Verwaltungskosten	-354,00	
	Summe Sonstige Kosten Zweckbetrieb		-354,00
00780	Abziehbare Vorsteuer 19 %	-305,61	
	Summe Vorsteuerabzug		-305,61
	Summe Kosten des Zweckbetriebes		-2.554,77
	Summe Zweckbetrieb		-472,36
	Verlust		1.242,78
Vereinsvermögen Stand 31.12.2010			
00940	Postbank	1.502,13	
00950	Postbank Business Festgeld	14.309,90	
09000	Saldovorträge Sachkonten	-17.054,81	
	Summe Vermögensverlust		1.242,78

Fortsetzung von S. 5

Erste Rückmeldung

Im Zeitalter der digitalen und öffentlichen Produktion, so auch dieses Heftes, kann ein Leserbrief zu ebendiesem Heft schon eintreffen, bevor es gedruckt wurde. Vor nicht allzuvielen Jahren war so etwas noch komplett unvorstellbar.

Dear Fred,

ich danke dir für deine Zeilen. Richte bitte meinen Glückwunsch an alle aus, die zur Fortführung der Vierten Dimension beigetragen haben.

Wie ich schon sagte, habe ich kein Forth-Treffen mehr besucht. Meine Handicaps werden größer und mein Wissen schwindet. Aber ich bin heute frühzeitig aufgestanden und habe mir das Video mit Chuck Moores "Fireside Chat" fast ganz angesehen. Ihr findet es unter <http://www.youtube.com/watch?v=odBjuSCX8jE>

Es ist beruhigend zu wissen, dass es immer noch intelligente, kreative und friedliebende Menschen gibt, die unsere Erde am Laufen halten.

Ich wünsche dir und den Deinen alles Gute zu den Festtagen und für das anstehende Neue Jahr!

Henry

(Übers.: Fred Behringer)



Adventures 10: Technoline Funksensoren belauschen!

Erich Wälde und Martin Bitter

Es kommt der Tag im Leben eines Kontrollettis¹, 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 [3] 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 Artikel [1] beleuchteten wir die Grundlagen. Dieser Artikel zeigt, wie man käufliche Außensensoren für übliche Wetterstationen (Technoline, [4]) erfolgreich belauscht. Ein dritter Teil wird zeigen, wie man seine eigene Funkstrecke aufzieht, beispielsweise um damit den Wasserstand in der dunklen Zisterne zu vermelden.

Neu in diesem Artikel ist die Verwendung der Assemblerworte `c!@spi` und `!@spi`. Das erste wurde unter dem alten Namen `spirw` schon bisher verwendet. Ein zusätzliches Wort, welches eine Zelle und nicht ein Byte überträgt (vorher `><spi`), gab Anlass zur Umbenennung. Die Worte sind im Repository von amforth ab der Version 4.6, erhältlich oder im Anhang.

Technoline TX Außensensoren

Außensensoren von Technoline (z. B. TX3–TH) werden mit einigen Funkwetterstationen verwendet. Die Station steht irgendwo im Haus, die Außensensoren für Temperatur und Luftfeuchte senden regelmäßig ihre Messwerte per Funk. Die Daten werden auf der Station angezeigt, falls der Empfang gut genug war. Die Außensensoren senden im 433-MHz-ISM Band, es sollte also möglich sein, mit einem rfm12-Modul diesen Datenverkehr zu belauschen.



Abbildung 1: Ein Außensensor von Technoline (TX3–TH) für Temperatur und Luftfeuchte

Das Übertragungsprotokoll

Es stellt sich heraus, dass Marko Piering sehr erfolgreich das Übertragungsgeschehen belauscht und entziffert hat [5]. Seine Dokumentation ist sehr lesenswert. Sie beschreibt im Detail den Prozess, das zunächst unbekanntes Protokoll zu verstehen.

Die Sensoren senden auf 433.92 MHz. Diese Frequenz hat Martin in länglichen Versuchsreihen als die günstigste

ermittelt. Funkamateure mit besserer Ausrüstung kommen da wahrscheinlich deutlich schneller zum Ziel. Die Sensoren senden im sogenannten *on-off-keying* (OOK–Betrieb). Das ist wie Morzen: der Sender ist an (Puls) oder aus (Pause). Wie beim Morzen werden kurze (ca. 600 µs) und lange (ca. 1300 µs) Pulse unterschieden. Die Pause zwischen zwei Pulsen bleibt immer gleich und beträgt etwa 900 µs. Ein kurzer Puls wird als logische Eins betrachtet, ein langer Puls als logische Null.

```
|~~~~~|_____|~~~~~|_____|
|<-- eins -->|<-- null -->|
```

Das Funkmodul wird so konfiguriert, dass es die Präsenz des Funksignals direkt auf dem Pin DATA ausgibt. Der Kontroller muss das Signal selbst abtasten und interpretieren.

Das kann auf verschiedene Weise geschehen: (a.) der Kontroller liest den Pegel periodisch (b.) der Kontroller vermisst die Pulslänge mit einem Timer im *Input-Capture*–Verfahren (c.) der Kontroller detektiert die Pulsflanken an einem externen Interrupt-Pin und steuert damit einen Timer. Ich habe mich für die letzte Variante entschieden:

Der DATA-Pin des Funkmoduls wird mit dem INT1-Pin des Kontrollers verbunden. Pegelwechsel lösen einen Interrupt aus. Die Interrupt-Service-Routine startet bei der steigenden Flanke `timer1`. Bei der fallenden Flanke wird der Zählwert von `timer1` ausgelesen, bewertet (war es eine Null oder eine Eins?) und gespeichert.

Die Daten der Sensoren kommen in Paketen zu 44 Bit. Jedes Paket wird nach einer kurzen Pause wiederholt. Misst der Sensor nicht nur die Temperatur, sondern auch die relative Luftfeuchtigkeit, dann wird diese in einem zweiten, ebenfalls wiederholten, Datenpaket gesendet. So eine Sequenz mit zwei oder vier Paketen wird jede Minute ein Mal verschickt.

Die Pulsdauer ist deutlich von den Betriebsbedingungen (Versorgungsspannung, Temperatur) abhängig. Wenn

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


```
MCUCR c@ $04 or MCUCR c!      \ irq on change
GIFR c@ $80 or GIFR c!        \ clear int1
GICR c@ $80 or GICR c!        \ enable int1
;
: -int1
  GIFR c@ $80 or GIFR c!      \ clear int1
  GICR c@ $80 invert and GICR c! \ disable
;
```

Nach dem Aufruf von `init1` aus dem ersten Beispiel braucht es lediglich das Aktivieren des Interrupts: `+int1`, dann sollte die LED wie von Geisterhand flackern, wenn ein Datenblock empfangen wird. Damit wäre bewiesen, dass die Registrierung der Interrupt-Service-Routine und die Aktivierung des Interrupts erfolgreich war.

Signale vermessen

Zum Vermessen der Pulslänge soll `timer1` verwendet werden. Die zugehörigen Register finden sich wie immer im Datenblatt.

Ich benutze vorzugsweise einen Baudraten-Quarz der Frequenz 11.059200 MHz. Wenn ich `timer1` mit einem Vorteiler von 64 aus der Frequenz der CPU speise, dann sind das 172800 Hz, ein `timer1`-Schritt entspricht dann $5.8 \mu\text{s}$. Nach 2^{16} Zyklen oder 379 ms läuft der Zähler über.

Eine Eins mit ca. $600 \mu\text{s}$ sollte einen Zählerstand von etwa 100 erreichen, eine Null ($1300 \mu\text{s}$) erreicht dann etwa 225. Diese Zahlen muss man sich zur Bestätigung ausgeben lassen. Gemessen habe ich Werte von ca. 150 und 295. Die Grenze, um die beiden Bitwerte zu unterscheiden (`bit=1`), habe ich mutwillig auf 200 festgelegt. Zu kurze oder zu lange Pulse werden nicht als fehlerhaft erkannt. Bei anderen Quarzfrequenzen muss man das entsprechend anpassen.

Zunächst definieren wir Worte, die den Zähler starten, anhalten, löschen und den aktuellen Zählwert lesen. Die zugehörige Interrupt-Service-Routine von `timer1` stoppt und löscht den Zähler. Sie wird beim Überlauf des Zählers aufgerufen und sorgt außerdem dafür, dass zu kurze (oder fehlerhaft empfangene) Telegramme verworfen werden (`bit#` löschen). Der Empfänger wird nicht länger blockiert, weil er nicht mehr auf vermeintlich fehlende Bits wartet.

```
decimal
44 constant bits/telegram
variable bit#
\ with timer1 driven by f_cpu/64 (11059200/64=
\ 172800Hz) a pulse indicating a one (~ 600 \mu
\ seconds) results in counts ~ 150. A zero pulse
\ (~1300 \mu seconds) results in counts ~ 300. So
\ counts below 200 are counted as logical 1
200 constant bit=1
variable RXtmp      \ shift collector for bits

: t1_start  $03 TCCR1B c! ;
: t1_stop   $00 TCCR1B c! ;
: t1_clear  $00 TCNT1H c! $00 TCNT1L c! ;
: t1_reset  $04 TIFR c! ;
: t1_get    TCNT1L c@ >> TCNT1H c@ or >> ;

: t1-overflow-isr
```

```
t1_stop
t1_reset
0 bit# !
;
: +timer1
  t1_clear
  0 TCCR1A c!
  t1_reset      \ clear t1ovfl flag
  t1_start      \ normal mode, f/64
                \ register timer1 overflow isr
  ['] t1-overflow-isr TIMER1_OVFAddr int!
                \ enable timer1 overflow int
  TIMSK c@ $04 or TIMSK c!
;
: -timer1      \ disable timer1 overflow int
  TIMSK c@ $04 invert and TIMSK c!
;
```

Die Interrupt-Service-Routine des externen INT1-Interrupts benutzt `timer1`: eine steigende Flanke bewirkt ein löschen und starten des Zählers. Eine fallende Flanke liest den aktuellen Zählerwert aus. Ist der kürzer als 200, dann wird der empfangene Puls als logische 1 gewertet. Das Bit wird in die Variable `RXtmp` geschoben. Die empfangenen Bits werden gezählt (Variable `bit#`). Sind 4 Bit empfangen, wird der Wert in einem Datenblock abgelegt. Wurden 44 Bit empfangen, dann wird über die Variable `fQueue` angezeigt, dass ein vollständiges Telegramm eingegangen ist.

```
44 constant RXsize
variable RXbuf RXsize cells allot \ receive buffer
variable RXtmp      \ nibble shift register

: int1-isr
  _data pin_high? if
    \ leading edge
    t1_clear t1_start

    led_2 low
  else
    \ trailing edge
    \ store bit
    t1_get      \ -- t_diff
    RXtmp @ 1 lshift \ -- t_diff tmp<<1
    swap      \ -- tmp<<1 t_diff
    bit=1 < if
      \ 1
      1+      \ -- tmp<<1 + 1
      \ else
      \ 0 +      \ -- tmp<<1 + 0
    then      \ -- tmp'
    RXtmp !

    \ store nibble
    bit# @ 4 mod 3 = if
      RXtmp @
      RXbuf bit# @ 4 / + c!
      0 RXtmp !
    then

    1 bit# +!
    bit# @ bits/telegram = if
      0 bit# !
      1 fQueue +!
    then
```

```

    led_2 high
  then
    GIFR c@ %10000000 or GIFR c! \ clear int1
;

```

Die Daten werden in Nibble (4 Bit) zusammengefasst und gespeichert. Damit wird zwar die Hälfte des Speichers ungenutzt verschenkt, aber die Deutung der Daten beruht ohnehin auf Nibble, und deren Handhabung wird damit einfacher.

Das Wort `.data` zeigt die empfangenen Daten ohne weitere Interpretation.

```

: .data
  hex
  bits/telegram 4 / 0 do
    RXbuf i + c@ 2 u0.r space
  loop
;

```

Die Funktion `init1` wird um ein paar Zeilen erweitert

```

: init3
  ...
  RXbuf RXsize cells erase
  0 fQueue !
  +timer1 +int1
;

```

In der Hauptschleife des Programms werden die empfangenen Daten ausgegeben:

```

: run3
  ..
  begin
    fQueue @ 0 > if
      .data cr
      0 fQueue !
    then
      key? until
  ...
;

```

Daten!

Hat man das Programm soweit erfolgreich am Laufen (siehe `main-3.fs`), dann wird man mit einer Ausgabe ähnlich dieser belohnt (die hinzugefügten Leerzeilen dienen der besseren Lesbarkeit):

```

00 0A 0E 03 04 06 00 00 06 00 0B
00 0A 0E 03 04 06 00 00 06 00 0B

00 0A 00 03 0B 07 03 09 07 03 05
00 0A 00 03 0B 07 03 09 07 03 05
00 0A 0E 03 0B 05 02 00 05 02 04
00 0A 0E 03 0B 05 02 00 05 02 04

00 0A 00 05 08 07 05 01 07 05 00
00 0A 00 05 08 07 05 01 07 05 00
00 0A 0E 05 09 03 07 00 03 07 0A
00 0A 0E 05 09 03 07 00 03 07 0A

00 0A 00 03 05 07 01 04 07 01 06
00 0A 00 03 05 07 01 04 07 01 06
00 0A 0E 03 04 06 00 00 06 00 0B
00 0A 0E 03 04 06 00 00 06 00 0B

00 0A 00 03 0B 07 03 09 07 03 05
00 0A 00 03 0B 07 03 09 07 03 05

```

Die Deutung der Daten kann man in dem schon erwähnten Dokument von Marko Piering nachlesen (Daten aus der ersten Zeile der Ausgabe von oben, in der Ausgabe ist immer eine führende Null, das ist der verschenkte Speicher):

- **Nibble 0:** (0) das Startnibble hat immer den Wert Null
- **Nibble 1,2:** (AE) diese beiden Nibble bilden das höhere Byte der Sensor-Adresse. A0 ist die Kennzeichnung eines Temperatursensors, AE die Kennzeichnung eines Feuchtesensors.
- **Nibble 3,4:** (34) 7 Bit dieser beiden Nibble bilden das niedrige Byte der Sensor-Adresse (Bits 7...1). Das niedrigste Bit ist ein Paritätsbit.
- **Nibble 5:** (6) Messwert, Zehnerstelle: Bei Temperaturdaten sind von der empfangenen Zehnerstelle 5 abzuziehen. Damit werden negative Temperaturen bis -50°C als Daten ≥ 0 übertragen.
- **Nibble 6:** (0) Messwert, Einerstelle
- **Nibble 7:** (0) Messwert, Zehntelstelle
- **Nibble 8:** (6) wie Nibble 5
- **Nibble 9:** (0) wie Nibble 6
- **Nibble 10:** (B) Checksumme der Nibble 1 bis 9 (Addition ohne Überlauf)

Mit dieser Information kann man den oben gezeigten Block schon sezieren.

- Es finden sich Datenblocks von Temperatur und Luftfeuchte.
- Alle Datenblocks werden wiederholt
- Es finden sich 3 verschiedene Sensoren mit den niedrigen Adressen 3A, 58 und 34
- Die Feuchtigkeiten betragen 52, 37 und 60 %.
- Die Temperaturen betragen 23.9, 25.1 und 21.4 Grad.

Es ist ein Einfaches, die einzelnen Nibble der empfangenen Daten auszugeben, aber natürlich wollen wir den Inhalt nicht selbst deuten. Das nachfolgende Wort nimmt uns das ab, allerdings ist es nicht sehr elegant programmiert.

```

include ewlib/format.fs
11 constant frame.size
: .D ( addr -- )
  hex
  dup >r
  \ --- raw dump of data ---
  frame.size 0 do dup i + c@ 2 u0.r space loop cr
  drop
  \ --- address ---
  r@ 1 + c@ 12 lshift
  r@ 2 + c@ 8 lshift +
  r@ 3 + c@ 4 lshift +
  r@ 4 + c@ +
  $fffe and \ addr
  ." addr:" space 4 u0.r cr
  \ --- value and unit ---
  r@ 5 + c@
  r@ 2 + c@ 0= if 5 - then \ temperature? minus 5
  100 *
  r@ 6 + c@ 10 * +
  r@ 7 + c@ + \ value

```



```
decimal
r@ 2 + c@ 0= if
  ." T C:"
else
  ." rF %:"
then
space 1 +.f cr
\ --- checksum ---
r@ 0
frame.size 1- 0 do over i + c@ + loop
swap drop
$000f and \ crc
hex
dup ." crc:" space 4 u0.r space
r@ frame.size 1- + c@ dup space 4 u0.r space
= if ." ok." else ." invalid" then cr
r> drop
;
```

Wird RXbuf .D anstelle von .data in der Schleife von run3 verwendet, dann ergibt sich beispielsweise folgende Ausgabe:

```
10 0A 00 03 04 07 01 00 07 01 01
addr: A034
T C: +21.0
crc: 0001 0001 ok.
00 0A 00 03 04 07 01 00 07 01 01
addr: A034
T C: +21.0
crc: 0001 0001 ok.
00 0A 0E 03 04 07 04 00 07 04 05
addr: AE34
rF %: +74.0
crc: 0005 0005 ok.
00 0A 0E 03 04 07 04 00 07 04 05
addr: AE34
rF %: +74.0
crc: 0005 0005 ok.
```

Datenverwaltung

Der Rest ist viel Fleißarbeit. Es empfiehlt sich, den vollständig empfangenen Block schleunigst umzukopieren in eine Art Ringpuffer. Dann können immer 4 Telegramme gesammelt und ausgewertet werden.

