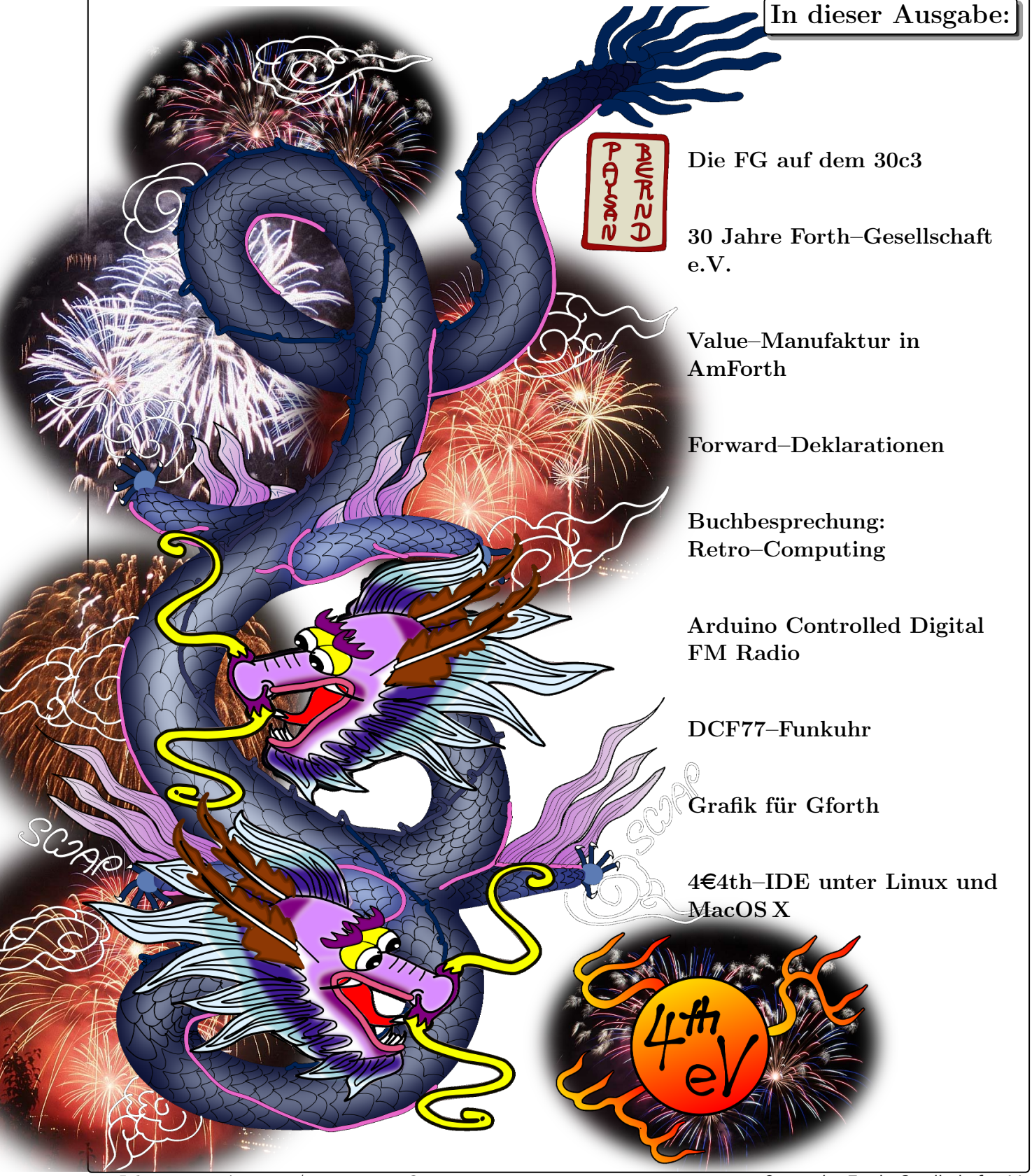
Vierte Dimension

1/2014

Das Forth–Magazin

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

In dieser Ausgabe:



Dienstleistungen und Produkte fördernder Mitglieder des Vereins

DAS RÄTSEL DES MONATS

Gesucht ist die Definition des neuen Forth - Wortes erst . Erst hinterläßt eine Zahl auf dem Stack. Vor der Ausführung weiß erst bereits, was für ein Ergebnis produziert werden wird und es benutzt diese prognostische Vermutung, um die Zahl zu berechnen.

```
: erst  ( - Zahl ) ??? ;
```

Dem Gewinner winkt eine bahnbrechende Erfindung.
Lösungen nimmt die Redaktion entgegen.

Schon in der ersten Ausgabe gab es ein Rätsel, welches unseres Wissens nie gelöst wurde.

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

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

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00€ + Porto u. Verpackung

Liebe Leser,

dieses Jahr gibt es den 30. Geburtstag der Forth–Gesellschaft e.V. zu feiern. Diese Ausgabe der *Vierten Dimension* macht sozusagen den Anfang. Die Jahresversammlung naht ebenfalls und dürfte den Geburtstag gebührend aufgreifen. Aus ebendiesem Anlass habe ich etliche Stunden im Archiv der *Vierten Dimension* gestöbert. Ja, wir haben alle Ausgaben als pdf–Dateien auf der Webseite des Vereins. Beim Stöbern finden sich längst vergessen geglaubte Dinge wieder: Es gab damals kein Internet! Jedenfalls nicht in der Form, in der es heute fast jedem hier aus der Gegend aus seinem Winzig–Funk–Telefon (sowas hat früher einen Kofferraum belegt) bekannt ist. Etliche Leute, die mir heute persönlich bekannt sind, waren damals schon in Sachen Weltherrschaft, äh Forth–Gesellschaft aktiv. Andererseits kann ich mir eine Mitgliederzahl von 500 so gar nicht vorstellen — ok, es waren *Interessenten* im Gefolge der Null–Nummer der Vierten Dimension (Mai 1984).