Auf dem Controller kann man die Daten eine Weile sammeln und auf ein Speichermedium schreiben. Man kann die Daten auch regelmäßig einsammeln, wenn ein Rechner ständig in Betrieb ist. Man kann die Daten auch nur auf einem LCDDisplay anzeigen. All das hängt vom individuellen Ziel und der verfügbaren Zeit ab. So ein Projekt ist erfahrungsgemäß nicht mal geschwind an einem Wochenende erledigt.

Referenzen

1. E. Wälde und M. Bitter, Funklöcher!, Die 4. Dimension 3/2011, Jahrgang 27
2. <http://amforth.sourceforge.net/>
3. <http://www.pollin.de>
4. technoline TX3–TH o.ä.
5. http://ccintern.dharclos.de/TX2TX4_433MHz_Wetterstation_1_0.zip
6. <http://www.hoperf.com>

²praktischerweise auch als Artikel in der VD!

Wenn die Daten dann ordentlich eingesammelt sind, wollen sie auf dem Rechner genauso ordentlich abgespeichert werden (Dateien oder Datenbank) und als Kurven angezeigt und bestaunt werden.

Versionsverwaltung des kompletten Projekts und ausreichende Dokumentation² sind auch für Gelegenheitsprogrammierer kein Luxus. Nota bene: der arme Tropf, der in zwei Jahren den wildgewordenen Programm-Code entziffern muss, weil irgendetwas nicht so tut wie gewünscht, der bist Du selbst.

So Sachen

Beim ersten Sezieren der Daten war mir noch entgangen, dass bei Temperaturen die Zehnerstelle um 5 erhöht übertragen wird. Und da es gerade einigermaßen passte, hegte ich den Verdacht, dass die Temperaturen in Fahrenheit übertragen werden — sowas wundert mich ja nicht. Allerdings wollten die Daten auf der Anzeige und die von mir umgerechneten Daten einfach nicht zusammenpassen. Martin hat mich netterweise von meinem Holzweg geholt. Die Deutung der übertragenen Daten steht sehr ausführlich in der Dokumentation von Marko Piering.

An dieser Stelle sei auch darauf hingewiesen, dass die Sensoren zwar alle 58 Sekunden Daten verschicken, dass die gekaufte Empfangsstation aber lediglich zwei Mal in 10 Minuten überhaupt Daten empfängt. Das kann unsere Selbstbaustation jetzt schon besser. Außerdem berichtet Martin, dass die Selbstbaustation immer noch wacker Daten empfängt und auswertet, wenn die gekaufte Empfangsstation nur noch drei langweilige Striche anzeigt. Deren Empfänger ist wohl nicht so gut wie das rfm12-Funkmodul.

Die Fernbedienung der neuen Rollos sendet ebenfalls deutlich sichtbar im 434-MHz-Band. Und ab und zu findet sich auch eine Station von Nachbarn in den Daten.

Werden in einen Außensensor neue Batterien eingelegt, dann ändert sich dessen Adresse, was ziemlich unpraktisch ist. Als Erstes wird ein Telegramm geschickt, welches spezielle Informationen zum Thema enthält, aber damit habe ich mich bislang nicht beschäftigt.

Spezieller Dank geht an Martin Bitter, der diesen Betriebsmodus mit bemerkenswerter Ausdauer erkundet hat, an Marko Piering, der das Protokoll entziffert und seine Erkenntnisse in sehr ordentlicher Form veröffentlicht hat, an Matthias Trute für die ständige Verbesserung von amforth und an die Teilnehmer des mittwöchlichen IRC-Treffens, für allerhand Rat und Tat.

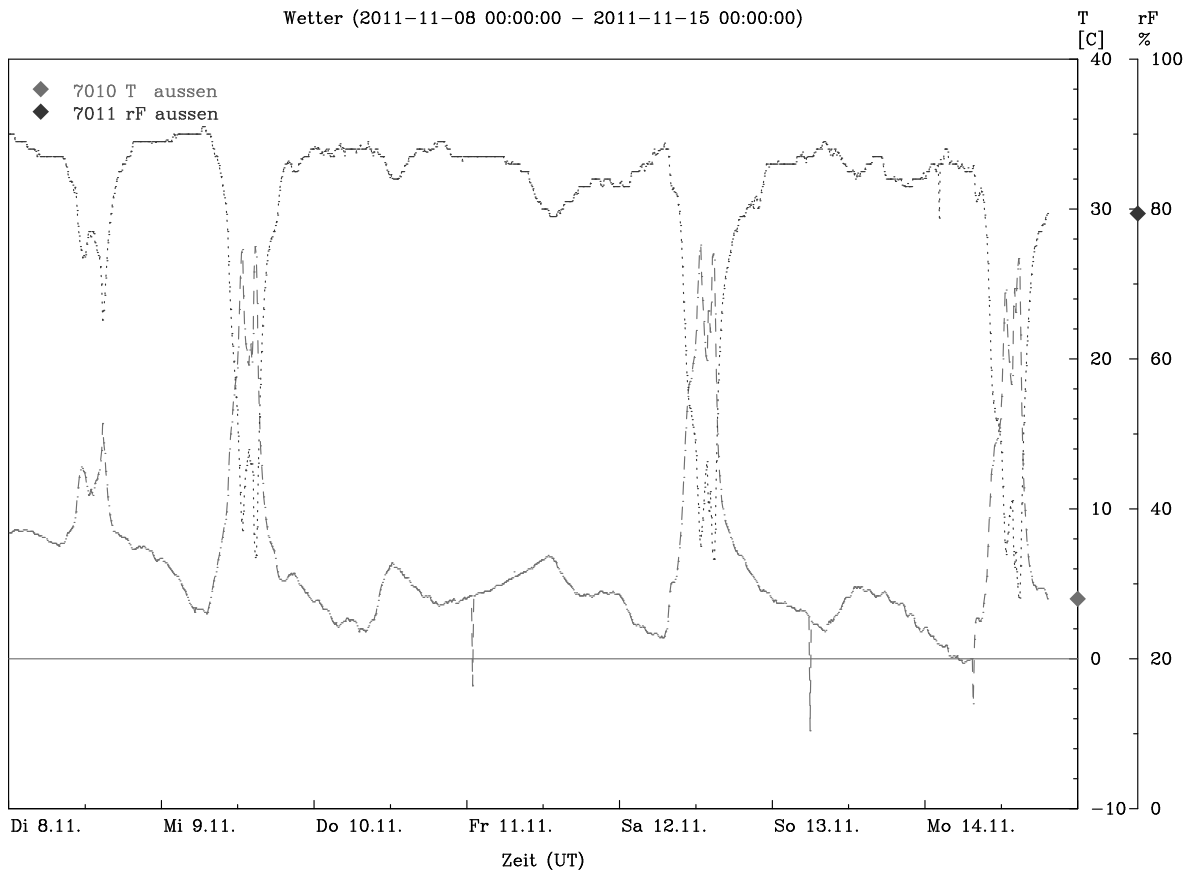


Abbildung 2: Mit dem vorgestellten Programm aufgenommene Daten für Lufttemperatur (rot, Striche) und relative Luftfeuchtigkeit (blau, Punkte). In den Temperaturdaten finden sich Ausreißer, deren Ursache im Moment unklar ist. Die Doppelspitze mittags kommt durch Beschattung des Sensors zustande. Der Sensor befindet sich auf der Südseite, die Mittagstemperaturen sind deutlich zu hoch.

Listings

Bei den Listings ist zu beachten, dass ich mein Funkmodul über den Pin PortA.7 mit Strom versorge. Dementsprechend gibt es die Worte `_w_pwr`, `rfm12.power.on` und `rfm12.power.off`. Diese Worte können natürlich weggelassen werden, wenn das Funkmodul immer mit Strom versorgt wird.

```

1  \ main-1.fs
2  \ report rfm12 activity on LEDs and
3  \ serial console
4  \ rfm12 power is on PORTA.7!
5
6  include part1.fs
7
8  : .data
9  _data pin_high? if
10     led_2 low  led_3 high  [char] |
11  else
12     led_2 high led_3 low   [char] .
13  then
14  emit
15  ;
16  : init1
17     led_0 high led_0 pin_output
18     led_1 high led_1 pin_output
19     led_2 high led_2 pin_output
20     led_3 high led_3 pin_output
21     -jtag
22     rfm12.power.on &1000 ms
23     +spi +rfm12 rfm12.init-ook
24     _data pin_input
25     ;
26     : run1
27         init1
28         w.status cr
29         begin
30             .data
31             key? until
32             key drop
33             rfm12.power.off
34             ;
35
36     1 ; words/spirw.asm
37     2 ; ( txbyte -- rxbyte)
38     3 ; MCU
39     4 ; SPI exchange of 1 byte
40     5 VE_SPIRW:
41         6 .dw $ff06
42         7 .db "c!@spi"
43         8 .dw VE_HEAD
44         9 .set VE_HEAD = VE_SPIRW
45     10 XT_SPIRW:
46         11 .dw PFA_SPIRW

```



```

12 PFA_SPIRW:
13     rcall do_spirw
14     clr tosh
15     jmp_ DO_NEXT
16
17 do_spirw:
18     out_ SPDR, tosl
19 do_spirw1:
20     in_ temp0, SPSR
21     sbrs temp0, 7
22     rjmp do_spirw1 ; wait until complete
23     in_ tosl, SPDR
24     ret
1 ; words/2spirw.asm
2 ; ( n1 -- n2 )
3 ; MCU
4 ; SPI exchange of 2 bytes, high byte first
5 VE_2SPIRW:
6     .dw $ff05
7     .db "!@spi",0
8     .dw VE_HEAD
9     .set VE_HEAD = VE_2SPIRW
10 XT_2SPIRW:
11     .dw PFA_2SPIRW
12 PFA_2SPIRW:
13     push tosl
14     mov tosl, tosh
15     call_ do_spirw
16     mov tosh, tosl
17     pop tosl
18     call_ do_spirw
19     jmp_ DO_NEXT
20
1 \ 2011-08-28 EW ewlib/spi.fs
2 \ spi, using hw interface
3 \ in dict_appl.inc:
4 \     .include "words/spirw.asm"
5 \     .include "words/2spirw.asm"
6 \ words:
7 \     +spi ( -- )
8 \     -spi ( -- )
9 \ transfer 1 byte: c!@spi ( c -- c' )
10 \ transfer 1 cell: !@spi ( n1 -- n2 )
11
12 \ needs these defined before loading:
13 PORTB 4 portpin: /ss
14 \ PORTB 5 portpin: _mosi
15 \ PORTB 6 portpin: _miso
16 \ PORTB 7 portpin: _clk
17
18 : +spi ( -- )
19 /ss high \ activate pullup!
20 _mosi high _mosi pin_output
21 _clk low _clk pin_output
22 \ not needed, see datasheet:
23 \ _miso pin_pullup_on
24
25 \ enable, master mode
26 \ f_cpu/128 speed
27 $53 SPCR c!
28 ;
29 : -spi 0 SPCR c! ;
1 \ 2011-05-29 EW ewlib/rfm12.fs
2 \ wireless used as synonym for rfm12
3 \

```

```

4 \ words:
5 \     +rfm12 rfm12.init
6 \     >wc wc? w? >w
7 \     w.status
8
9 \ wait for wireless data tx ready
10 : w? ( -- )
11     _rfm12 low
12     begin
13         _miso pin_high?
14     until
15 ;
16 \ write 1 cell to wireless control
17 : (>wc) ( x -- x' )
18     _rfm12 low
19     !@spi \ 2spirw
20     _rfm12 high
21 ;
22 \ write wireless control, drop reply
23 : >wc ( x -- )
24     (>wc) drop
25 ;
26 \ read wireless status
27 : wc? ( -- x )
28     0 (>wc)
29 ;
30 : w.status
31     wc? 4 hex u0.r
32 ;
33 \ write wireless data
34 : >w ( c -- )
35     $00ff and $b800 +
36     w?
37     (>wc) drop
38 ;
39 \ read one byte
40 : <w ( -- c )
41     $b000
42     w?
43     (>wc)
44 ;
45 : +rfm12
46     _rfm12 low
47     _rfm12 pin_output
48 ;
49 : rfm12.init
50 \ init sequence
51 $80d7 >wc
52 $82d9 >wc \
53 $a530 >wc \ 433.620 MHz
54 $c647 >wc \ 4800 baud
55 $94a0 >wc \ bandwidth: 134 kHz
56 $c2ac >wc
57 $ca81 >wc
58 $c483 >wc \ AFC
59 $9854 >wc \ (5+1)*15 = 90 kHz Hub, -12dB
60 $e000 >wc
61 $c800 >wc
62 $c000 >wc
63 ;
1 \ main-2.fs
2 \ report rfm12 activity on LEDs via
3 \ interrupt service routine on INT1
4 \ rfm12 power is on PORTA.7!
5

```

```

6  include part1.fs
7
8  : int1-isr
9    _data pin_high? if
10     \ leading edge
11     led_2 low
12   else
13     \ trailing edge
14     led_2 high
15   then
16     \ clear int1
17     GIFR c@ %10000000 or GIFR c!
18   ;
19   : +int1
20     ['] int1-isr INT1Addr int! \ register isr
21     MCUCR c@ $04 or MCUCR c! \ irq on change
22     GIFR c@ $80 or GIFR c! \ clear int1
23     GICR c@ $80 or GICR c! \ enable int1
24   ;
25   : -int1
26     \ clear int1 and disable
27     GIFR c@ %10000000 or GIFR c!
28     GICR c@ %10000000 invert and GICR c!
29   ;
30
31   : init1
32     led_0 high led_0 pin_output
33     led_1 high led_1 pin_output
34     led_2 high led_2 pin_output
35     led_3 high led_3 pin_output
36   -jtag
37   rfm12.power.on &1000 ms
38   +spi +rfm12 rfm12.init-ook
39   _data pin_input
40   ;
41
42   \ main-3.fs
43   \ receive rfm12 00K telegrams (44 Bits)
44   \ use int0-isr to receive the bits,
45   \ use timer1-isr to measure pulse lengths
46   \ rfm12 power is on PORTA.7!
47
48   include part1.fs
49
50   decimal
51   \ --- TX2,3,4 sensor reading -----
52   \ data telegrams are send and received
53   \ in chunks of 44 Bits
54   &44 constant bits/telegram
55   variable bit#
56   \ with timer1 driven by f_cpu/64
57   \ (11059200/64=172800Hz) a pulse
58   \ indicating a one (~ 600 \mu seconds)
59   \ results in counts ~ 100. A zero pulse
60   \ (~1300 \mu seconds) results in counts
61   \ ~ 225.
62   \ Measurements yield ~ 150 and ~ 300
63   \ counts. So counts below 200 are taken
64   \ as logical 1
65   &200 constant bit=1
66
67   \ indicate a full frame has been
68   \ received, triggers processing of
69   \ the received frame
70   variable fQueue
71
72   include rfm12_timer1.fs
73
74   &44 constant RXsize
75   \ receive buffer, really too large
76   variable RXbuf RXsize cells allot
77   \ nibble shift register
78   variable RXtmp
79
80   include rfm12_int1.fs
81
82   : .data
83     hex
84     bits/telegram 4 / 0 do
85       RXbuf i + c@ 2 u0.r space
86     loop
87   ;
88
89   include ewlib/format.fs
90   11 constant frame.size
91   : .D ( addr -- )
92     hex
93     dup >r
94     \ --- raw dump of data ---
95     frame.size 0 do
96       dup i + c@ 2 u0.r space
97     loop cr drop
98     \ --- address ---
99     r@ 1 + c@ 12 lshift
100    r@ 2 + c@ 8 lshift +
101    r@ 3 + c@ 4 lshift +
102    r@ 4 + c@ +
103    $fffe and \ addr
104    ." addr:" space 4 u0.r cr
105    \ --- value and unit ---
106    r@ 5 + c@
107    \ temperature?
108    r@ 2 + c@ 0= if 5 - then
109      100 *
110    r@ 6 + c@ 10 * +
111    r@ 7 + c@ + \ value
112    decimal
113    r@ 2 + c@ 0= if
114      ." T C:"
115    else
116      ." rF %:"
117    then
118    space 1 +.f cr
119    \ --- checksum ---
120    r@ 0
121    frame.size 1- 0 do
122      over i + c@ +
123    loop swap drop
124    $000f and \ crc
125    hex
126    dup ." crc:" space 4 u0.r space
127    r@ frame.size 1- + c@
128    dup space 4 u0.r space
129    = if ." ok." else ." invalid" then cr
130    r> drop
131  ;
132
133  : init3
134    led_0 high led_0 pin_output
135    led_1 high led_1 pin_output
136    led_2 high led_2 pin_output
137    led_3 high led_3 pin_output

```



```

97  -jtag
98  rfm12.power.on &1000 ms
99  +spi +rfm12 rfm12.init-ook
100 _data pin_input
101
102 RXbuf RXsize cells erase
103 0 fQueue !
104 +timer1 +int1
105 ;
106 : run3
107   init3
108   w.status cr
109   begin
110     fQueue @ 0 > if
111       .data cr
112       RXbuf .D
113       0 fQueue !
114       then
115       key? until
116       key drop
117       rfm12.power.off
118 ;

1  \ rfm12_timer1.fs
2  : t1_start  $03 TCCR1B c! ;
3  : t1_stop   $00 TCCR1B c! ;
4  : t1_clear  $00 TCNT1H c! ;
5             $00 TCNT1L c! ;
6  : t1_reset  $04 TIFR  c! ;
7  : t1_get    TCNT1L c@ ><
8             TCNT1H c@ or >< ;
9
10 : t1-overflow-isr
11   t1_stop
12   t1_reset
13   0 bit# !
14 ;
15 : +timer1
16   t1_clear
17   0 TCCR1A c!
18   t1_reset          \ clear t1ovfl flag
19   t1_start          \ normal mode, f/64
20                     \ register timer1 overflow isr
21   ['] t1-overflow-isr
22   TIMER1_OVFAddr int!
23   \ enable timer1 overflow int
24   TIMSK c@ $04 or TIMSK c!
25 ;
26 : -timer1 \ disable timer1 overflow int
27   TIMSK c@ $04 invert and TIMSK c!
28 ;

1  \ rfm12_int1.fs
2  : int1-isr
3    _data pin_high? if
4      \ leading edge
5      t1_clear t1_start
6
7      led_2 low
8    else
9      \ trailing edge
10     \ store bit
11     t1_get          \ -- t_diff
12     RXtmp @ 1 lshift \ -- t_diff tmp<<1
13     swap            \ -- tmp<<1 t_diff
14     bit=1 < if
15       \ 1
16       1+          \ -- tmp<<1 + 1
17       \ else
18       \ 0 +          \ -- tmp<<1 + 0
19       then          \ -- tmp'
20       RXtmp !
21
22       \ store nibble
23       bit# @ 4 mod 3 = if
24         RXtmp @
25         RXbuf bit# @ 4 / + c!
26         0 RXtmp !
27       then
28
29       \ telegram complete?
30       1 bit# +!
31       bit# @ bits/telegram = if
32         0 bit# !
33         1 fQueue +!
34       then
35
36       led_2 high
37     then
38     \ clear int1
39     GIFR c@ $80 or GIFR c!
40 ;
41 : +int1
42   \ register int1-isr
43   ['] int1-isr INT1Addr int!
44   \ irq on level change
45   MCUCR c@ $04 or MCUCR c!
46   \ clear int1
47   GIFR c@ $80 or GIFR c!
48   \ enable int1
49   GICR c@ $80 or GICR c!
50 ;
51 : -int1
52   \ clear int1 and disable
53   GIFR c@ $80 or GIFR c!
54   GICR c@ $80 invert and GICR c!
55 ;
56
1  \ 2008-10-21 EW  format.fs
2
3  \ _always_ display sign '+' or '-'
4  : sign! 0 < if
5    [ char - ] literal
6  else
7    [ char + ] literal
8  then hold
9  ;
10
11 \ +.f, d+.f
12 \  always display sign [+,-],
13 \  scaled to n digits after '.'
14 \  no leading/trailing zeros/spaces
15 : d+.f ( d n -- )
16   over >r >r dabs
17   <# r> 0 ?do # loop
18   [ char . ] literal hold
19   #s r> sign! #>
20   type
21 ;
22 : +.f ( x n -- )
23   swap s>d rot d+.f
24 ;
25

```


EuroForth 2011 in Wien

Bernd Paysan

Die 27. EuroForth-Konferenz fand dieses Jahr vom 23. bis 25. September bei schönem Wetter in Wien statt — turnusgemäß, möchte man sagen, da sich Anton als zuverlässiger Veranstalter des dritten Orts („weder Deutschland noch England“) gefunden hat. Wie auch in den Jahren zuvor waren vor der Tagung zwei Tage Forth200x-Standardisierung angesetzt, wie immer, wenn es an diesem dritten Ort stattfindet, ist die Runde etwas kleiner.

Forth 200x: Der Snapshot

Der Standardisierungsprozess ist so weit fortgeschritten, dass wir einen Snapshot produzieren, der dann — gemäß den ANSI-Prozeduren — einem öffentlichen Review zugeführt wird. Als Snapshot wird das Ergebnis des Treffens beschlossen. Damit war auch klar, womit wir uns hauptsächlich beschäftigen würden, mit „crossing the eyes and dotting the teeth“, wie die verballhornte Version lautet. Da gibt es immer Details, die nicht sauber beschrieben sind, und entsprechend umformuliert werden müssen.

Zunächst wurden also die bereits vorgenommenen Änderungen begutachtet, und für gut befunden. Danach die schon im Raum stehenden, also durch einen CfV beschlossenen Vorschläge, also die enhanced locals, BUFFER: und SUBSTITUTE. Diese Vorschläge wurden ebenfalls akzeptiert.



Abbildung 1: Der Tagungsraum

Von den noch in Diskussion befindlichen (RfD) wurden die Änderungen am IS-Vorschlag akzeptiert, ebenso wie die Erläuterungen für Parsing Words in Blöcken von Multitasking-Systemen und die Einschränkung, dass der bei SOURCE-ID zurückgelieferte Dateideskriptor nicht anderweitig benutzt werden kann. Die Diskussion über FOUND ist noch nicht so weit ausgegoren, dass man daraus einen Standard machen kann.

Danach ging es um noch durchzuführende Änderungen, um aus dem Dokument einen richtigen Snapshot zu machen. Dabei werden einige Anhänge gestrichen, bzw. wandern auf die Web-Site des forth200x-Committees.

Der Standardisierungsprozess ist aber dadurch natürlich nicht abgeschlossen, sondern geht weiter. Es gibt noch

offene Baustellen, insbesondere die Internationalisierung (die jetzt, mit xchars und substitute im Standard, alle notwendigen Komponenten zur Verfügung hat), aber auch die Diskussion zum Cross-Compiler, zu IEEE Floating Point, zum Laden von Dateien relativ zur gerade geladenen Datei (was Stephen Pelc inzwischen auch unterstützt), und das Memory Access Wordset.

Konferenz

Stephen Pelc: Standardize OOP Now

Stephen Pelc ist noch nicht ganz aus dem Standardisierungs-Modus 'raus, und er macht gleich ein richtig dickes Fass auf. Sein Vortrag befasst sich mit der Situation von OOP in Forth, und warum die derzeitige Situation nicht gut sei: praktisch jedes Programm, das tatsächlich OOP verwendet, bringt sein eigenes, zu allen anderen Forth-OOP-Paketten inkompatibles OOP mit. Stephen fordert die Entwickler bestehender OOP-Erweiterungen auf, einzusehen, dass ihre bisherigen Vorschläge alle unzulänglich sein müssen, weil sich deshalb ja keines durchgesetzt hat, und also zurück auf Los zu gehen, und nochmal neu anzufangen — und dabei die Egos zurückzustellen. Er schlägt vor, auf FMS (Forth meets Smalltalk) von Doug Hoffman aufzubauen, gibt aber auch zu, dass er selbst kein OOP-Experte ist. Das Thema bedarf näherer Betrachtung; ich habe deshalb einen ganzen Artikel für die VD geschrieben.

Bernd Paysan: net2o: Application Layer

Ich habe mir noch mehr Gedanken über die Neuerfindung des Internets gemacht, und trage meine Ideen vor. Wofür es ist — um Informationen zu tauschen. Wie man die Informationen organisiert. Was die Elemente einer Webseite sind. Dass der Browser eine Laufzeitumgebung für Anwendungen im Netz ist. Dass man nicht nur Code faktorisieren kann und muss, sondern auch Daten. Wie man die ACID¹-Bedingungen in einem verteilten Datenspeicher erreicht. Welche Rolle Hashes als Index auf die Daten spielen. Wie man Anwendungen aus dem Netz sicher ausführt (und dass Sicherheit immer relativ ist). Dass man nicht pollen, sondern pushen soll. Und Sourcecode verwenden, keine Binaries. Auf welche Library-Funktionen man aufsetzt. Und was die dümmsten Fehler am aktuellen Internet sind.

¹ Atomicity, Consistency, Isolation, Durability



Leon Wagner: Forth on the Cortex-M1 FPGA Development Kit

Leon Wagner hat tatsächlich Hardware dabei, ein Altera-Development-Board, und führt vor, wie man in Quartus einen Cortex-M1 mit Peripherie zusammenklickt, auf dem dann SwiftX läuft. Dabei muss man tatsächlich nicht in die Tiefen des Verilogs eintauchen, wobei man das bei ARM auch nicht kann — das ist alles verschlüsselt. SwiftX darauf zu portieren ist etwa genauso schwierig oder einfach wie auf eine andere CPU, schließlich hat der Cortex-M1 einfach einen abgespeckten Thumb2-Befehlssatz.

Anton Ertl: Ways to Reduce the Stack Depth

Anton macht sich Gedanken, wie man mit zu vielen Variablen auf dem Stack umgeht. Mehr als 2 oder 3, und es wird ungemütlich. Eine Reihe von Vorgehensweisen sind bekannt (das sind auch „Design Pattern“, aber nicht für OOP, sondern für Forth), und die erläutert er an Beispielen. Man kann die Daten im Speicher gruppieren, mehrere Stacks verwenden, Locals benutzen, globale Variablen oder Kontext-Variablen, implizite Parameter, Pipes und Staged Execution.

Andrew Haley: Forth concurrency: what are we going to do about „volatile“?

Wie schon letztes Jahr ist Andrew Haleys Thema Forth im Kontext von Multiprozessor-Systemen. Es geht dieses Mal um Schreib- und Lesebarrieren im Speicher, die in C-artigen Sprachen durch den Typbezeichner „volatile“ abgebildet werden — oder auch nicht, denn so klar sind sich die Implementierer von C-artigen Sprachen nicht, was denn jetzt wirklich mit volatile gemeint ist (insbesondere C selbst hat eine praktisch nutzlose Definition von volatile). Andrew ist über das Problem gestolpert, als VFX-Forth aus `counter @ dup 2+` einfach `counter @ 2+ counter @ swap` gemacht hat, und Stephen auf den Bugreport mit „das ist das volatile-Problem“ geantwortet hat. Damit hat das Problem seinen Namen weg. Bei den C/C++-Leuten weicht die bisherige Ignoranz langsamer Erleuchtung, und wenn Forth eine Zukunft auf Multiprozessorsystemen haben will, dann auch nur, wenn man sich dazu Gedanken macht.



Abbildung 2: Impressionen

Willi Stricker: A processor as hardware version of the Forth virtual machine

Willi Stricker stellt seinen Forth-Prozessor im FPGA vor; wir kennen den Prozessor ja schon von der Forth-Tagung. Sein Konzept ist sehr nahe an der virtuellen Forth-Maschine für threaded code, weshalb bei ihm Primitives und Unterprogrammaufrufe auch einfach nur Adressen im Speicher sind, und für die Primitives reserviert er die ersten 64 Bytes — das sind dann Pseudo-Adressen, denn an denen steht nicht etwa der Code für den Befehl, sondern das FPGA weiß, was zu tun ist, und führt die Primitives gleich direkt aus. Willi hat das auf einem Actel-FPGA implementiert.

Nick Nelson: Crash Never

Als Kontra gegen Stephen Pelcs „crash early, crash often“-Philosophie schlägt Nick Nelson eine „crash never“-Philosophie vor, zumindest für ausgelieferte Programme. Kunden sollen auf Crashes allergisch reagieren, und auch Nick selbst hat auf der EuroForth in Exeter von ein paar Crashes erzählt, die für die Zuhörer sicher lustiger waren als für die Beteiligten. Also listet Nick die Techniken auf, die er so benutzt, um seine Programme robust zu machen, auch wenn sie nicht perfekt sind, und hier und da natürlich trotzdem crashen — oder crashen würden, wenn man sie nicht von außen stützen täte. So fängt er Stackfehler ab, indem die Hauptschleife eines Tasks den Stack immer wieder normalisiert. Wenn es trotzdem zum Crash kommt, wird eine möglichst detaillierte Meldung produziert. Schleifen mit zu großer Anzahl Schritten werden zurückgewiesen oder verkürzt. Watchdogs („prodger“) stoppen Programme, die nicht mehr reagieren.

Divisionen durch 0 sollen Unendlich zurückgeben (mit korrektem Vorzeichen, falls man nach einem Vorzeichen gefragt hat), und die Anzahl Threads minimiert man — wenn mehrere Prozesse alle 100 ms etwas tun wollen, dann ruft man sie hintereinander aus dem gleichen Thread auf. Und für die Kommunikation verzichtet man auf eigene Locks, sondern verwendet möglichst simple Methoden.

Gerald Wodni: SWIG & The Forth Net: Hands-On

Gerald Wodni hat immer noch das „the“ im `theforth.net`; weil die andere Domain leider belegt ist, funktioniert der `drop`-Operator nicht. Aber das System hat Fortschritte gemacht, und für Anton gibt es auch einen Stil ohne bunte Farben, und man kann sich ohne OpenID-Account direkt einloggen. Es wartet nun darauf, mit Libraries gefüttert zu werden. Forth-Systeme können diese Libraries direkt über's Internet ziehen, man muss sie nicht umständlich installieren oder so, sie sind einfach da. Das Forth-Net schickt auf die HTTP-Anfrage Forth-Code zurück, der den ganzen Rest erledigt. Die Wörter dafür heißen dann einfach `frequire` und `finclude`, und verwenden `fget`, das auf das Forth-Netz zugreift.

Libraries muss man auch nicht selber schreiben, man kann die vorhandenen Libraries (üblicherweise in C geschrieben) verwenden. Damit man da keinen großen Aufwand treiben muss, das in Forth einzubinden, gibt's ein SWIG-Modul, das aus den C-Headern über einen zweistufigen Prozess Forth-Bindings generiert.

Am Sonntag finden die Impromptu-Talks und Workshops statt. Dieses Mal mit sehr vielen Themen.

Carsten Strotmann

Carsten berichtet über seine Aktivitäten, Forth einer breiteren Öffentlichkeit nahezubringen. Ich denke, die interessanteste Information dabei ist, dass die früheren Vorbehalte gegen Forth inzwischen in den Hintergrund drängen; es scheint, dass die Welt jetzt offener für Konzepte abseits des Mainstreams ist.

Stephen Pelc: Why compilation is a mess

Stephen Pelc hat das Konzept des „intelligenten compile,s“ inzwischen auch implementiert, und stellt sein Konzept eines „universellen Tokens“ vor, das als Ersatz für das bisherige Durcheinander aus `immediate`, `state-smart`-Wörtern und langen `case`-Statements im weniger intelligenten `compile`, herhalten muss. Dieses Token ist bei Stephen das ganz normale `xt`, bei Dual-`xt`-Systemen wie Gforth klappt das so nicht. Aber Gforth ist auch noch einen Schritt weiter hinten bei der Implementierung des intelligenten `compile,s` (der Cross-Compiler kann's, aber das System selbst nicht).

Anton Ertl: Replacing FIND

Die Überlegungen, `FIND` durch ein zeitgemäßeres Wort zu ersetzen, gehen auch in die Richtung, im Compilations-Prozess aufzuräumen. Gforth mit seinem `nt` (name token) hat da etwas andere Operatoren als Stephens universelles token, was sich auch auf einen möglichen Ersatz von `find` auswirkt. Die Diskussion hier, so scheint mir, ist definitiv noch nicht so weit ausgereift, dass man von einem Ergebnis sprechen kann.