Ebenfalls anlässlich des Geburtstags habe ich per email herumgefragt, wie die Mitglieder an die Forth–Gesellschaft geraten sind, und was sie mit Forth denn damals so taten oder heute tun. Für die Antworten möchte ich mich ausdrücklich bedanken und weitere Leser ermuntern, uns ein paar Zeilen zu schicken.

Es gibt ein Gerücht, welches besagt, dass die *Vierte Dimension* die letzte gedruckte Zeitschrift zu diesem Thema sei. Wenn das stimmt, dann ist das beeindruckend, denn auch diese Ausgabe wurde nur durch den Fleiß der Autoren und Mithelfer möglich — und das in unterschiedlicher Form seit 30 Jahren! Auf der letzten Jahrestagung hatte ich angeregt, auch englisch–sprachige Artikel aufzunehmen, nicht zuletzt in der Hoffnung, den Kreis der interessierten Leser ein wenig zu vergrößern. Für diese Ausgabe konnten wir Craigh Lindley gewinnen, sein UKW–Radio vorzustellen. Es ist in AmForth auf einem Arduino realisiert und zweifellos von hohem Gebrauchswert.

Der Editionsprozess hat diesmal — in bester Forth–Tradition — einen direkt gefädelten Artikel hervorgebracht. Allerdings mit je einem Faden für vorwärts und rückwärts. Das erinnert mich daran, dass ich heute morgen unserem Neuen im Geschäft erklärt habe, dass eine (doppelt) verkettete Liste nixx anderes ist als ein paar Adressen im Speicher — die Bedeutung der Inhalte legt der Programmierer fest, sonst niemand. Und bei Forth ist das richtig gut nachvollziehbar.

Ach so, Werbung ist schon vorbei? Na dann! Ich hoffe, dass unsere Leser auch in dieser Ausgabe das eine oder andere Interessante finden und wünsche viel Spaß beim Lesen.

Erich Wälde, the guy who calls himself an Anfänger

Archiv der VD: http://forth-ev.de/filemgmt/viewcat.php?cid=2

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
http://fossil.forth-ev.de/vd-2014-01

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

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

4e4th auf MAC OS X dank Virtualisierung

Andrew Reid (siehe VD Heft 2013-03, MSP430 LaunchPad Educational Environment) hat seine Anwendungen rund um das MSP430–LaunchPad, 4e4th und die 4e4th–IDE inzwischen auch in der Mac OS X Umgebung ausprobiert: *"4e4th und alle Anwendungen einschließlich des Graphwriter laufen auch in dieser Umgebung mittels "Parallels", einem Programm, das es mir erlaubt, mich sehr einfach zwischen den Betriebsystemen hin und her zu bewegen."*
(Quelle: Andrews email vom 15.01.2014)

Siehe auch den Artikel *4€4th* auf Seite 32.

db, mk, cas

mecrisp: Loran–C Radionavigation

> *I wish to announce a software defined Loran-C longwave radio navigation receiver running on STM32F407 ! It is written in Forth with pieces of assembler. If you happen to live somewhere on earth with Loran-C signal coverage like Europe and parts of Asia, I would like to get in contact with you for beta testing out in the wild.*
>
> *The first experimental release is available now on download section on*
> *http://mecrisp.sourceforge.net/*
>
> *Best wishes, Matthias*

(Quelle: email von Matthias Koch, 13.1.2014)

Anmerkung der Redaktion: Loran–C ist ein Radionavigationssystem von etwa 1958. Es arbeitet auf einer Frequenz von 100 kHz und ist mit einfachen Mitteln zu empfangen. Die Position des Empfängers aus dem empfangenen Signal zu destillieren, ist etwas aufwendiger. Das vorgestellte Programm bietet dafür eine Lösung. Ebenfalls lesenswert:
[1] `http://de.wikipedia.org/wiki/LORAN-C`
[2] `http://phk.freebsd.dk/loran-c/`

mk, ew

noForth auf dem MSP430

Hallo allesamt,

hier fünf Starter-Kits, auf welchen noForth1400 für den MSP430 läuft, in Reih und Glied. Die neue Version von noForth liegt nun in kompakter Form als `noForth C1400-G` vor. Vor allem auf dem großen F149 nimmt sich `noForth V1400-Fu0` mit Vocabularies als äußerst bequem aus. `V1400` bedeutet *Version mit Vokabular 2014* und `Fu0` heißt *F149 unter Verwendung von UART0*. Es gibt auch eine Version mit UART1 (`Fu1`). Ich finde vor allem das kleine F149–Miniplatinchen außerordentlich charmant. Es ist kaum größer als das LaunchPad und bringt 60 kByte Flash, 2 kByte RAM und 6 I/O-Ports mit. Das alles für 8,90€! Überdies noch 4 LEDs, Beeper, 2 Drucktasten und Reset. Alle I/O-Einrichtungen am Außenrand.

Der Bootloader auf dieser kleinen Platine arbeitet problemlos — und das ist wirklich eine feine Sache: Zwei Jumper umstecken und man hat RS232 über USB...

— Herzliche Forth–Grüße, Willem Ouwerkerk.

Übersetzung Fred Behringer, mk

MicroFlo auf Arduino

Unter der Überschrift *"A bit of visual programming with MicroFlo"* ist mir ein Hinweis zu einem graphischen Programmierwerkzeug für Mikrokontroller über den Weg gelaufen [1]. Also flugs auf die offizielle Seite [2] geschaut und die Installationsanleitung gesucht [3]. Ich kann berichten, dass das funktioniert, wenn man dem arduino die MicroFlo–Umgebung lädt, und wenn man genügend NodeJs–Module nachinstalliert. Dann kann man sich via Webbrowser mit MicroFlo verbinden. Programmieren heißt dann: Kästchen malen und den Ausgang vom einen mit dem Eingang von einem anderen Kästchen verbinden. Dann generiert man den Code und lädt ihn auf den Controller (in der Handhabung der ArduinoIDE sehr ähnlich). Faszinierend!

[1] `http://lwn.net/Articles/575524/`
[2] `http://github.com/jonnor/microflo/`

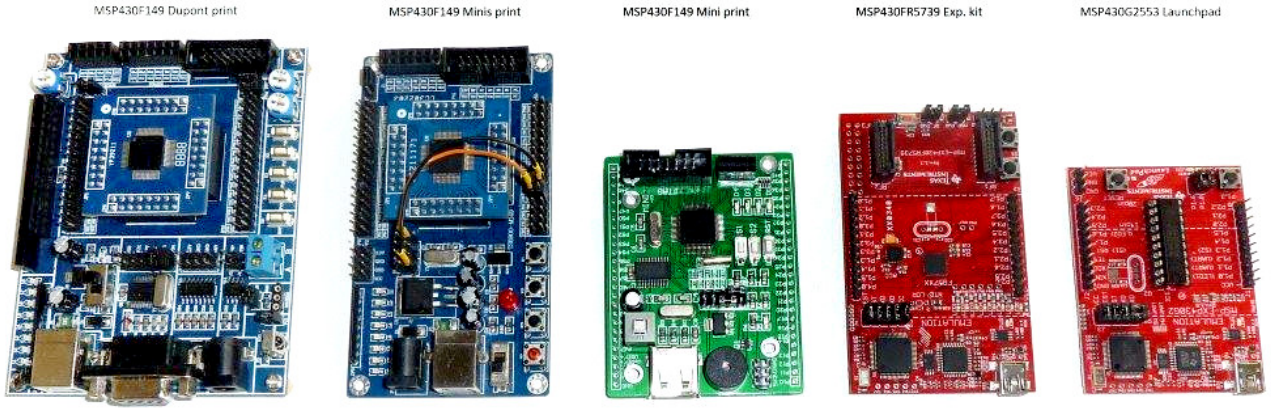


Abbildung 1: Fünf mal MSP 430 mit noForth

[3] http://github.com/jonnor/microflo/blob/master/doc/arduino-getstarted.md

ew

AmForth 5.2 erschienen

Kurz vor Weihnachten ist die Version 5.2 von AmForth [1] erschienen. Darin neu:

- überarbeitete Implementierung der Wortlisten, so dass in den Namen 8Bit–Zeichen verwendet werden können, z.B. Σ
- eine Implementierung von TRAVERSE-WORDLIST
- überarbeitete Implementierung von VALUE, siehe auch S.11 in diesem Heft.

[1] http://amforth.sourceforge.net/ *ew*

TCP/IP für AmForth

Jens Haselbauer hat in mühevoller Kleinarbeit einen simplen TCP/IP Stack für AmForth geschrieben und den Code dem Projekt zur Verfügung gestellt. Noch funktioniert das nur mit einem Atmega–644–Kontroller mit einem ENC28J60 Ethernet–Kontroller, aber es funktioniert. Diese Kombination gibt es zum Beispiel auf dem AVR–NetIO Board von Pollin oder auf dem etherrape Board von lochraster.org. Der Quelltext (komplett in Forth) und ein Rezept finden sich auf der Projektseite. Verbesserungen sind gerne gesehen.

http://www.lochraster.org/etherrape/
http://www.pollin.de Bestell–Nummer 810 058
http://amforth.sourceforge.net/TG/recipes/Telnet.html

ew

Aufgelesen

Chuck Moore: Programming a 144-computer Chip to Minimize Power
http://www.infoq.com/presentations/power-144-chip/

Armbanduhr mit MSP430, ob dort auch ein Forth laufen kann?
http://www.itopen.it/2013/11/28/the-hackable-watch-a-wearable-msp430-mcu

kleiner Linux IP Server im Ethernet-Stecker, mit IPv6:
http://linuxgizmos.com/rj45-sized-linux-networking-server-goes-ipv6

cas

Die FG auf dem 30c3

Bernd Paysan

> *Die Forth–Gesellschaft hatte eine „Assembly“ auf dem 30c3; über den Versuch, Hackern und Hacksen das Fortheln nahezubringen, soll hier berichtet werden.*

Der Chaos–Computer–Club ist nicht nur genauso alt wie die Forth–Gesellschaft, sondern hat sogar einen gemeinsamen Gründer: Klaus Schleisiek. Der war dann aufgrund des ganzen Trubels um Snowden sogar in der Tagesschau zu sehen[1], und hat sein wichtigstes Anliegen an eine breite Öffentlichkeit gebracht: Nämlich, dass wir ein anonymes, verschlüsseltes Internet brauchen.