Abbildung 3: Ertlgasse und Antonshöhe

Klaus Schleisiek: 11 OOP primitives

Am Ende holt uns das OOP-Thema vom Anfang wieder ein, und wie zu erwarten, sind die Egos natürlich

nicht in den Hintergrund zu drängen. Klaus Schleisiek hält Manfred Mahlows OOP für den Stein der Weisen, und verspricht, nach kurzer Vorstellung, was das so ungefähr ist, eine Portierung auf Gforth (0.6.2) zu machen, damit man sich das ansehen kann.

Andrew Haley: What OOP needs: by example of the Proxy design pattern

Im Versuch, das Unverständnis, worum es bei OOP überhaupt geht, zu reduzieren, versucht Andrew Haley das Proxy design pattern zu erklären. Mir scheint aber, dass die Konzentration aller Beteiligten so weit nachgelassen hat, dass die Erklärung nicht wirklich verstanden wurde. Zudem sind Design-Pattern keine leichte Kost, und die Gang of Four, die das erste dicke Buch zu OOP-Design-pattern herausgegeben hat, sagt selbst „wir haben es auch nicht verstanden, als wir das Buch zum ersten Mal geschrieben haben.“ Ich habe das Buch gelesen, als es neu war (und es dazu in der Uni-Bibliothek ausgeliehen), da sind inzwischen 15 Jahre vergangen, und die Ausgabe, die ich jetzt bei Amazon bekommen habe, ist nur für Indien bestimmt ;-). Man sollte sich auch nicht ganz so sklavisch an Pattern aus der Java-Ecke halten, Forth ist eine wesentlich dynamischere Sprache, und kann bestimmt mehr, wenn man es richtig macht.

Neben dem formellen Teil der Konferenz ist der informelle mit Exkursion und Abendessen natürlich auch sehr wichtig. Die Gespräche gehen immer bis tief in die Nacht, wodurch sich der Start am nächsten Tag etwas verzögert. Zumal es durch das Standard-Treffen jetzt vier Nächte hintereinander sind, die alle etwas kurz geraten. Anton scheint in Wien sehr populär zu sein, es gibt nicht nur eine Ertlgasse im 1. Distrikt, sondern auch eine Antonshöhe, und an beiden sind wir vorbeigekommen. Das Internet im Erzherzog Rainer ist inzwischen auch kostenlos, und das Essen war überall gut.



Abbildung 4: Im Weingarten

Design Pattern und objekt-orientierte Programmierung

Bernd Paysan

STEPHEN PELC hat auf der EuroForth ein dickes Fass aufgemacht, und aufgerufen: „Standardize OOP now!“ In der darauf folgenden Diskussion hat sich gezeigt, dass jeder etwas anderes unter OOP versteht, und sich deshalb keine gemeinsame Praxis herauskristallisieren kann. Dieser Artikel unternimmt den Versuch, da etwas Licht ins Dunkel zu bringen. OOP ist „no silver bullet“, wie FRED BROOKS schreibt, es löst nicht das Problem, dass Programmieren schwer ist. Es ist aber ein brauchbares Werkzeug für eine Reihe von Problemen, die man anders noch schwerer lösen kann.

Einleitung

Die SAPIR-WHOLF-Hypothese besagt, dass man nicht denken kann, was in der eigenen Sprache nicht ausgedrückt werden kann. Die Hypothese wurde ursprünglich bezüglich natürlicher Sprachen aufgestellt, und ist im Kern ein Konstrukt aus einer Zeit, in der andere Leute kulturelle Unterschiede mit Rassismus erklärt haben¹. Sie gilt heute als widerlegt — in jeder natürlichen Sprache kann man alles ausdrücken, es ist nur manchmal umständlicher als in anderen Sprachen. Und vor allem ist es schwieriger, sich in einer Sprache auszudrücken, die man nur mäßig beherrscht — das ist dann auch der Grund, warum die beiden Herren zu ihrer Hypothese gekommen sind.

Für Programmiersprachen gilt diese Kritik aber nicht — Programmiersprachen sind künstlich, sie eröffnen tatsächlich einen Horizont zu einer neuen Denkweise, nämlich wie man einen Computer programmieren kann. Das ist etwas grundsätzlich anderes als sich mit Menschen zu unterhalten, weshalb Computersprachen auch grundsätzlich anders strukturiert sind als natürliche Sprachen. Die Denkweisen, die man mit einer Computersprache erlernt, sind neu, und eröffnen deshalb tatsächlich neue Horizonte. Die Konzepte, wie man sich in einer Computersprache ausdrückt, um gewisse Probleme elegant zu lösen, nennt man „Design Pattern.“ Das war so Mitte der 90er-Jahre der Hype schlechthin, und ich habe mir damals natürlich das Buch der „Gang of Four“ [1] aus der Uni-Bibliothek ausgeliehen und gelesen. Ich habe auch das Buch von CHRISTOPHER ALEXANDER [2] gelesen, ein Architekt, der das Konzept der Design Pattern im Zusammenhang mit Architektur entdeckt und beschrieben hat (schon in den 70ern), und dessen Buch Wegbereiter für diese Welle war — und der sehr überrascht war, dass plötzlich eine Menge Informatiker sein Buch gelesen haben. Design Pattern werden durch die Möglichkeit einer Sprache vorgegeben — das Design-Pattern-Buch, wenn man es so nennen will, für die natürlichen Möglichkeiten, die Forth von sich aus mitbringt, ist „Thinking Forth“ von LEO BRODIE [3].

Forth ist eine sehr flexible und dynamische Sprache, man kann es in alle möglichen Richtungen erweitern. Fügt man OOP zu Forth hinzu, so fügt man ein anderes Programmier-Paradigma hinzu, denn Forth selbst

hat keine OOP-Fähigkeiten. Ich habe früher öfter gehört „Forth braucht kein OOP, wir haben ja CREATE DOES>“. Das ist so etwa, wie wenn man sagt „ich brauche keine Freundin, ich habe ja ein Fahrrad.“ Man kann mit CREATE DOES> ein OOP-System implementieren, aber es selbst *ist* kein OOP. Genauso kann man mit dem Fahrrad zu seiner Freundin fahren, aber es *ist* keine Freundin. Die Philosophen bezeichnen so eine Antwort als „Kategoriefehler.“ Es ist aber eine ganz normale Tatsache, dass man Dinge, die man nicht kennt, auch nicht vermisst. Insofern ist es nicht verwunderlich, dass der typische Forth-Programmierer mit CREATE DOES> und Fahrrad zufrieden ist ;-). CREATE DOES> ist eher so etwas wie eine Closure, also das Bündeln von Daten mit einer einzigen Funktion.

OOP verspricht, komplexe Programme wartbar zu machen. Was wir durch OOP bekommen haben, sind riesige, schwerfällige Programme, die die Gigahertz-CPUs des 21. Jahrhunderts zum Schwitzen bringen, und die Terabyte-Festplatten füllen. Das ist aber kein Widerspruch, denn Programmierer neigen dazu, ihre Programme wachsen zu lassen, bis sie die Grenzen der Wartbarkeit erreicht haben. Im Umkehrschluss heißt das natürlich, dass Forth-Programme nur deshalb so kompakt und schnell sind, weil größere Programme in Forth einfach nicht mehr wartbar sind, und die Programmierer von sich aus darauf verzichten, sie mit Features zu überfrachten. Forth-Programmierer haben natürlich einen Ausweg aus dem Dilemma gefunden: es ist die Unzahl verschiedener, untereinander nicht kompatibler OOP-Erweiterungen, die auch OOP-basierte Forth-Programme ab einer gewissen Größe unwartbar machen (zumindest für Dritte, denn ab einer gewissen Größe braucht man Teams von Programmierern, um Programme weiter wachsen zu lassen), also dafür sorgen, dass sie klein und schnell bleiben.

STEPHEN PELC hat dieses Problem auf der EuroForth angesprochen, und gefordert „Standardize OOP Now!“ Das Thema führte dann am Ende der Konferenz noch zu zwei Workshops und nach der Konferenz zu weiterer Diskussion per Mail, über die ich hier ausführlicher berichte: ANDREW HALEY bringt dazu zwei Sachen ins Spiel: Zum einen die „Design Patterns“, deren Implementierung mit einem brauchbaren OOP möglich sein muss, zum anderen das Metaobject Protocol [4], mit dem die

¹ Das heißt, die beiden Herren wollten damit erklären, „warum Neger doof sind“, statt erst mal herauszufinden, ob das überhaupt so ist, oder vielleicht gar nicht erklärt werden muss. . .

Common-Lisp-Leute ihr sehr ähnliches Problem gelöst haben (auch dafür gab es einen ganzen Sack voller untereinander inkompatibler OOP-Erweiterungen), und ein Paper, das einen interessanten Ansatz zur Verallgemeinerung bringt. Stephen gefällt Forth meets Smalltalk von DOUG HOFFMAN, er möchte das als Basis verwenden.

Stephen erkennt mit seinem Vorschlag, dass gerade DOUG HOFFMAN zentrales Teil des Problems ist (auch wenn FMS nicht mehr so grauenhaft ist wie der Vorgänger Neon). Die Diskussion darüber ging dann auch noch per E-Mail nach dem Meeting weiter, wobei KLAUS SCHLEISIEK MANFRED MAHLOWS OOP als „Lösung“ anpries, und seine völlige Unkenntnis über Late Binding und vtables offenbarte. Genau das ist meiner Meinung nach das zentrale Kern-Problem von OOP in Forth: Zu viele Leute verstehen darunter lediglich Strukturen mit einem Namensraum für Funktionen. Nein, OOP ist ein ganzer Sack verschiedener Konzepte, weshalb es eben *nicht* ausreicht, mal eben 10% davon zu implementieren, denn die 10%-Lösungen, die alle jeweils unterschiedliche 10% berücksichtigen, die haben wir ja schon. Leon Wagner bestätigt diese These, denn seiner Meinung nach ist SWOOP völlig ausreichend, auch wenn es bei Late Binding ebenso unbrauchbar ist. Das Argument, das man immer hört, ist „wir/unsere Kunden schreiben damit ernsthafte Programme.“ Nein, das ist nicht die richtige Antwort, weil sich die Situation derzeit wohl so darstellt, dass jedes System eben seine eigenen Stärken und Schwächen hat, und deshalb für die zugehörige Anwendung maßgeschneidert ist — damit aber nicht durch ein beliebiges anderes ersetzt werden kann.

Ich habe mich jedenfalls mit ein paar Büchern eingedeckt, und denke jetzt erst mal darüber nach, wie man das Problem richtig löst. Code haben wir genug geschrieben, es ist jetzt Zeit, den wieder wegzuerwerfen, und das inzwischen Gelernte anzuwenden — oder, wenn da Defizite vorhanden sind, Neues zu lernen. Man muss sich auch im Klaren sein: Die bisher benutzten Systeme werden nicht verschwinden, ihre Syntax und Semantik wird erhalten bleiben. Andrews Vorschlag, ein erweiter- und modifizierbares Grundgerüst zu nehmen, auf dem man dann verschiedene Syntax und Semantik aufbauen kann, scheint mir der einzige brauchbare Ansatz zu sein. Solange ich aber von meinen Mit-Forthern beim Wort „vtable“ immer ein „das ist mir zu komplex“ höre, habe ich Bedenken — das dabei entstehende System wird dann bestimmt als eierlegende Wollmilchsau angesehen, und der Forth-Philosophie widersprechend, wird also auch nicht akzeptiert.

Instanz-Variablen, Methoden, Vererbung und Polymorphismus

Die drei Grundkonzepte von OOP sind Instanz-Variablen, Polymorphismus und Vererbung. Der Satz „Ein Objekt besteht aus Variablen und Methoden, die auf diese Variablen zugreifen“ scheint überall angekommen zu sein, das ist der gemeinsame Minimal-Konsens, es ist das Verständnis einer um Funktionen erweiterten Datenstruktur, es ist aber nur ein Teil dessen, was OOP

ausmacht. Ebenso Konsens scheint über die Fähigkeit zur Vererbung zu bestehen, also dass man diese Strukturen und Methoden nachträglich erweitern kann, wodurch eine Unterklasse entsteht, der zusätzliche Instanz-Variablen hinzugefügt werden können, und die die Methoden der Basisklasse überschreiben kann, wobei „überschreiben“ bedeutet, dass die veränderten Methoden nur für die Unterklasse gelten. Die in der Basisklasse verankerten und damit allen Unterklassen gemeinsamen Methoden nennt man „Interface,“ das ist eine Kollektion von Messages, die man den Objekten schicken kann (man sendet eine Message an ein Objekt, aufgerufen wird dann eine Methode — Message und Methode sind zwei verschiedene Sachen, der Operator, um eine Message in eine Methode zu überführen, und damit ausführbar zu machen, heißt „bind“).

Damit hat man schon ein Werkzeug, das über die natürlichen Fähigkeiten von Forth hinausgeht. KLAUS SCHLEISIEK hat einmal einen Vortrag über die verschiedenen Adressräume von Microcontrollern und DSPs gehalten, und die Forth-übliche Lösung führt zu einer Explosion der Wörternamen und unwartbaren Programmen — weil man sich bei jeder Adresse merken muss, ob die jetzt im Flash, im EEPROM, im internen RAM, im X- oder Y-RAM eines DSPs, im Code-Memory einer Harvard-Architektur, im externen RAM, im IO-Bereich, in einem über SPI oder I²C adressierten externen IO-Baustein oder wo auch immer ist. Führt man hier ein bisschen OOP ein, kann man jeder Variablen die Methoden @ und ! zuordnen, und diese jeweils für die verschiedenen Speicherbereiche unterschiedlich implementieren. Die so definierten Variablen „wissen,“ wie sie auf @ und ! zu reagieren haben, der Programmierer muss das nicht mehr selbst machen. Die Variablen sind im Quelltext durch ihren Namen referenziert, ihre Klasse ist also bekannt und die Methoden können früh gebunden werden. Wartbarkeit ist wieder erreicht, durch das Prinzip „information hiding,“ denn wo die Variable jetzt liegt, und wie sie angesprochen werden muss, ist ihr Privat-Wissen, dem Programmierer, der sie verwendet, kann das jetzt egal sein. Hier spricht man von „interfaces“ (das sind in diesem Beispiel @ und !) und „implementations“ (das sind die jeweiligen Umsetzungen für unterschiedliche Adressräume). Der Code dazu sieht ungefähr so aus (Syntax: Bernd-OOO, ich nehme an, dass die forthige Lösung für den Zugriff auf Flash und SPI schon implementiert ist, und nur noch der OOP-Wrapper darum gelegt wird, um das ganze wartbar zu machen). Zur Konflikt-Bereinigung zwischen den Methoden @ und ! und den Forth-Wörtern @ und ! gibt es das Präfix F, das explizit im Forth-Wörterbuch sucht. Man könnte auch dem Vorschlag von DOUG HOFFMAN folgen, und allen Messages einen : hinzufügen. Das ist aber ein Konzept aus einer anderen Sprache. In Forth sind Messages auch einfach nur Forth-Wörter.

```
object class memvar
  cell var addr
  method @
  method !
how:
```



```

: init ( addr -- ) addr F ! ;
: ! ( n -- )   addr F @ F ! ;
: @ ( -- n )   addr F @ F @ ;
class;

```

Ich verwende memvar im Folgenden auch gleich als Basisklasse, die hier nicht abstrakt ist, sondern konkret — mit der einfachst möglichen Implementierung.

```

memvar class flashvar
how:
: ! ( n -- )   addr F @ flash! ;
class;

```

Das Flash beschreiben funktioniert anders, lesen kann man es aber wie normalen Speicher.

```

memvar class spivar
how:
: ! ( n -- )   addr F @ spi! ;
: @ ( -- n )   addr F @ spi@ ;
class;

```

SPI liest und schreibt man über einen seriellen Datenbus. Jetzt brauchen wir noch etwas Test-Code:

```

here 0 , memvar : a
$20      spivar : b
$FFEA flashvar : c
$1234 a !
a @ c !
c @ b !
b @ hex. ( sollte $1234 geben )

```

Die meisten mir bekannten Forth-OOP-Systeme implementieren das jetzt als Early Binding. Das, was daraus entsteht, ist kein OOP, sondern „operator overloading“, der gleiche Operator auf verschiedene Datentypen angewandt wird unterschiedlich implementiert — die Datentypen sind ihrerseits aber statisch, also von vornherein bekannt.