Das war auch der Primärgrund, warum ich dort war, weil ich mit net2o an so einem Ding arbeite, und mich sowohl darüber informieren wollte, was es da an konkurrierenden Projekten gibt, und damit die auch erfahren, was ich so mache — die #youbroketheinternet–Session, die die Wau–Holland–Stiftung organisiert hat, war da das richtige Umfeld. Und wenn ich schon da bin, und weil sich mit Gerald Wodni, Gido Baumann, Martin Bitter und Carsten Strotmann auch noch ein paar weitere Mitstreiter gefunden haben, haben wir gleich eine „Assembly“ gemacht. Auf Deutsch ist das ein Tisch, um den herum sich ein paar Hacker mit Laptops setzen, und gemeinsam hacken, und anderen Hackern erklären, was sie da machen. Auf unserem Tisch stand natürlich der Triceps als Blickfang (Bild siehe Seite 13).

Leider hat Air Berlin einen kleinen Teil unserer Infrastruktur zerdeppert, nämlich ein paar Fischertechnikteile für den Tablet–Stand (da muss etwas robusteres her). Damit konnten die Hacker kein Go spielen; dieses Spiel wäre in der Umgebung durchaus machbar gewesen. Auch der Solitaire spielende Triceps war natürlich ein Publikumsmagnet. Gerald Wodni hat noch ein zweites, extra für den 30c3 begonnenes Projekt vorgestellt: Ein Namensschild. Das besteht aus einem 4€4th–Board und einem LED–beleuchteten LCD und 3 Buttons als Aufsatz. Zwischen beiden ist noch eine Batterie für die Stromversorgung. Gerald hat auf dem 30c3 das Programm erweitert, und am Schluss konnte man sogar ein kleines Spiel auf dem Namensschild spielen. Die Sourcen gibt's in einem Fossil–Repository, das Bild auf Seite 31.[2]

Der CCC ist anstrengender als der LinuxTag, es geht zwar erst ungefähr um 12 los, aber dann auch bis 3 Uhr nachts, und die letzten Gäste sind immer die hartnäckigsten. Mit über 9000 Besuchern ist das eine riesige Veranstaltung, die auch das Kongress–Zentrum in Hamburg sprengt, trotz großer Vortragssäle. Das Publikum ist natürlich nerdig, und zeigt reges Interesse.

Fortsetzung auf Seite 13

[1] http://www.tagesschau.de/multimedia/video/video1361860.html
[2] http://fossil.wodni.at/r.cgi/nametag

30 Jahre Forth–Gesellschaft e.V.

Erich Wälde et al.

Die erste Ausgabe der Vierten Dimension, die Ausgabe 0, wurde im Mai 1984 veröffentlicht. Nach meinen Informationen wurde die Forth–Gesellschaft e.V. *am 28.11.1984 auf dem Dachsberg–Treffen in Kamp Lintford als Verein gegründet. Dreißig Jahre sind seither vergangen und fordern ihren Tribut an Luftschlangen, Konfetti, Böllern an die Swap–Drachen und ihre Hüter. Nun bin ich selbst erst seit 2006 dabei, also quasi noch ein Jungspund. Deswegen kommen hier hauptsächlich andere Stimmen zu Wort.*

Ein Bericht zur Gründung von Horst–Günther Lynsche, die Vorstellung des ersten Direktoriums (mit Fotos) und die erste Satzung lassen sich in der Ausgabe 1985/02 der VD im Archiv [1] nachlesen. Die Eintragung ins Vereinsregister erfolgte am 22.11.1984.

Auf der vereinsinternen Mail–Liste hatte ich nachgefragt, was es denn zu erzählen gäbe. Mehrere Personen haben sich tatsächlich hingesetzt und geantwortet. Vielen Dank für die Zeilen. Und an die Leser: es gibt noch drei Hefte im Jubiläumsjahr zu füllen, also ist noch ausreichend Platz für Anekdoten, interessante Dinge oder Begebenheiten, gerne mit Referenzen. Stöbern Sie im Heftarchiv — oder im Heftestapel zu Hause — um Ihre Erinnerungen aufzufrischen. Die Redaktion freut sich über Zuschriften [2].

Auf die nächsten dreißig Jahre!

I

Erich Wälde

1984 war ich noch nicht mal Student, und grad mal mit einem Taschenrechner ausgerüstet — nein, kein HP. Stacks waren mir zu der Zeit unbekannt. Zwar hatte ich schon einen Lötkolben — das Löten hatte mir Peter beigebracht, der ältere Bruder von einem Mitschüler — aber Ahnung hatte ich keine. Die Ahnungslosigkeit hielt mindestens bis zur Hinsch'schen Elektronikvorlesung im Wintersemester 1989/90 an. Zwar bestand ich die Prüfung in Elektronik als angewandter Physik, aber die Ahnungslosigkeit war immer noch groß.

Erst im Jahr 2000 holte ich meinen Lötkolben wieder raus, damals motiviert durch den Modellbau — eine Segelwinde brauchte ganz dringend eine Wegbegrenzung. Die Schaltung funktionierte selbstredend nicht. Aber diesmal fand ich heraus, warum nicht, und konnte sie zum Leben erwecken. Dann hatte die Vorlesung die Ahnungslosigkeit doch reduziert — immerhin.

Etwa in dieser Zeit machte ich meine ersten Versuche mit PIC–Microkontrollern (P16F84), mit selbstgelötetem Programmer. Durch ein gutes Buch wechselte ich zum 8051–Kontroller, programmierte weiter in Assembler und manchmal auch in C.

Auf irgendeinem Linuxtag in Karlsruhe fragte ich Ewald Rieger ein Loch in den Bauch. Ich glaube, da hörte ich zum ersten Mal von Forth. Ich erstand eine gebrauchte Ausgabe von Leo Brodie (Starting Forth) und verstand erst mal auch nicht viel. Auf dem Linuxtag in Wiesbaden sah ich dann aber den Renesas–R8C–Kontroller mit gforth–ec — so einen hatte ich noch herumliegen. Viele Abende pflügte ich mich durch Neuland. Irgendwann hatte ich dann auch einen `ok`–Prompt — eigenartig, wie manche Menschen ob dieser zwei Buchstaben in Verzückung geraten können `:-)`

Dann hab ich das aufgeschrieben und an `comp.lang.forth.de` geschickt. Und Bernd Paysan wollte das für die VD haben. Und schon war ich drin, in dieser verrückten, anderen, drachengeschwängerten Welt. Rolf Schöne legte mir eine Mitgliedschaft in der FG sehr nahe und beherbergte mich auch netterweise, als ich einmal meine neugierige Nase auf dem Stammtisch in München zeigte.

Der Umstieg auf die Atmel–AVR–Prozessoren war dem geringen Datenspeicher des R8C geschuldet. Mit AmForth fand ich ein sehr komfortables System. Immerhin hab ich ein auf mehrere Platinen verteiltes Mess–System im Haus installiert, welches in AmForth programmiert ist und seit Jahren klaglos seinen Dienst tut. Der Rest ist Geschichte und kann in der VD nachgelesen werden.

Stacks habe ich übrigens erst mit `emacs/calc` verstanden.

II

Rolf Schöne

Wie bist Du zu Forth gekommen? Im Mai 1978 erfuhr ich in "Dr. Dobbs Journal of Computer Calisthenics & Orthodontia", dass es Forth gibt. Nun frage mich nicht, wie ich an ein papierenes Listing der FIG für den 6502 gekommen bin, den ich mir baute und für den ich nicht so etwas wie ein Betriebssystem hatte. Internet gab es noch nicht, gab es Gopher schon? Ich weiß es nicht mehr. Jedenfalls lief nach einer Adaption auf Code–Ebene (Assembler gab es noch keinen) irgendwann nach Interface–Basteleien und dem Anschluss eines ausrangierten Fernsehers endlich ein `OK` am Bildschirm auf. Hurra!

Wie und wann bist Du zur Forth–Gesellschaft gekommen? Fred Behringer warb mich. Schon komisch: Beide arbeiteten wir am gleichen Institut, und doch dauerte es zwei Jahre, bis wir gleichartige Interessen feststellten. Das "wann" findet man sicher noch im Archiv.

Was hast Du (damals und heute) mit Forth gemacht? Auf welchen Gerätschaften? Damals mit dem 6502 einen Vortrag über PL/65 und seine Vorzüge (entgegen meiner Überzeugung, nur weil er schon vorbereitet war). Dann zusammen mit meinem Kollegen ein Disk–Interface mit einem Transputer–Baustein (hieß der M3?).

Irgendwelche Anekdoten zu erzählen? Nur das: Die FG besitzt seit langem einen LEGO NXT-Bausatz. Ein Forth dafür gibt es immer noch nicht. Für den Nachfolger auch nicht. Und für den EV3 erst recht nicht. Schade. Was meint Bernd dazu? Egal. Vagisses. Mein Enkel kommt auch ohne Forth zurecht. Besser wäre allerdings mit Forth, nachdem sein fast dementer Opa keines mehr schreiben kann.

III

Jens–Hanno Schwalm

Ich bin als Student über einer der ersten Nummern der c't (1983) und dann über den Zech an Forth gekommen, habe dann erste Versuche auf meinem ZX81 gemacht. Auf einer Forth–Tagung habe ich dann Klaus Schleisiek (Aachen) kennen gelernt, der brachte mich auf Forthmacs auf dem Atari–ST.

Mein erstes größeres Projekt war dann die Portierung von Forthmacs auf meinen Acorn–Archimedes, das ARM–Forthmacs–System wurde dann die Grundlage für den ARM–Firmworks–Port ... Seitdem interessiere ich mich persönlich hauptsächlich für *threading*–Anwendungen und *realtime–audio*–Anwendungen.

Seit mehr als 10 Jahren benutze ich ausschließlich iforth auf Linux Rechnern und bin auch seit 10 Jahren an der iforth–Entwicklung beteiligt. (xlib–Treiber, threading, audio ...)

Für die 4D habe ich, glaube ich, nur einmal einen Artikel geschrieben: über JACK–Audio und Forth. Ich pflege das Audiosystem fjack für vfx und iforth. Es gibt auch diverse konkrete Anwendungen — z.B. in einem Klangexperiment in einem Museum.

IV

Peter Glasmacher

Ich bin 1990 auf ein geniales FORTH–Derivat von Professor Mayer–Lindenberg gestoßen. Es nannte sich FIFTH und war ein Host/Target-System zur Entwicklung von Mikrocontrollern — in unserem Fall Maxim DS5000 (Typ 8051) — und benötigte kein aufwendiges Entwicklungssystem, sondern nur einen PC und eine RS232–Schnittstelle. FIFTH hatte viele sehr sinnvolle Verbesserungen gegenüber FORTH und entsprach deshalb nicht genau den FORTH–Konventionen.

Inzwischen verwende ich SwiftX–MSP430 von FORTH Inc. und bin auch damit sehr zufrieden.