Das ist natürlich noch nicht sonderlich flexibel — normale Variablen kann man einfach auf den Stack legen, und generische Funktionen können einfach @ und ! darauf anwenden, um etwa +! zu implementieren:

```

: +! ( n addr -- ) dup >r @ + r> ! ;

```

Man kann nun natürlich einfach +! als Methode für jeden Speicherbereich implementieren, das +! darf auch ruhig jedes Mal gleich aussehen, aber wenn @ und ! früh gebunden werden, muss der Code jedes Mal neu übersetzt werden — für jede Klasse einmal. Einfacher wäre es, @ und ! würden spät gebunden, also erst zur Laufzeit (in obigem Beispiel ist das so, weil BerndOOF tatsächlich per Default spät bindet). Diese Eigenschaft nennt man „Polymorphismus“, Unterklassen haben unterschiedliche Gestalt, und das Senden der gleichen Message führt zu verschiedenen Aktionen — wobei dem verwendenden Programm nicht bekannt sein muss, mit welcher Unterklasse es zu tun hat. Wir haben es hier also mit dynamischem Typing zu tun — der Datentyp (die Klasse) des Objekts ist erst zur Laufzeit bekannt.

Das (recht klassische und einfache) Design-Pattern, das hier verwendet wird, um das zu implementieren, ist das der „Indirektion.“ Man kann in der Informatik, so heißt

es, jedes Problem durch eine zusätzliche Indirektion lösen — also indem man einen Pointer benutzt, statt den Wert selbst.

Die Indirektion, die hier nötig ist, ist die beim Aufruf der Methode. Naheliegender ist es, einfach eine Instanz-Variable für jede Message zu definieren, und dort das xt der Methode zu speichern — wie bei einem deferred word. Man kann das so machen, es verschwendet nur Speicher, denn die Methoden sind ja für alle Objekte einer Klasse gleich.

Die Lösung dazu ist (wie in der Informatik üblich) eine Indirektion, die als „vtable“ oder „virtual method table“ bezeichnet wird. Man gibt also jedem Objekt einer Klasse einen Pointer mit, auf eine Datenstruktur, in der die Methoden einer Klasse gemeinsam verwaltet werden. Üblicherweise ist das ein Array, d.h. jede Methode ist über einen Index schnell und ziemlich direkt zugreifbar, aber man kann da auch Hash-Tables oder Listen verwenden, wenn etwa das Ziel eine Implementierung ist, bei der jedes Objekt jede Message verstehen kann, und damit ein Array als vtable zu viel Speicherplatz kosten würde. Abbildung 1 zeigt, wie das im Speicher aussieht.

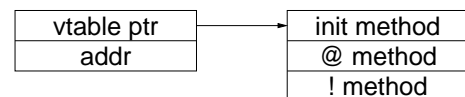


Abbildung 1: Ein Objekt der Klasse memvar und seine vtable

Deshalb, um das nochmal deutlich zu machen: Eine vtable ist deshalb getrennt vom Objekt abgelegt, um Speicherplatz zu sparen. Die entscheidende Indirektion ist nicht der Pointer auf die vtable, sondern der Pointer auf die Methode, die aus einem Forth-Wort, das im Kontext des Objekts aufgerufen wird, eine polymorphe Message macht — eine Art deferred word, nur eben nicht als Instanz-Variable, sondern für alle Instanzen einer Klasse gemeinsam.

Viele objektorientierte Sprachen verstehen unter „information hiding“, dass der Zugriff auf im Objekt intern verwendete Funktionen von außen nicht gestattet ist. Das widerspricht der Forth-Philosophy „don't bury your tools“, und muss zumindest für's Debugging abschaltbar sein. Meiner Meinung nach ist das auch ein Missverständnis, denn wirklich gemeint ist, dass man zur Kommunikation mit anderen Objekten ihre Interfaces, nicht ihre Implementierung verwenden soll. Das sind aber alles Regeln, keine Gesetze; Regeln, deren Befolgung meistens sinnvoll ist, und deren Verletzung begründet werden sollte. In Forth ist die Durchsetzung solcher Regeln durch den Compiler nicht üblich.

Design Pattern: Komposition statt Vererbung

Das Meta-Pattern schlechthin bei den Design-Pattern ist, statt Vererbung die Komposition von Objekten zu verwenden. Vererbung ist sinnvoll, um verschiedene Ausformungen einer Klasse zu bekommen; die Komposition

von Objekten ist aber flexibler, und verknüpft ganz unterschiedliche Eigenschaften. Vererbung ist ein Kopiervorgang (ungeschlechtlich, wie die Fortpflanzung von Bakterien), Komposition ist im Verhältnis dazu wie Sex — mit mehreren Teilnehmern, die ihre unterschiedliche Herkunft einbringen.

Ich möchte das einmal am Beispiel eines Dialog-Fensters in MINOS erläutern, was damit gemeint ist. Die beiden Relationen, um die es hier geht, werden mit „is-a“ und „has-a“ bezeichnet. Ein Dialog-Fenster in MINOS ist eine Komponente (das ist die Klasse, von der es abgeleitet wird). Es **hat** mehrere Sachen: Zum einen ein Display-Objekt, in dem es dargestellt wird (das ist üblicherweise ein Fenster). Zum anderen hat es Widgets, in der Regel eine ganze Menge, die ihrerseits in horizontalen und vertikalen Boxen angeordnet sind. Auch für Widgets gelten derartige Beziehungen: Ein Widget ist z.B. ein Button. Es hat eine (oder mehrere miteinander verkettete) Aktion(en), die ausgeführt werden, wenn man darauf klickt. Und es hat natürlich auch ein Display, in dem es dargestellt wird.

Die „is-a“-Beziehung wird durch Instantiierung und Vererbung erworben, einmal instantiiert lässt sie sich nicht mehr verändern (ein Button bleibt ein Button). Die „has-a“-Beziehung durch Zuweisungen, etwa durch Einhängen in die Liste einer Box. Letzteres erfolgt zur Laufzeit, und kann auch während der Laufzeit geändert werden (es können neue Elemente in eine Box eingehängt werden, oder Elemente daraus gelöscht werden). Das ist sehr dynamisch und sehr flexibel. Das dazu verwendete Konstrukt ist, wie nicht anders zu erwarten, eine Indirektion, ein Pointer auf ein Objekt.

Die meisten dieser Design-Pattern funktionieren nur mit late binding, manche brauchen sogar Mehrfachvererbung. Ein umfassendes OOP-System sollte es ermöglichen, alle Pattern auszudrücken; wenn man nur spezielle Anwendungsfälle für einige der Pattern hat, reicht manchmal eine reduzierte Version. Ich liste diese Pattern in der Reihenfolge der Gang-of-Four auf, und erkläre kurz, was damit gemeint ist, sowie welche Fähigkeiten des OOP-Systems nötig sind, um diese Pattern zu implementieren. Da Forth eine sehr dynamische Sprache ist, und die Design-Pattern sich auf C++ und Java beziehen, fehlen Design-Pattern, die man in diesen Sprachen einfach nicht machen kann. Man muss also auch dieses hier als Minimal-Konsens verstehen, und durchaus in Erwägung ziehen, dass man das auch anders (und besser) machen kann.

Creational Patterns

Diese Pattern haben das Erzeugen von Objekten zum Thema

Abstract Factory Erzeugt verwandte oder voneinander abhängige Objekte, ohne die konkrete Klasse zu spezifizieren.

Beispiel: Wir verwenden obige Klassen, um Variablen in verschiedenen Speicherbereichen zu verwalten, wollen aber den Ort unabhängig von den Variablen selbst

spezifizieren. Dazu bauen wir uns einen globalen Status, der dem aktuell verwendeten Speicherbereich entsprechend gesetzt ist, und erzeugen in der Abstract Factory immer die passenden Objekte. Wenn unser ADC jetzt vom SPI-Bereich in den I²C-Bereich verschoben wird (z.B. weil er beide Interfaces anbietet, der eine Controller aber SPI und der andere I²C verwendet), dann ändern wir nur den Bereich, und lassen den Rest des Quellcodes unverändert.

Dabei kann es verschiedene Klassen geben, die ausgewählt werden, etwa Klassen für Bytes, für 16- und 32-Bit-Wörter, die alle gemeinsam abhängig vom gerade gewählten Speicherbereich erzeugt werden können.

Das OOP-System muss Objekt-Pointer beherrschen und echten Polymorphismus, weil der Aufrufer der Abstract Factory nicht weiß, was für ein Objekt er bekommt. Es ist hilfreich zur Implementierung, wenn Klassen ebenfalls herumgereicht und als Pointer abgelegt werden können (etwa, um eine Klassenmatrix anzulegen). Im Beispiel oben werden die Objekte während der Compilation erzeugt und in anderen Teilen des Programms direkt als benannte Variable verwendet (und da das Binding nach der Erzeugung der Objekte geschieht, auch ggf. mit Early Binding), das ist etwas, was in C++ und Java so gar nicht geht.

Builder Erzeugt komplexe Objekte, etwa zum Konvertieren von einem Dateiformat in ein anderes.

Beispiel: Wir wollen ein RTF-Dokument in ASCII oder L^AT_EX konvertieren. Dazu spezifizieren wir das Ziel, und der Builder erzeugt entweder ein ASCII- oder ein L^AT_EX-Dokument-Objekt. Beide Objekte nehmen Text und Formatierungsanweisungen des RTF-Readers entgegen, der ASCII-Builder baut aber nur die Strings in die Ausgabe ein, während der L^AT_EX-Builder auch die Formatierungsanweisungen übersetzt.

Auch hier muss das OOP-System Objekt-Pointer und Polymorphismus beherrschen.

Factory Method Erzeugt Objekte, wobei die Klasse durch einen Parameter spezifiziert ist, oder in Unterklassen variiert wird, ähnlich der Abstract Factory, nur dass hier eine Methode verwendet wird, um ein Objekt zu erzeugen. Die Anforderungen an das OOP-System sind die gleichen wie bei der Abstract Factory

Prototype Hier spezifiziert man zunächst eine Instanz, den Prototypen (mit Variablen und Methoden), und erzeugt dann weitere Instanzen als Kopien davon.

Die wesentliche Forderung an das OOP-System ist, dass es den clone-Operator implementieren kann, und dass Objekte während der Laufzeit neue Instanz-Variablen und Methoden bekommen können — dann ist das Prototyp-Pattern direkt im System implementiert.

Singleton Eine Klasse, die nur genau einmal instantiiert werden kann; dieses eine Objekt muss global verfügbar sein. Die Instantiierung erfolgt beim ersten Zugriff, nicht beim Programmstart.



Die Forderung an das OOP-System hier ist, globale Variablen innerhalb einer Klasse zu ermöglichen, und Methoden, die schon vor der Instantiierung aufgerufen werden können — diese Methoden können nicht late binding sein.

Structural Patterns

Adapter Ein Adapter konvertiert die Aktionen eines Interfaces in ein anderes, um verschiedene bereits existierende Programme aneinander anzupassen.

Beispiel: Eine Grafik-Ausgabe mit einem kartesischen Koordinatensystem soll mit einem Adapter in eine Turtle-Grafik umgesetzt werden, weil ein anderes existierendes Objekt eine Turtle-Grafik erwartet. Das Adapter-Objekt wird also die Position und Richtung der Turtle als Zustandsvariablen enthalten, und aus forward/backward eine entsprechende lineto-Message berechnen.

Ein Adapter kann entweder durch Mehrfachvererbung (Class Adapter) oder durch Komposition (Object Adapter) erzeugt werden.

Bridge So ähnlich wie ein Adapter, aber von vornherein im Entwurf so vorgesehen.

Wenn die Turtle-Grafik im obigen Beispiel von vornherein Design-Ziel ist (wir wollen das als stabiles Interface für den User zur Verfügung stellen), und die Grafik-Ausgabe je nach Ausgabe-Device das mehr oder weniger direkt unterstützt (X Window System schlecht, OpenGL mächtig, PostScript gut), dann sind die Objekte, die das konvertieren, Bridges, nicht Adapter.

Composite Erlaubt es, Objekte zusammenzusetzen, und diese zusammengesetzten Objekte ihrerseits genauso wie einzelne Objekte zu behandeln.

Beispiel MINOS: Hier gibt es Widgets und Boxes, und natürlich sind Boxes ihrerseits wieder Widgets, die eben mehrere andere Widgets enthalten.

Das erfordert Objekt-Pointer und Late Binding.

Decorator Fügt zusätzliche Verantwortlichkeit dynamisch zu einem Objekt hinzu.

Beispiel MINOS: Die Aktionen, die ein Widget hat, können etwa mit einem Tooltip-Objekt dekoriert werden. Die Aktion selbst ist unverändert (ein Klick produziert das Gleiche wie ohne Tooltip), aber wenn der Benutzer die Maus über dem Widget stehen lässt, wird ein Tooltip angezeigt.

Auch hier ist man mit Objekt-Pointern und Late Binding dabei.

Facade Stellt ein „einfaches“ Interface (in einer Klasse) zur Verfügung, die in Wahrheit über eine ganze Kollektion verschiedener Objekte implementiert wird.

Beispiel MINOS: Ein Slider besteht aus Pfeilen nach oben und unten, einem abgesenkten Bereich und einem Slider darin, der den sichtbaren Ausschnitt des Dokuments repräsentiert. Nach außen stellt sich der Slider als ein Objekt dar, das die Position und den sichtbaren Ausschnitt des Dokuments erfasst und verändern kann.

Je nach Anwendungsfall kann die Facade auch mit Objekt-Instanzvariablen und Early Binding oder mit Objekt-Pointern und Late Binding implementiert werden.

Flyweight Verwendet Objekte gemeinsam, um eine große Zahl von Objekten effizient zu verwalten.

Beispiel: Die vtable eines Objektes kann man ja auch als Objekt auffassen. Wie oben beschrieben, ist es sinnvoll, alle Objekte in einer Klasse auf dieselbe vtable zeigen zu lassen, statt die vtable für jedes Objekt zu duplizieren. In einem prototyp-basierten OOP ist es aber möglich, einzelne Objekte später zu ändern, und neue Methoden hinzuzufügen — dann, aber erst dann, muss die vtable kopiert werden. Die Gang of Four gibt als Beispiel eine Textverarbeitung an, die jeden Buchstaben als Objekt (mit Bounding Box und grafischer Darstellung) speichert. Jeden Buchstaben für sich natürlich nur einmal, um Speicher zu sparen. Zwingend notwendig: Objekt-Pointer.

Proxy Ein Platzhalter für ein Objekt, um den Zugang zu kontrollieren.

Proxies werden auf vielerlei Weise eingesetzt; das Beispiel, das Andrew Haley erwähnte, ist, um das Erzeugen des Objekts zu verzögern. Eine Textverarbeitung kann einen Proxy nehmen, um das Laden und Darstellen eines Bildes zu verzögern — solange der Benutzer die Seite mit dem Bild nicht anzeigt, existiert nur der Proxy, der die Größe des Bildes dem Umbruchalgorithmus übergibt. Erst, wenn die draw-Methode des Proxies aufgerufen wird, wird das eingebundene Bild komplett geladen und dargestellt (das dauert und kostet Speicherplatz, will man längere Dokumente bearbeiten, ist es sinnvoll, das nur bei Bedarf zu erledigen — der Speicher nicht mehr sichtbarer Bilder kann dann auch wieder freigegeben werden). Auch die im Flyweight beschriebenen Buchstaben-Objekte werden ein Proxy-Objekt benötigen, das z.B. die Position speichert.

Ein Proxy kann auch Zugangsbeschränkungen implementieren, etwa verhindern, dass eine Datenbank manipuliert wird, wenn sich der User nicht eingeloggt hat. Der Proxy implementiert in diesem Fall den Login-Vorgang, und reicht schreibende Zugriffe erst nach einem erfolgreichen Login auf das Datenbankobjekt durch. Ein Proxy kann aber auch verwendet werden, um Objekte über's Netz anzusprechen.

Objekt-Pointer sind hier zwingend erforderlich; je nach Anwendungsfall kann von vornherein bekannt sein, welches Objekt der Proxy ersetzt (Early Binding möglich), oder auch nicht. Für den Anwendungsfall, mit einem Proxy Nachrichten über's Netz zu schicken, ist es sinnvoll, wenn das System Objekte serialisieren kann (also in einer Form abspeichern, die man anderswo wieder laden kann — in Fort wäre das Sourcecode).

Behavioral Patterns

Chain of Responsibility Mehrere Objekte in einer Kette bekommen die gleiche Eingabe serviert, bis eines darauf reagiert.

Beispiel MINOS: Das Text-Eingabefeld muss auf Cursor-Tasten und ähnliches reagieren. Jeder derartigen Taste ist ein Objekt zugeordnet, das entsprechend reagiert.

Beispiel Recognizer: Die Recognizer, die MATTHIAS TRUTE in amForth eingebaut hat, können als Chain of Responsibility aufgefasst werden — jeder Recognizer sieht sich das Forth-Wort an, und reagiert entsprechend — wenn es im Wörterbuch ist, wird es ausgeführt oder kompiliert, wenn es eine Zahl ist, umgewandelt, und auf den Stack gelegt bzw. als Literal kompiliert.

Dieses Pattern benötigt Object-Pointer und Late Binding, es kann auch sinnvoll sein, deferred words als Instanz-Variablen zu ermöglichen.

Command Macht aus einem Kommando ein Objekt. Die Methode, die ein command-Objekt zur Verfügung stellt, ist execute, es kann also ausgeführt werden. Was es dabei tut, ist seine Sache. Es kann auch mehrere Varianten von execute geben.

Beispiel MINOS: Die Aktionen, die ein Widget hat, rufen ganz spezifischen Code auf (im Kontext eines anderen Objekts), haben aber ein gemeinsames Interface. Kommandos sind die OO-Version von Callbacks.

Beispiel Forth: Wörter sind Kommandos; folgt man der Idee des „smart compile“, dann ist ein Wort auch tatsächlich ein richtig komplexes Objekt, das ausgeführt, interpretiert, kompiliert und mit postpone später kompiliert werden kann. Das geht über die einfache Struktur eines Command-Objekts schon deutlich hinaus.

Objekt-Pointer und Late Binding sind zwingend erforderlich, deferred words hilfreich. Auch hilfreich ist es, ein Stück Code im Kontext eines Objekt-Pointers auszuführen, also dort mehrere Methoden aufzurufen, ohne jedesmal zwischen dem Aufrufer und dem aufgerufenen Objekt umzuschalten (Multi-Message).

Interpreter Erlaubt es, eine (einfache) Sprache als abstrakten Syntax-Baum zu implementieren. Dieses Pattern ist für Forth-Nutzer wohl eher uninteressant, da wir einfache Sprachen direkt in Forth implementieren, und komplexere Sprachen einen Parser-Generator benötigen, der von dem Pattern nicht umfasst wird.

Iterator Ein Iterator durchläuft zusammengesetzte Objekte sequenziell, ohne die innere Struktur des Objekts offenzulegen (also unabhängig davon, ob es ein Array oder eine Liste ist).

Der Iterator wandelt eine Klasse ab, also ist entweder Mehrfachvererbung oder die Zuordnung eines bereits definierten Interfaces zur Klasse hilfreich. Late Binding ist ebenso zwingend erforderlich.

Mediator Ein Mediator ist ein Objekt, das die Kommunikation zwischen verschiedenen Objekten zentral behandelt, damit die einzelnen Objekte nichts voneinander wissen müssen.

Beispiel MINOS: Das Komponenten-Objekt, das Dialoge implementiert, ist auch ein Mediator zwischen den einzelnen Objekten im Dialog. Die Kommandos werden alle an den Mediator gesendet, der weiß, wie die verschiedenen Objekte miteinander interagieren.

Objekt-Pointer sind zwingend erforderlich, je nach notwendiger Flexibilität des Mediators kann aber Early Binding ausreichen.

Memento Erfasst und speichert den Zustand eines Objekts, um es später zu restaurieren.

Beispiel: Um die Undo-Operation in einem Editor zu implementieren, speichert das Memento bei jedem Backspace das gelöschte Zeichen und dessen Position im Text. Führt man später Undo aus, werden diese Zeichen wieder an der richtigen Stelle eingefügt.

Objekt-Pointer und Late Binding sind erforderlich.

Observer Wenn der Zustand eines Objekts sich ändert, werden alle abhängigen Objekte benachrichtigt.

Beispiel Facebook: Wenn man einen Eintrag auf seine Pinwand schreibt, werden alle Freunde benachrichtigt, und bekommen den Eintrag in ihren Neuigkeiten zu sehen.

Objekt-Pointer, Late Binding und zumindest Interfaces (ggf. auch Mehrfachvererbung) sind nötig.

State Dieses Pattern beschreibt eine State-Maschine, wobei für jeden Zustand eine andere Klasse verwendet wird — die Funktionen zum Ändern des Zustands sind die Methoden der Zustands-Objekte, sie ändern sich abhängig vom Zustand.

Das State-Pattern kann direkt in einem OOP-System implementiert werden, wenn es möglich ist, die Klasse des Objekts zur Laufzeit zu ändern (durch Ändern des vtable-Pointers).

Strategy Definiere eine Familie von Algorithmen, die austauschbar sind.

Beispiel: Ein Roboter sucht sich einen Weg, und verwendet dabei je nach Situation unterschiedliche Algorithmen. Auf freiem Feld kann er etwa den kürzesten Weg zum Ziel berechnen, in einem Labyrinth muss er das Terrain erkunden und eine Karte erzeugen.

Objekt-Pointer und Late Binding sind erforderlich.

Template Method Definiert das Skelett eines Algorithmus, der dann in Unterklassen verfeinert wird.

Beispiel Forth: Compiler und Interpreter parsen jeweils ein Wort aus dem Eingabestrom, bearbeiten das dann aber unterschiedlich. Statt STATE @ IF .. ELSE .. THEN zu schreiben, definiert man einen Hook, der dann in den Unterklassen compiler und interpreter implementiert wird.

Late Binding ist erforderlich.

Visitor Eine Operation auf alle Elemente eines Composite.



Beispiel MINOS: Bei der Formatierung der Objekte wird jedes Objekt in einer Box nach minimaler und maximaler Ausdehnung gefragt, und daraus berechnet, welche Position und Größe es in der Box haben wird.

Objekt-Pointer und Late Binding sind erforderlich, die Möglichkeit, mehrere Methoden im Kontext eines Objekt-Pointers hintereinander aufzurufen, hilfreich. Es ist auch hilfreich, wenn man Teile eines Wortes innerhalb einer impliziten Schleife ausführen kann — in MINOS verwende ich dafür Return-Stack-Tricks. Ebenfalls nützlich ist eine Tail-Call-Optimization, um Returnstack-Platz zu sparen.

Was bleibt zu tun

Ich habe mein OOP („BerndOOF“) schon ein paar Jahre vor der Veröffentlichung des Design-Pattern-Buchs geschrieben, und zum Teil erst hinterher so erweitert, dass die meisten Design-Pattern implementierbar sind. Manchmal nicht ganz so elegant, wie man sich das wünscht, manchmal auf eine sehr forthige Art, die zumindest den Compiler von VFX Forth überfordert (das Visitor-Pattern wird in MINOS durch eine implizite Schleife implementiert, die am Returnstack herummanipuliert). Die Implementierung eines OOP-Systems ist leider immer „Guru Code“, wie Stephen Pelc es ausdrückt; es ist nichts offensichtlich, es muss alles umfangreich dokumentiert werden; da ein neues Programmier-Paradigma implementiert wird, kann man sich nicht auf gemeinsame Kenntnisse verlassen. Das ist bei praktisch keinem existierenden OOP-System wirklich der Fall, jedenfalls nicht in dem nötigen Ausmaß. Auch wenn BerndOOF meiner Ansicht nach verwendbar ist, und die anderen OOP-Systeme für Forth zu viele Mängel haben, ist es keine Lösung für das Problem, das wir haben. Es ist auch nicht wirklich portabel, die Implementierung einer performanten Lösung ist nicht trivial, und ein signifikanter Teil des VFX-Compilers (der Inliner) musste komplett neu geschrieben werden, um das richtig zu compilieren (das war aber eine Aktion, die ohnehin notwendig war). Ich finde den Common-Lisp-Ansatz des Metaobjekt-Protokolls sehr interessant, und werde versuchen, etwas Ähnliches in Forth zu implementieren. Die Idee dahinter ist, das OOP-System selbst unter Verwendung objektorientierter Programmierung zu implementieren, und so die notwendige Flexibilität zu erreichen. Als Implementierungsbasis, also zum Bootstrappen des OOPs, werde ich mein Mini-OOP verwenden, die 12 Zeilen, die Vtables und Instanz-Variablen implementieren, mehr aber nicht. Ziel ist, dass man damit dann Syntax und Semantik bestehender OOP-Systeme, also etwa von SWOOP, FMS, BerndOOF und anderen implementieren kann, und diese

Systeme auch jederzeit erweitern kann (etwa um Mehrfachvererbung), ohne wie bisher intime Kenntnis über die jeweiligen Systeme zu haben — nur die Kenntnis des Metaobjekt-Protokolls ist natürlich zwingend nötig.

Es soll im so entstehenden System möglich sein, Klassen aus verschiedenen OOP-Systemen zu verknüpfen, und gemeinsam zu verwenden. Dabei muss es möglich sein, dass die Implementierung deutlich voneinander abweicht — BerndOOF z.B. hat eine C++/Java-artige Klassenhierarchie, in der jedes Objekt nur die Messages versteht, die es entweder geerbt oder selbst definiert hat. Andere Messages werden schon zur Compile-Zeit abgewiesen. FMS dagegen implementiert das Smalltalk-Model, bei der jedes Objekt prinzipiell jede Message versteht, zumindest über die „kannitverstan“-Methode, die aber erst zur Laufzeit ausgeführt wird, und eine Fehlermeldung produziert. Die vtables in BerndOOF implementiert man am besten als Arrays (da sie immer dicht besetzt sind, die kompakteste Darstellung im Speicher), die vtables von FMS eher als Listen oder Hashes, da sie dünn besetzt sind (die meisten Klassen implementieren nur wenige Messages, und verstehen die vielen anderen Messages nicht).

Es soll auch möglich sein, degenerierte Klassen und Objekte zu erzeugen, die eben nur Early Binding können und keinen vtable-Pointer haben, weil viele Forth-Programmierer offensichtlich damit glücklich sind. Die einzelnen Fähigkeiten eines kompletten OOP-Systems sollten getrennt voneinander implementierbar sein, und als Erweiterungen betrachtet werden können. Wenn ich also z.B. Multiple Inheritance brauche (BerndOOF kann das derzeit nicht), dann lade ich es einfach dazu.

Die notwendigen Primitives, also der Zugriff auf Instanz-Variablen über einen this/self-Pointer, die performante Ausführung von bind(message) über vtables (Arrays und Listen/Hashes), und die Erweiterbarkeit des äußeren Interpreters um Namensräume für Klassen müssen von der eigentlichen Syntax und Semantik des OOP-Systems getrennt werden können, damit man nur diesen Low-Level-Teil portieren und für jedes Forth-System extra warten muss.

Literatur

- [1] ERICH GAMMA, RICHARD HELM, RALPH JOHNSON, JOHN VLISSIDES: *Design Patterns: Elements of Reusable Object-Oriented Software*
- [2] CHRISTOPHER ALEXANDER, SARA ISHIKAWA, MURRAY SILVERSTEIN, MAX JACOBSON, INGRID FIKSDAHL-KING, SHLOMO ANGEL: *A Pattern Language*
- [3] LEO BRODIE, *Thinking Forth*
- [4] GREGOR KICZALES, JIM DES RIVIÈRES, DANIEL G. BOBROW, *The Art of the Metaobject Protocol*

Kontrollstrukturen als Colon-Definitionen und ohne die Immediate-Eigenschaft ?

Fred Behringer

Vorweg

DOS, und wenn ja, welches? Ich habe forth.com (das Turbo-Forth-System) u.a. auch in ein FAT16-Verzeichnis auf einem 1-GB-Stick gepackt. Dazu dann listing.txt (das für diesen Zweck extra so eingerichtete Listing des vorliegenden Artikels - natürlich ohne die Zeilennummern des VD-Heftes). Und das Ganze dann unter XP mit dem Explorer per forth aufgerufen. Dann über die Tastatur include listing.txt eingegeben. Ging alles bestens! Mit anderen Worten, ich konnte damit alle hier vorgeschlagenen Forth-Worte ohne Schwierigkeit auf ihr Funktionieren hin überprüfen.

Turbo-Forth-16 hat nur 25,5 KB und stellt eine Antwort auf eine Frage in dclf nach einem 'kleinen' Forth dar!

Voraussetzungen

Es möge ein Colon-Compiler zur Verfügung stehen. Das braucht kein ausgereiftes Forth-System zu sein. Ein geschrumpftes System genügt. Ulrich Hoffmann hat in [UH91] einen Artikel über 'Ein Weg, wie man Forth klein macht' geschrieben. Man vergleiche auch [FB99]. In TF83 gibt es die Möglichkeit, das Forth-System so auf die jeweilige Anwendung zuzuschneiden, dass es nur noch die tatsächlich verwendeten Forth-Worte enthält. Man könnte auch in Richtung Einwort-Compiler [JF09] denken. Als Einwort-Primitive? Warum nicht? Ich gehe von Turbo-Forth-83 in der 16-Bit-Ausführung (im Weiteren als TF83 bezeichnet) für den PC unter DOS aus. Zum Hochziehen eines neuen Forth-Systems sollen (nur) die Primitives von Willi Stricker [WS09] oder/und die jeweils daraus schon konstruierten Colon-Definitionen zugelassen werden. Das Primitive-Wort +c aus [WS09] werde durch das als Primitive gefasste d+ aus [FB11] ersetzt. Dass Konstanten beim Compilieren schon von Anfang an eingebracht werden können, betrachte ich, wie auch in [WS09] geschehen, als selbstverständlich. Es waere auf jeden Fall gut, die Überlegungen des vorliegenden Artikels in Bezug auf die gemachten Voraussetzungen alle noch einmal rigoros durchzugehen.

Ergebnisse

Die drei Primitives lit branch ?branch aus [WS09] können eingespart und durch Colon-Definitionen ersetzt werden. (lit aus [WS09] wird in TF83 als (lit) bezeichnet.) Es wird ein Hilfswort ?branch2 zum Überspringen mit fester Sprungweite 2 als Colon-Definition eingeführt. Damit gelingt es, die Worte ?dup min max abs dabs lshift rshift aus dem Kernel von [WS09] als Colon-Definitionen ohne if-then zu schreiben.

Vermutungen

Einiges deutet darauf hin, dass der gesamte Kernel aus [WS09] schon mit den hier besprochenen Mitteln, ohne Kontrollstruktur-Elemente und unter Berücksichtigung der hier genannten Primitive-Einsparungen, geschrieben werden kann. Vieles deutet darauf hin, dass sich neben den im Vorliegenden behandelten Worten begin exit again auch die restlichen Kontrollstruktur-Worte if else then until while repeat als Colon-Definitionen und ohne die Immediate-Eigenschaft schreiben lassen. Die Konstrukte if-else-then, begin-until und begin-while-repeat werden für die hier behandelten Kernel-Worte aus [WS09] auf Umwegen (über Ersatzkonstruktionen) auch schon im Vorliegenden erledigt (siehe Listing).

Mein Interesse für minimale Primitive-Sätze

wurde u.a. durch die Arbeiten [RA98], [BP98] und [WS09] geweckt. Ein paar Gedanken sind mir dazu in [FB09] eingefallen. Anregungen in Bezug auf Kontrollstrukturen als Colon-Definitionen ohne Immediate-Eigenschaft durfte ich aus [CH93] schöpfen. Die Vermeidung von Assembler in Forth-Worten wurde kürzlich in [WS11] angeschnitten. Auf die Idee mit d+ als Primitive zum Ersatz von if-then-Abfragen [FB11] kam ich über eine Bemerkung von [AN06]. In Verbindung mit Compilation und Meta-Compilation habe ich viel mit [RZ87] und [ZP88] gearbeitet. Neuerdings kam mir dazu [JF09] zu Gesicht. Bei Fragen des Forth-Standards habe ich vorwiegend unter [AF94] nachgeschlagen.

Meta-Compilation?

Liest man im Quelltext von TF83 über dessen erste Hochziehung nach, so sieht man, dass das Ganze auf eine schrittweise immer wieder neu angesetzte inkrementelle Meta-Compilation hinauslief. Was aber heißt Meta-Compilation? Blättert man im Quelltext des Meta-Compilers PHENIX von TF83 [ZP89] nach, so wird man an die Wand gedrückt: Ein gar nicht so kleiner Meta-Forth-Quelltext, der zu seiner Compilation ein ausgereiftes TF83-Forth-System benötigt. Zudem erscheinen auch schon im zu compilierenden Quelltext des hochziehenden eigentlichen Forth-Systems einzelne Meta-Compiler-Anweisungen. Ein ziemlicher Aufwand! Eigentlich ist doch der Colon-Compiler schon Compiler genug - und sollte für die inkrementelle Compilation ausreichen (?)

Cross-Metacompilation?

Allerdings muss man zugeben, dass die Trennung von Meta-Compilation und Compilation auch ihre Vorteile



hat. Beim Aufbau meines Transputer-Forth-Systems F-TP 1.0 (liegt u.a. auf dem amerikanischen FTP-Server taylorgeta.com) habe ich mich Anfang der Neunziger jedenfalls über diese Trennung gefreut. Sie machte es mir leichter, die angestrebte Meta-Compilation gleich auf eine Cross-Meta-Compilation auszudehnen: Mit Hilfe eines auf dem PC unter DOS laufenden 16-Bit-TF83, das den Metacompiler beherbergte, wurde ein 32-Bit-Forth-System für einen ganz anderen Prozessor, nämlich für den INMOS-Transputer T800, hochgezogen. Dass das nicht ohne gelegentliche Einsprengsel von Elementen des Metacompilers in den vom Metacompiler eigentlich 'automatisch' zu übersetzenden Quelltext des hochzuziehenden Forth-Systems abging, störte mich damals weniger. Einzelheiten zu den Vorschlägen finden sich (auch) im Listing.