V

Michael Kalus

Die Zeit fliegt grad nur so dahin. Forth ist mal wieder Schuld daran. Sohn Daniel wollte wissen, wie das geht mit so einem PID–Regler, wenn man damit ein Motörchen in eine bestimmbare Drehzahl versetzen will, die bei Last gehalten wird. Teufel auch, gar nicht so einfach, das mit einem einfachen Modell eines billigen Gleichstrom–Bürstenmotors zu machen. Da lernt man staunen, wie elegant so billige Lüftermotörchen das heute können. Wahre Wunderwerke der Technik sind das, kann ich da nur sagen.

Du willst wissen, wie ich zu Forth gekommen bin? Da hat mich meine Neugier hin getrieben, und das ist auch heute noch der einzige Grund dafür. Jetzt, wo es diese fantastischen kleinen MCUs gibt, kann man damit viele elektronische Experimente machen. Das ist viel besser als zu den AIM65 Zeiten! Denn damit hab ich angefangen. Damals trieb mich die Frage um, wie es sein kann, dass so ein Haufen Draht mit mir reden kann, und tut, was man ihm sagt. Das war die Zeit als in den Krankenhäusern — ich war frisch gebackener Assistenzarzt — die Mikroelektronik Einzug hielt. Plötzlich kamen da bildgebende Geräte, die mit Ultraschall funktionierten. Und komfortable EKGs, Blutanalysegeräte im Labor, und all so ein Zeugs. Da wollte ich einfach wissen, wie das geht. Forth auf dem AIM65 brachte die Einsicht in die Bits und Bytes, und das tuts für mich heute noch.

Zur Forth–Gesellschaft kam ich damals auf der Suche nach mehr Infos über Forth. Das Rockwell–RSC–Forth–Buch war eine große Hilfe, aber dann hörte man von neuen Standards, FPC und so was. Und es sollte da eine Zeitung geben in den USA, "Forth Dimensions", die habe ich bestellt. Und darin gelesen, dass es *local chapters* gab überall auf der Welt. Das fand ich hilfreich und gut, und so versuchte ich, auch so eins zu gründen, in Wuppertal damals. In Hamburg gab's das schon, und so traf ich auf Lynsche und Schleisiek, und das erste deutschsprachige Heft der Forthzeitung, die "Vierte Dimension". Und just damals stand an, den Forth–Verein in der BRD zu gründen. Da war ich natürlich gleich Feuer und Flamme und dabei. Internet gabs noch nicht, Post und Telefon, eine Typenrad-Schreibmaschine am PC, Papier, Kleber und Schere, das waren die Mittel, die Zeitung zu machen. Und ehe ich mich versah, war ich selbst schon der nächste Editor der Hefte. 40 Seiten DIN-A4, von den Autoren getippte Seiten, kopiert, zerschnippelt aufgeklebt auf die Vorlagen–Seiten, lagen da im Flur der Reihe ausgelegt. Layout war da wörtlich gemeint. Mit den fertigen Seiten gings dann zum Copy–Center, jede einzeln für alle Mitglieder kopiert, sortieren und heften konnten die Kopier–Maschinen noch nicht. Zuhause dann wurde die Frau mit eingespannt, Heft zusammen legen, immer rund um den Tisch, die Blätter von den Stapeln ziehen, bis ein Heft zusammen war, klammern, fertig — und das nächste. Eine abendfüllende Beschäftigung. Und mit dem Kurier–Zug der Bundesbahn ging dann der ganze Karton nach Hamburg zum Versand. Eintüten und frankieren war da sicher auch nicht das beliebteste. Der Teil ist übrigens bis heute geblieben, denke ich. Das Heft selbst entsteht ja inzwischen vollständig im PC, in LaTeX.

Richtig *gemacht* mit Forth habe ich eigentlich nichts. Mein Beruf war ja ein anderer. Das Projekt, das mir die meiste Einsicht brachte zu meiner grundlegenden Frage an den Haufen Draht, war der "Disforthler". Das RSC–Forth konnte damit komplett disassembliert werden. So verstand ich jeden Schritt, den so eine CPU macht — die 6502 damals — um mit mir zu reden. Ein langer Weg, aber die Dinger waren ja sehr schnell, 1 MHz damals! Bei dem Tempo sieht das dann schon so aus wie Zauberei, wenn ein Tastendruck auf dem Display erscheint, und ein Wort ausgeführt werden kann. Die Magie war enthüllt. Aber so ganz ist der Zauber nicht verschwunden, ich staune darüber heute noch. Dem Disforthler folgte lange nur Spielerei hier und da. Mit Forth in Windows Tiefen vorzudringen, ist mir verschlossen geblieben. Da gab's lange kein passendes Einstiegswerkzeug. Win32forth konnte mir die GUI nicht wirklich erhellen. Aber nun mit der Ära der MCUs, erst Atmels ATmega und nun Texas Intruments MSP430, erlaubten mir wieder Einsichten in jene Welt dank Forth. Das AmForth von Matthias Trute, und das Camelforth von Brad Rodriguez, brachten mir viele kurzweilige Stunden bei der Erforschung der Fähigkeiten solcher MCUs. Das eigene Forth ist dabei nicht herausgekommen, aber ein Clone des Camelforth, immerhin, gut angepasst auf das MSP–Launchpad von TI. Und so, wie man das RSC–Forth komplett verstehen und beherrschen konnte, so ist es mit den kleinen Forths auf den MCUs heute auch wieder. Das macht Spaß.