Literatur

[AF94] ANS-Forth-Standard: The American National Standard for the Forth language (ANSI X3J14:1994).
[RA98] Allwright, Ray: From the Net: Minimal Word Sets. Forthwrite (FIGUK) 95, 3/1998.
[FB99] Behringer, Fred: So kriegt man Forth auch klein. Vierte Dimension 1/1999.

[FB09] Behringer, Fred: Drei Primitives weniger in Willi Strickers Forth-Minimalbasis. Vierte Dimension 4/2009.
[FB11] Behringer, Fred: Über Flags in Forth. Vierte Dimension 1/2011, S.33-34.
[JF09] Fox, Jeff: Man suche in Google unter 'A one-word metacompiler'.
[CH93] Haak, Coos: 'Lusstructuren maken zonder IMMEDIATE'. Vijgeblad 43 (1993), S.10. Auch unter <http://www.forth.hccnet.nl/vijgebladarchief/> - 118k .
[UH91] Hoffmann, Ulrich: Ein Weg, wie man Forth klein macht. Vierte Dimension 1/1991.
[AN06] Nijhof, Albert: E-Mail an [FB11], wiedergegeben in Übersetzung. Vierte Dimension 1/2006, S.10.
[BP98] Paysan, Bernd: Beitrag 'Aus dem Netz', besprochen in [RA98].
[WS09] Stricker, Willi: Minimaler Basis-Befehlssatz für ein Forth-System. Vierte Dimension 3/2009, S.15-17.
[WS11] Stricker, Willi: Forth-Compiler-Hilfsbefehle als High-Level-Befehle. Vierte Dimension 3/2011, S.20-21.
[RZ87] Zech, Ronald: Forth 83, Franzis-Verlag 1987.
[ZP88] Zupan, M. et M. Petremann: Méta-Compilateur Turbo-Forth PHENIX 1.0 (1988) [= Paramétrable, Homologique, Elliptique, Néoténique, Intégral, X-tra].

Listing

```
1  hex                      \ Alle Wertangaben sind hexadezimal zu verstehen.
2
3  \ Die drei Worte lit branch ?branch koennen aus dem Satz von Primitives aus
4  \ [WS09] herausgenommen werden. Ich kann sie (unter alleiniger Verwendung der
5  \ uebrigen Primitives aus [WS09]) als Colon-Definitionen fassen:
6
7  : lit ( -- n )           \ Dieses Wort ist das uebliche Wort aus F83 [RZ87]
8      r@ @ r> 2 + >r ;    \ oder TF83 oder anderen Forth-Systemen zur Behandlung
9                          \ von Konstantwerten im Quelltext, die auf den Stack
10                         \ gelegt werden sollen. Es braucht nicht als Primitive
11                         \ gefuehrt zu werden. Um ein Beispiel zu nennen: In
12                         \ der Colon-Definition : xxx lit 4711 ; wird bei
13                         \ Aufruf von xxx der Wert 4711 auf den Stack gelegt
14                         \ und danach wird an der Adresse nach der Adresse
15                         \ fortgefahren, an welcher 4711 enthalten ist. (In
16                         \ Turbo-Forth wird fuer lit die Bezeichnung (lit)
17                         \ verwendet. lit habe ich in der klammerlosen
18                         \ Schreibweise von [WS09] uebernommen.)
19
20 : branch ( -- )          \ Auch dieses Wort ist ueblich und auch dieses Wort
21     r> @ >r ;            \ braucht nicht als Primitive gefasst zu werden. Es
22                         \ Es ist fuer konstant vorgegebene (unbedingte)
23                         \ Spruenge in einer Colon-Definition zustaendig.
24                         \ Bei Aufruf des nachfolgenden Beispiels aa2 wird an
25                         \ der Adresse jener Zelle innerhalb aa2 fortgefahren,
26                         \ in der aa1 enthalten ist (Sprung nach aa1). Die
27                         \ Zelle (in aa2) hinter branch (das Wort dup wird hier
28                         \ als Platzhalter verwendet) muss vor dem Sprung mit
29                         \ der Adresse jener Zelle (in aa2) belegt worden
30                         \ sein, die das anzuspringende Ziel (hier mit Inhalt
31                         \ aa1) enthaelt.
32                         \ -----
33                         \ Selbstverstaendlich kann das Sprungziel mitten aus
34                         \ dem Geschehen heraus durch Umbelegung des aa1-Wertes
35                         \ (an der Adresse hinter branch im Wort aa2) jederzeit
36                         \ beliebig veraendert werden!
```

```

37 \ -----
38 \ Vergleiche das Wort then in if-then-Strukturen fuer
39 \ diese Art von Spruengen. (Fuer die Auszaehlung im
40 \ Dictionary: Der Wert 4 im Beispiel aa2 wird vom
41 \ Colon-Compiler als lit 4 compiliert und benoetigt
42 \ also 4 Bytes - und nicht etwa nur 2 Bytes!)
43 \ Verwendung nur innerhalb von Colon-Definitionen!
44
45 : aa1 ." Alles klar!" ;
46 : aa2 branch dup 4 . aa1 ;
47 ' aa2 >body 0a + ' aa2 >body 2 + !
48
49 \ Das Wort ?branch stellt, anders als branch, eine Moeglichkeit zur Verzweigung
50 \ (zu bedingten Spruengen) dar. Um auch das ohne Assembler und ohne die
51 \ Immediate-Eigenschaft bewaeltigen zu koennen, mache ich von der Idee aus
52 \ [FB11] Gebrauch, das Wort d+ als Primitive im Austausch gegen +c aus [WS09]
53 \ als Primitive zu verwenden. Ausserdem benoetige ich fuer ?branch noch einige
54 \ andere Dinge, die ich zunaechst einmal alle dem Wort ?branch vorziehen darf.
55
56 \ In Willi Strickers VD-Arbeit [WS09] wird +c als Primitive verwendet, um dem
57 \ Umstand Rechnung zu tragen, dass es in Forth keine High-Level-Flags gibt.
58 \ Das Wort +c ist in Forth-Systemen unueblich. In [FB11] konnte ich zeigen,
59 \ dass bei Verwendung des ANS-Forth-Wortes d+ als Primitive ein vollgueltiger
60 \ Ersatz fuer +c geschaffen werden kann. Den Trick mit d+ zur Herausschaelung
61 \ von einfach-genauen Abfragen auf Gleichheit mit 0 habe ich aus einer
62 \ Bemerkung von Albert Nijhof gelernt [AN06].
63
64 \ Im vorliegenden Artikel leistet mir d+ gute Dienste, um bedingte Spruenge in
65 \ High-Level-Forth darzustellen. Fuer Nachpruefzwecke ist es nicht unbedingt
66 \ noetig, d+ als Primitive (d.h. als Code-Definition) gefasst zu haben. Ich
67 \ brauche hier aber d+ zum Nachweis dessen, dass beim Minimal-Primitive-Satz
68 \ von Willi Stricker und dessen 'Kernel' [WS09] das Primitive ?branch einfach
69 \ weggelassen werden kann.
70
71 \ Ich darf die TF83-Code-Definition fuer d+ aus [FB11] wiederholen und ich
72 \ verwende im weiteren Verlauf d+ anstelle des Strickerschen +c als Primitive.
73 \ Das +c (mit den Teilaspekten + und c) sei wie in [FB11] aus d+ hergeleitet.
74
75 \ Die folgenden Stackparameter d1, d2 und d1+d2 fuer das unter TF83 gefasste
76 \ d+ sind Punktzahlen, also 32 Bit breit (doppelt-genau).
77
78 code d+ ( d1 d2 -- d1+d2 )
79     ax pop  bx pop  cx pop  dx pop  cx push  dx push  ax push  bx push
80     66 c,  ax pop  66 c,  bx pop  66 c,  ax bx add  66 c,  bx push
81     ax pop  bx pop  ax push bx push  next end-code
82
83 \ Das Forth-Wort d+ hat gegenueber +c aus [WS09] den Vorteil, dass es sich
84 \ hier um ein (ganz normales) ANS-Forth-Wort (aus dem Double-Word-Set) [AF94]
85 \ handelt. (In [WS09], wie in vielen anderen Systemen auch, wurde das Wort d+
86 \ aus dem Grundsystem von Primitives hergeleitet.)
87
88 \ Neben den Primitives lit und branch enthaelt [WS09] noch das (fuer bedingte
89 \ Vorwaertsspruenge (if-then etc.) wichtige) Wort ?branch. Auch das kann als
90 \ Colon-Definition geschrieben werden - und kommt dabei ohne die Eigenschaft
91 \ 'immediate' aus. Es kann also im Satz von Primitives in [WS09] weggelassen
92 \ werden. Schwierigkeiten hat mir (zunaechst) der Umstand gemacht, dass ich
93 \ nicht ohne zwei unterschiedliche Ruecksprung-Ebenen (fuer Sprung oder
94 \ Nicht-Sprung) auskam und dafuer ein Hilfswort einfuehren musste, fuer das
95 \ es auf den ersten Blick keine rechte Rechtfertigung zu geben scheint.
96
97 \ Auf die Frage, die mich eigentlich interessierte, naemlich, ob es moeglich
98 \ ist, den vollen if-then-Konstrukt mit seinen Derivaten als Colon-Definition
99 \ zu schreiben, habe ich bisher leider noch keine zufriedenstellende Antwort
100 \ gefunden: Wie soll man mit einem Einpass-Compiler beim Compilieren (mit dem
101 \ Colon-Compiler) ein Sprungziel an den Anfang des if-then-Konstrukts in die
102 \ diesen enthaltende Colon-Definition setzen, das erst nach der Compilation

```



```
103 \ von if-then - nach dem Abschluss durch then - zur Verfuegung steht (erst
104 \ dann zur Verfuegung stehen kann)? Ich darf diese Frage fuer spaeter
105 \ zurueckstellen.
106
107 \ Als Vorarbeit darf ich das Hilfswort ?branch2 (zur Einleitung von bedingten
108 \ Vorwaertsspruengen mit fester Sprungweite 2 in Colon-Definitionen)
109 \ vorschlagen. (Mir geht es hier in erster Linie um einen minimalen Satz von
110 \ Primitives, nicht so sehr um die Minimierung der Anzahl von benoetigten
111 \ Colon-Definitionen fuer den Kernel aus [WS09]). Die Bedingung (fuer den
112 \ Sprung) holt sich ?branch2 vom Stack: Sprung erfolgt bei fl=-1 (-1=true).
113
114 \ Das fuer ?branch eigentlich benoetigte Hilfswort ist (hier bei mir) nicht so
115 \ sehr ?branch2, sondern vielmehr (branch) - die Klammern gehoeren absichtlich
116 \ zum Wort. Andererseits hilft mir ?branch2 beim Aufbau von ?branch (also
117 \ Hilfswort fuers Hilfswort). Und zudem entwickelt ?branch2 ein Eigenleben
118 \ bei der Konstruktion von ?dup min max dabs lshift rshift ohne 'immediate'
119 \ und ohne Assembler (als Colon-Definitionen - siehe weiter unten).
120
121 \ Vorher erst noch das Wort 0= aus [WS09] als Colon-Definition. Es benoetigt
122 \ neben dem Primitive d+ aus [FB11] das Primitive drop und die Kernel-Worte
123 \ swap und - (minus) aus [WS09].
124
125 : 0= ( n1 -- n2 )      \ n1=0 -> n2=-1 ; sonst n2=0
126   0 -1 0 d+ swap
127   drop 1 - ;          \ 0= hat die uebliche Wirkung: Will man generell aus
128                        \ dem nicht eindeutig gegebenen Logik-Stackwert '<>0'
129                        \ den fuer viele Zwecke genehmeren Wert -1 (fuer true)
130                        \ machen, dann erreicht man das, wie ueblich, auch
131                        \ hier per 0= 0= . Das ist ein Beispiel, bei welchem
132                        \ d+ in einer Verzweigung hilft, Assembler-Code zu
133                        \ umgehen.
134
135 : ?branch2 ( fl -- )    \ Dieses Wort sorgt in einer Colon-Definition dafuer,
136   0= 0 -1 0 d+ swap     \ dass das Wort, welches unmittelbar auf eben dieses
137   drop dup +            \ ?branch2 folgt, uebersprungen wird, wenn fl=0. Ist
138   r> + >r ;             \ dagegen fl<>0, dann wird nichts uebersprungen. Bei
139                        \ Anwendung dieser Ueberlegung auf das unten stehende
140                        \ Wort ?branch beachte man unbedingt, dass das dem
141                        \ ?branch2 vorangestellte Wort 0= die Stackwirkung und
142                        \ damit die Wirkung von ?branch2 umkehrt. Im unten
143                        \ aufgefuehrten Wort ?branch wird also (branch)
144                        \ ausgefuehrt, wenn fl=0, und aber uebersprungen, wenn
145                        \ fl<>0.
146
147                        \ Achtung: Im Gegensatz zum ueblichen Wort ?branch
148                        \ wird hinter ?branch2 kein Sprungziel aufbewahrt:
149                        \ Die Sprungweite ist bei ?branch2 zu 2 vorgegeben.
150                        \ Eine Erweiterung zu ?branchc, wobei c eine fest
151                        \ vorgegebene Zweierpotenz ist, bietet sich sofort an.
152
153 : (branch) ( -- )      \ Springt an eine (vorgegebene) Ziel-Adresse.
154   r> r> @ >r >r ;      \ Zusammenwirken wie im folgenden Beispiel bb1, mit
155                        \ dem in bb1 enthaltenen ?branch, welches wiederum
156                        \ eben dieses (branch) enthaelt. Man kann sagen,
157                        \ (branch) wirkt wie branch, aber mit einer weiteren
158                        \ (mit einer dazwischengeschobenen) Ruecksprung-Ebene.
159
160 : ?branch ( fl -- )    \ In der aufrufenden Colon-Definition (als Beispiel
161   0= dup                \ nehme ich bb1) wird der Speicherplatz hinter ?branch
162   ?branch2 (branch)     \ (den ich durch dup fuer die Belegung mit dem
163   dup ?branch2 dup       \ Sprungziel frei gehalten habe) uebersprungen, wenn
164   drop drop ;           \ fl<>0. Ist dagegen fl=0, dann nicht.
165                        \ Die Erklaerung von ?branch darf ich zur Uebung
166                        \ stellen. Geht es mit weniger dups und weniger drops?
167 \ Beispiel:
168
```

```

169 : bb1 ( fl -- )
170   ?branch dup      \ dup = Platzhalter zur Aufbewahrung des Sprungziels
171   4 . 5 . 6 . ;    \ fl=0 -> 6 ok fl=-1 -> 4 5 6 ok
172
173 \ An Stelle von dup laesst sich auch mit anderen 'Platzhaltern' arbeiten. Ich
174 \ habe das mit swap und drop nachgeprueft. Ich konnte so die Verwendung des
175 \ Compiler-Kommas als Primitive vermeiden. Letzteres ist nicht Bestandteil
176 \ des Satzes von Primitives aus [WS09].
177
178 \ Im Moment noch (als Notmassnahme der Vorbereitung zur manuellen Einbringung
179 \ des Sprungziels in den Platzhalter im Beispiel bb1):
180
181 ' bb1 >body 10 + ' bb1 >body 2 + ! ( [ret] )
182
183
184 \ Auf dem Weg zum Thema des Titels
185 \ -----
186
187 \ Es folgen die von Coos Haak vorgeschlagenen Kontrollstruktur-Worte, die als
188 \ Colon-Definitionen gefasst und nicht immediate sind [CH93]. Eben diese Worte
189 \ haben mich dazu veranlasst, ueber 'High-Level-Forth' und den nach aussen hin
190 \ nicht erkennbaren Unterschied zwischen immediate und nicht-immediate
191 \ nachzudenken.
192
193 : begin ( -- ) r@ >r ; \ Kopiert die Adresse nach begin auf den Returnstack.
194 : again ( -- ) r> drop \ Entfernt die Adresse nach again vom Returnstack.
195   r@ >r ; \ Springt an die Adresse nach begin.
196 : exit ( -- ) r> drop \ Entfernt die Adresse nach begin vom Returnstack.
197   r> drop ; \ Das wirkt genau so wie das urspruengliche exit.
198
199 \ Eine Schattenseite tut sich insofern auf, als dieses exit nicht ganz das von
200 \ TF83 und anderswoher gelaefige exit ist.
201
202 \ Fortfuehrung der Idee von Coos Haak
203 \ -----
204
205 \ Das folgende Beispiel zeigt eine Moeglichkeit eines (zugegeben 'primitiven')
206 \ begin-until-Konstrukts, das ohne Assembler und ohne Immediate-Worte auskommt.
207
208 : cc1 4 begin 1 + dup . dup 47 = ?branch2 exit again ;
209
210 \ Dieses Beispiel bildet die Zahlen 5-47 auf dem Bildschirm ab. Es zeigt die
211 \ Brauchbarkeit des oben eingefuehrten Wortes ?branch2 und die Brauchbarkeit
212 \ der Haakschen Worte begin exit again. Erwarten wuerde man hier eine
213 \ Zusammenfassung der Wortfolge ?branch2 exit again zum ueblichen until (im
214 \ vorliegenden Zusammenhang ohne Assembler und ohne Immediate). Doch dazu
215 \ fehlen mir noch ein paar Ueberlegungen. Ich verschiebe das auf spaeter.
216
217 \ Eine Unsauberkeit besteht darin, dass beispielsweise im leicht erweiterten
218 \ Beispiel
219
220 : cc2 4 begin 1 + dup . dup 47 = ?branch2 exit again drop ;
221
222 \ immer noch der Abfrage-Ueberhang 47 auf dem Stack zurueckbleibt. Das drop
223 \ kommt nach Aussprung aus begin-until nicht mehr zur Wirkung. Ganz so leicht
224 \ ist es mit until also nicht!
225
226 \ Weitere Worte aus [WS09], hier ohne if und then
227 \ -----
228
229 \ In Willi Strickers Kernel aus [WS09] war ?dup (mit der Nummer 11:) das erste
230 \ Wort, das mich nachdenklich stimmte: Wo kommen if und then her? In den
231 \ Primitives von [WS09] sind sie nicht enthalten. Man muesste sie ueber die
232 \ zugelassenen Primitives oder/und die schon definierten Colon-Definitionen
233 \ erst einmal definieren. Zugelassen sein sollte zwar ein funktionsfaehiger
234 \ Colon-Compiler (irgendeiner), aber noch nicht unbedingt ein vollstaendiges

```



```
235 \ Forth-System. Und was, wenn der Colon-Compiler 'nur' als Primitive zur
236 \ Verfuegung steht (ein Ein-Wort-Compiler [JF09], irgendwie in Form einer
237 \ ausfuehrbaren DOS-Datei)?
238
239 \ Ich gebe im Folgenden ein paar Worte aus dem Kernel von [WS09] an, bei
240 \ welchen ich ohne if und then (und natuerlich auch ohne Assembler) auskomme.
241 \ Vollstaendigkeit wird (hier noch) nicht angestrebt. Man beachte, dass ich
242 \ dabei kein Immediate-Wort) benoetige.
243
244 : compile ( -- ) r>      \ Das ist genau die Definition von compile aus TF83.
245   dup 2 + >r @ , ;      \ Sie nutzt die Ruecksprung-Adressen-Manipulation.
246                           \ Fuer dieses Wort, das bestechend einfach aussieht,
247                           \ hier aber eigentlich sonst gar nicht verwendet wird,
248                           \ muesste man den Strickerschen Minimal-Primitive-Satz
249                           \ um das Wort , (oder zumindest um c,) erweitern.
250
251 \ Die folgenden Worte aus [WS09], die ich hier ohne if-then darstelle, koennen
252 \ als weitere Beispiele fuer eine Anwendung von ?branch2 dienen.
253
254 : ?dup ( n -- n n ) dup      ?branch2 dup      ;
255 : min ( n1 n2 -- ) over over > ?branch2 swap drop ;
256 : max ( n1 n2 -- ) over over < ?branch2 swap drop ;
257 : abs ( n -- !n! ) dup negate max      ;
258 : dabs ( d -- !d! ) dup 0<      ?branch2 dnegate      ;
259
260 \ Und abschliessend noch lshift und rshift. In TF83 gibt es weder lshift noch
261 \ rshift. Es handelt sich dabei aber um ANS-Core-Worte [AF94], die im Kernel
262 \ von [WS09] auf jeden Fall ihre Berechtigung haben.
263
264
265 : (lshift) begin dup 0= ?branch2 exit 1 - swap dup + swap again ;
266 : lshift ( n1 u -- n2 ) (lshift) drop      ;
267
268 : (rshift) begin dup 0= ?branch2 exit 1 - swap u2/ swap again ;
269 : rshift ( n1 u -- n2 ) (rshift) drop      ;
270
271 \ Ich benoetige hier die Hilfsworte (lshift) und (rshift). Geht es auch ohne
272 \ diese Hilfsworte? Die Schwierigkeiten liegen im Wort exit. exit springt
273 \ nicht etwa nur hinter again, sondern hinter das Semikolon, es verlaesst
274 \ also gleich die ganze Colon-Definition. Und das liegt nicht speziell am
275 \ exit aus [CH93]: Mit dem exit aus F83 oder TF83 oder aus ANS-Forth
276 \ bestuende dasselbe Dilemma. Schade, denn das Ueberspringen per ?branch2
277 \ waere eine einpraegsames Methode! Aber nicht so schlimm, denn Assembler-Code
278 \ wird auch hier immer noch nicht benoetigt. Und darauf kam es mir an.
279
280 \ -----
281
282 \ Bei einer Klausur (liegt schon weit zurueck) kommen einem die besten
283 \ Einfaele, wenn es Zeit ist abzugeben. Beim Schreiben eines VD-Artikels
284 \ ist es aehnlich...:
285
286 \ Schleifen, die sich nur auf schon durchlaufene Stellen im Quelltext
287 \ beziehen, muessten eigentlich alle nach dem hier besprochenen Schema
288 \ behandelbar sein. Dazu gehoeren (neben dem oben erwaehnten begin-again
289 \ aus [CH93]) auch begin-until und begin-while-repeat. Bei begin-until
290 \ haette ich es beinahe schon geschafft. Aber nur beinahe! Die weitere
291 \ Beschaeftigungsrichtung ist damit jedoch bereits vorgegeben.
292
293 \ Turbo-Forth-Feinheiten: TF83 ist eigentlich auf ein Zusammenarbeiten mit
294 \ ansi.sys in der config.sys eingestellt. Unter XP und hoeher sind diese
295 \ Begriffe nicht mehr bekannt. Was tun? Aus dem Internet ansi.com holen und
296 \ unter XP aufrufen, geht nicht. XP weigert sich. Man kann aber nach unter
297 \ XP aufgerufenem Turbo-Forth die 'Attribute' auch einfach per ctrl-F1
298 \ off-schalten. Dadurch geht nichts Wesentliches verloren - aber der
299 \ anfaengliche Bildschirmsalat verschwindet.
```


Neues von VFX Forth

Stephen Pelc

Gefragt, was sich beim VFX Forth tut, war Stephen so freundlich, einige Notizen zur jüngsten Entwicklung des VFX-Forth-Systems und seiner Werkzeuge zu übermitteln, die hier übersetzt wiedergegeben sind. mk

Die Cross Platform GUI

MPE hatte nach einer geeigneten GUI-Bibliothek Ausschau gehalten, die für verschiedene Betriebssysteme tauglich sein sollte. Und vor einigen Jahren haben wir uns für GTK+ entschieden. Und da es nun VFX Forth für Windows, Mac OS X und Linux gibt, war es an der Zeit, sich ausgiebiger dem Prototyp des Linux-Interfaces zuzuwenden. Zur selben Zeit wünschte ein Kunde sich dringend ein grafisches Interface zurück, das ähnlich seines geliebten Borland-BGI-Systems aus der DOS-Zeit funktionieren sollte. Mit etwas Überlegung fanden wir heraus, dass es durchaus möglich ist, mit einer cross platform GUI portable Grafik zu erzeugen.

Die drei Bilder sind einmal mit Windows, dann Mac OS X und schließlich Linux gemacht worden. Alle drei wurden erzeugt, indem der gleiche Quellcode ohne irgendeine Änderung kompiliert worden ist. Das Bibliotheks-Interface enthält eine kleine Anzahl bedingter Kompilationen für einige Dateinamen aus der Sammlung, doch das ist auch schon alles, was die Betriebssysteme unterscheidet.



Abbildung 1: Linux-Beispiel

Die GUI ist mit Hilfe des Glade-GUI-Designers entworfen worden, der daraus eine XML-Beschreibung der GUI erstellte. Diese XML-Beschreibung wurde in den Builder

geladen. Der Code zeigt als Beispiel einen callback (eine Aktion, die benötigt wird, wenn ein bestimmtes Ereignis eintritt) und den Code, der das Display erzeugt.

```
2 0 CCBproc: on_about1_activate \ *widget *user -- ; -- entry
\ *G Callback to run the About dialog.
2drop MainAbout runDlg drop
;
```

Der obige Code erzeugt einen callback in der Form einer C-Konvention, so wie auch GTK sie benötigt. Der Name des callbacks wurde auch mit Glade definiert. Ein callback im Builder verknüpft den Glade-Namen mit dem Forth-callback-Namen. In diesem Falle startet der About-Dialog, wenn der "Help-About"-Knopf angeklickt wird.

```
: showGraphics3 \ --
\ *G Test word to read a Glade Builder XML file and run the
\ ** GUI it describes.
initGTK +gtimer \ start GTK and timer

z" gdemo3bld.ui" loadBuilderXML \ load GTK builder design
0= abort" Can't load builder file"
z" window1" builderObject to window1 \ get handles
z" drawingarea1" builderObject to DrawingArea1
z" aboutdialog1" builderObject to MainAbout
freeBuilder

window1 gtk_widget_show_all \ show design
DrawingArea1 win2 initGWin \ initialise graphics area
win2 enable-graphics win2 onto dirty
black cleardevice \ drawing commands
white 50 circle
green 200 100 line
red 30 30 at 70 30 filled rectangle
80 70 at 100 40 blue filled ellipse
;
```

Der Cortex-Mx-Cross-Compiler

Die Cortex-Mx-Prozessoren verwenden eine erweiterte Version des ARM-Thumb-Befehlssatzes. Zunächst wurde dieser originale Thumb-1-Befehlssatz und seine 16-Bit-Opcodes benutzt, um die Forth-Decodierung in ARM-32-Bit-Befehle zu übersetzen. Der Thumb-1-Befehlssatz war aber für sich genommen dafür nicht mächtig genug und musste daher um einige ARM-32-Bit-Befehle ergänzt werden.

Für die Cortex-M0/M1-CPUs wurde daher der Thumb-1-Befehlssatz zu einem eigenständigen, aber minimalen Befehlssatz ergänzt. Die Cortex-M0-CPUs sind hauptsächlich für kostensparenden Ersatz von 8-Bit-CPUs gemacht, manche Nuvoton-Bauteile kann man bei gewissen Mengen schon für 0,55 US\$ haben. Ihr niedriger Preis wird aber durch einen ziemlich hässlichen Befehlssatz mit niedriger Leistung erkauft. Doch ungeachtet dessen leistet der VFX-Code-Generator auch dabei nun gute Arbeit.



GTK/GDK demo

File Edit View Help

VFX About GTK/GDK demo

mpe

GTK/GDK demo 1.0

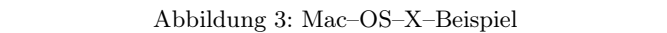
VFX Forth Glade Test Program

(c) 2008, 2011 MicroProcessor Engineering

[MPE Website](#)

Credits Licence Close

OK Cancel Random Failure



Stephen Pelc
stephen@mpeforth.com
MicroProcessor Engineering
+44 (0)23 8063 1441

<http://www.mpeforth.com/>



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, im Sommer (Mai–September) im Chilli Asia Dachauer Str. 151, 80335 München, im Winter in der Kukul-Kneipe, Weiltstraße 140, 80995 München (Feldmoching-Hasenberg) oder in der Pizzeria gegenüber (La Capannina, Weiltstr. 142).

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

8.–11. März 2012 Forth-Tagung



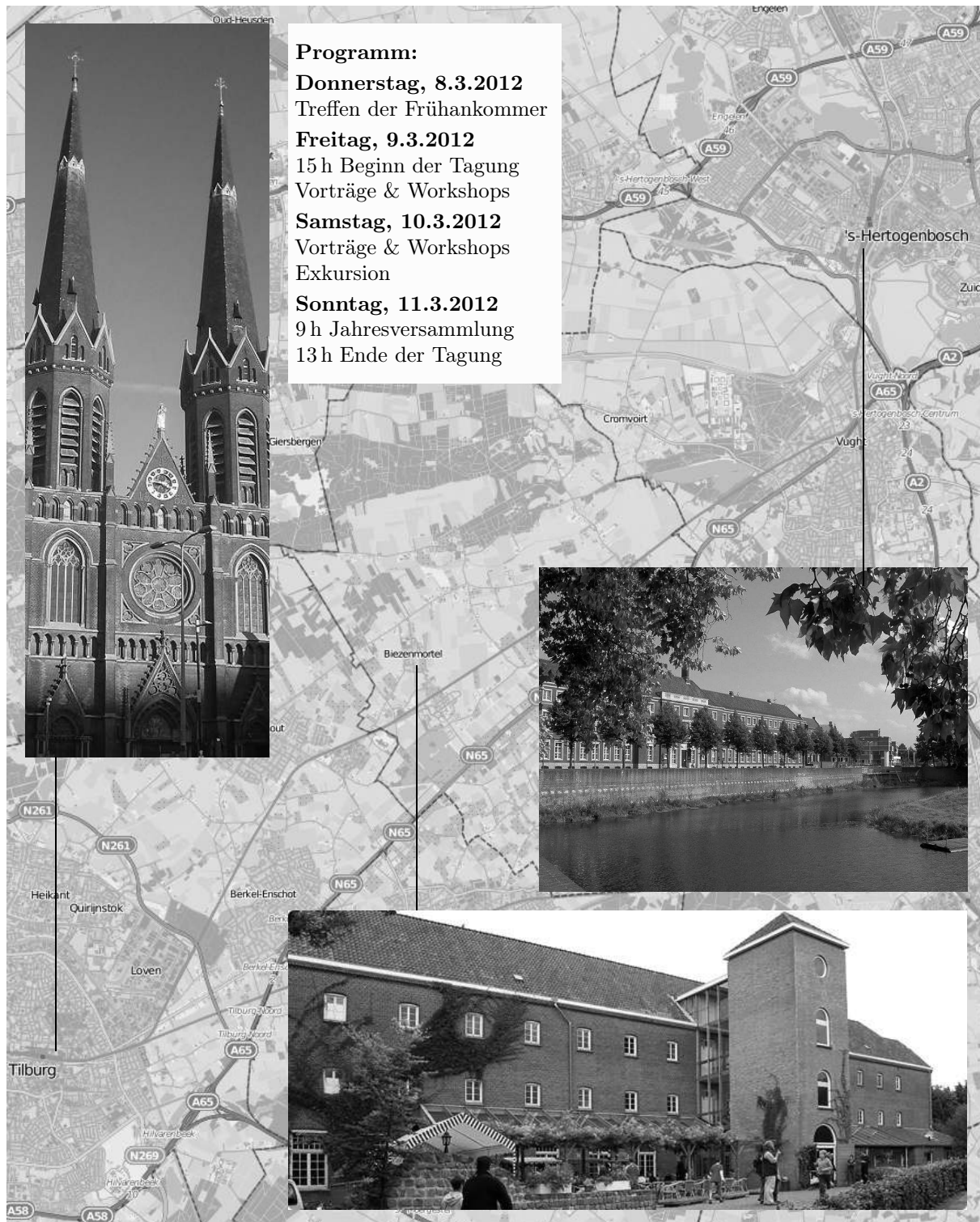
Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen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

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



Einladung zur
Forth-Tagung 2012 vom 8. bis 11. März 2012
im Beukenhof

Capucijnenstraat 46 · 5074 PJ Biezenmortel · Noord-Brabant · Niederlande
<http://www.denieuweklasse.nl/recreatief/accommodaties/beukenhof>



Programm:

Donnerstag, 8.3.2012

Treffen der Frühankommer

Freitag, 9.3.2012

15 h Beginn der Tagung

Vorträge & Workshops

Samstag, 10.3.2012

Vorträge & Workshops

Exkursion

Sonntag, 11.3.2012

9 h Jahresversammlung

13 h Ende der Tagung

Anreise siehe: <http://www.openstreetmap.org/?lat=51.62313&lon=5.18049&zoom=16&layers=M>

Quellen: www.denieuweklasse.nl/recreatief/accommodaties/beukenhof, openstreetmap.org, wikimedia.org