Was für mich noch ansteht, ist eigentlich so ein Forthprozessor im FPGA. Einige davon gibt es ja schon. Das mal mit einfachen Mitteln nachvollziehen zu können, wäre ein lang gehegter Wunsch. Was für die Rente denke ich, dann habe ich wieder viel Zeit.

VI

Jörg Völker

Wie bist Du zu Forth gekommen? Den ersten Kontakt hatte ich in der Homecomputer Ära, mit meinem "Dragon 32" `:-)` Das war sehr frustrierend, denn der Lernzyklus war: Eintippen, Absturz, neu von Kassette laden... Ich fand es aber immerhin so spannend, dass später ein experimenteller Mini–Compiler in Basic (!) entstand, aus lauter DATA-Zeilen.

Wie und wann bist Du zur Forth–Gesellschaft gekommen? Daran ist Klaus Schleisiek schuld. Ende 2003 kam völlig überraschend eine E–Mail, Wolf Wejgaard (Holon11) hatte meine Mail–Adresse weitergegeben, wohl so nach dem Motto *da in Wedel gibs noch einen, kennt Ihr den eigentlich?* Klaus lud mich daraufhin zu einem Küstenforth–Treffen in Hamburg ein. Wenig später bin ich dann Mitglied geworden. Und gleich im Frühjahr 2004 war ich dann erstmals auf Fehmarn dabei mit einem LKW-Modell...

Was hast Du (damals und heute) mit Forth gemacht? Auf welchen Gerätschaften? Anfangs war das alles nur Spielerei. Beruflich wurde es erst 1998 mit SwiftX 68k von Forth Inc. für eine Industrie–Steuerung und das war ein verdammt harter Kampf. Danach kam der Modellbau in den Fokus und Holon11 für den 68HC11, dann die ersten komplett selbstgestrickten Systeme, erst für den Freescale HC08, dann Freescale S08 und nun noch Freescale Coldfire V1. Aktuell sind wir hier in der tematik GmbH jetzt vier Entwicklungsingenieure, die "Forth sprechen". Davon arbeiten zwei tag–täglich in der Softwareentwicklung in Forth — die hab ich sozusagen selber ausgebildet `:-)` Unser Geld verdienen wir mit Modellbauelektronik, also Fahrtreglern, Lichtanlagen, Soundmodulen und Modellfunk Sendern- und Empfängern. Seit 2001 haben wir gut 39.000 Baugruppen verkauft, in ca. 80% der Module tickt ein Forth–Interpreter, der Rest ist direkt in Assembler programmiert. Wir haben Forth also so gesehen ganz schön verbreitet, leider merken unsere Kunden davon rein gar nichts, denn es sind ja "embedded systems".

VII

Martin Bitter

In meiner Kindheit gab es noch gar keine Programmierer, besser gesagt: in meinem Umfeld kannte man so etwas nicht. Als Jugendlicher waren für mich Programmierer noch Menschen, die vor einem riesigen Steckbrett saßen und Kabel mit Krokodilsteckern in irgendwelche Löcher steckten. Damals waren viele Rechner noch Analogrechner. "Digital" war ein ganz neues Wort, bei dessen Erwähnung noch immer gesagt wurde, dass es von "an den Fingern abzählen" komme. Später, als jungem Erwachsenen (meine Frau zweifelt immer noch an, ob ich überhaupt erwachsen bin), begegnete mir dann das Wort "Personal Computer", die Abkürzung PC kam erst viel später auf.

Von einem meiner ersten Gehälter in Festanstellung kaufte ich mir dann so ein Ding. Ich rationalisierte dies mit der Begründung, Arbeitsblätter zu gestalten und meine stets fehlerbehafteten Texte einfacher korrigieren zu können. Meine Examensarbeiten hatte ich auf geliehenen IBM–Kugelkopfschreibmaschinen (mit Korrekturband!) getippt. (Ganze Absätze waren später rückwärts auf dem Korrekturband zu lesen.)

Es gab damals eine Vielfalt an Rechnersystemen. Ich hatte keine Ahnung, aber der Mann einer Kollegin war Fernmeldetechniker und hatte sich den WDR–Kleincomputer gebaut, also hatte der "Ahnung". Er empfahl mir wegen des Preis–Leistungs–Verhältnisses einen Atari–ST. Einen solchen erstand ich dann auch — gebraucht, mit defekter Diskette. Das Betriebssystem war im ROM. Das Gerät war sofort nach dem Einschalten an. Schneller als der Röhrenmonitor hell war, aber ich hatte kein einziges Programm. Ein freundlicher Verkäufer bei Karstadt (die waren damals tatsächlich kompetent, was die neuen "PCs" anging) kopierte mir die zwei Originaldisketten neu und dann ging's los.

Ich konnte jetzt Texte für die Schule und für Seminar schreiben. Dazu hatte ich mir ein "Interface" (auch so ein neues Wort) für meine Triumph–Adler Typenradschreibmaschine mit Korrekturband besorgt. Im Vergleich zu den damaligen Nadeldruckern war das Schriftbild super und obendrein kam niemand auf den Verdacht, dass ich so ein "modernes Teufelsteil" hatte. Aber ich musste

einen “Druckertreiber” für das “Interface” schreiben, damit auch alle Zeichen der Schreibmaschine verfügbar waren. Genaugenommen mehrere Treiber. Denn ich hatte ja nicht nur Typenräder mit unterschiedlichen Schriftstilen, sondern auch welche mit unterschiedlichen Zeichensätzen. “Druckerteiber” hört sich komplizierter an, als es tatsächlich war. Im Grunde genommen waren das Asciitabellen der Art `45 - 46 +` usw. “Ascii” war auch wieder so ein neues Wort (genau genommen eine Abkürzung).

Internet gab es nicht. Informationen klaubte man sich aus Data–Becker–Büchern zusammen oder las Zeitschriften. 68’er–Magazin (nö nicht politisch, der Atari hatte einen 68000 Prozessor (wieder ein neues Wort).) Ein anderes war das ST–Magazin. Auf einer der Atari–Disketten war auch ein “BASIC” — *beginners all purpose simple instruction code* (Hab’ gerade mal nachgegoogelt: es heißt korrekt nicht *simple* sondern *symbolic*.) “Code” war ebenfalls ein neues Wort. Ich habe anfangs immer “Kot” verstanden und war leicht irritiert ;-)

Mit BASIC gab es erste Programmierversuche. Ein bildschirmfüllendes 640×400 Apfelmännchenbild war über Nacht fertig berechnet! Ein Mensch hätte (gefühlte) Jahrzehnte dafür gebraucht. Arbeitsblätter mit Grafik entstanden. Aber alles sehr mühsam. Dann las ich in einem der Magazine eine klitze–kleine Anzeige: Eine PD (public domain, wieder ein neues Wort) Programmiersprache namens VolksForth. Ich ließ mir per Post die drei Disketten kommen und bewunderte das Demoprogramm. Das war ein Diskettenkopierprogramm. So etwas war damals nicht unbedingt Bestandteil eines Betriebssystems. Es gab unzählige davon. Dieses hier konnte sogar mit doppelseitigen und einseitigen Disketten umgehen. Ich war überzeugt.

Dann lieh ich mir einen Nadeldrucker, druckte alle Quellfiles aus und band sie zu einem hübschen Buch. Das habe ich heute noch. Im Begleitbrief zu den drei Disketten wurden mir die Bücher von Brodie empfohlen mit dem Rat: Reißen Sie die Kapitel über den Editor raus. Weiterhin wurde die Forth–Gesellschaft empfohlen. So fing es an. Vielleicht finde ich ja noch den Brief von Herrn Pennemann, indem er mich über Abstürze, unbeabsichtigte Endlosschleifen und Ähnliches hinwegtröstet und mir mehr Spaß mit Forth verspricht. Da hat er mal so etwas von Recht gehabt!

VIII

Ulrich Hoffmann

Forth? Ich hab’ Sommer 1984 im Buchladen das Buch *Die Programmiersprache FORTH* von Ronald Zech gesehen und fand spannend, was da erklärt wurde. Für meinen ZX–81 hatte ich auch ein Forth, irgendwie. Buch gelesen, viel gerätselt und dann hab ich mich daran gemacht, das ZX–81–Forth zu dekompilieren, um mehr zu verstehen. War dann schnell klar, dass es eine Art FIG–Forth war. Ich habe sogar den ZX–81 aufgebohrt und um ein Character–RAM erweitert, damit ich statt Klötzchen ein richtiges `@` darstellen konnte `:-)`

Mein nächster Computer war ein ITT–3030 CP/M-Computer. Forth hatte ich dafür nicht.

Im Herbst 1984 habe ich dann Kontakt zur Forth–Gruppe in Hamburg bekommen und war auf einem der ersten Treffen im *Common Interface Alpha* im Schanzenviertel mit H.G. Lynsche, Klaus Schleisiek und Andreas Goppoldt, Georg Rehfeld und auch Bernd Pennemann.

Von H.G. habe ich F83 für CP/M bekommen, von Klaus viel Literatur und die Idee, volksForth unter CP/M ans Laufen zu bringen (Damals gab’s ultraForth auf dem C64 von Georg und Bernd und erste Versuche von volksForth auf dem Atari-ST von Dietrich Weineck). Ich hab volksForth mit dem laufenden F83 gebootstrapt und hatte so nach ein paar Wochen mein eigenes metacompilierbares Forth unter CP/M. Es fühlt sich sehr gut an, ein System zu haben, das vollständig im Quellcode vorliegt und sich selbst (re)generieren kann. Ich habe seinerzeit viel gelernt, insbesondere auch, wie man systematisch Fehlerursachen findet. Vielen geht das ab. Ich bin sehr froh, dass ich immer wieder Neues durch Forth lernen darf.

Referenzen

1. Heftarchiv der VD `http://www.forth-ev.de/filemgmt/viewcat.php?cid=2`
2. Redaktion der VD `mailto://vd@forth-ev.de`
3. Leo Brodie — Starting Forth `http://home.iae.nl/users/mhx/sf.html` oder `http://www.exemark.com/FORTH.htm`
4. AmForth `http://amforth.sourceforge.net`
5. gopher `http://de.wikipedia.org/wiki/Gopher`
6. 6502 `http://de.wikipedia.org/wiki/MOS_Technology_6502`
7. ZX81 `http://de.wikipedia.org/wiki/Sinclair_ZX81`
8. Ronald Zech — Die Programmiersprache FORTH, Franzis Verlag, ISBN-10: 3772372635
9. Forthmacs `http://www.forthfreak.net/index.cgi?Forthmacs`
10. Acorn Archimedes `http://de.wikipedia.org/wiki/Acorn_Archimedes`
11. iforth homepage `http://home.iae.nl/users/mhx/`
12. swiftX–msp430 `http://www.forth.com/embedded/swiftx-embedded-systems-14.html`
13. msp430 im FG Wiki `http://www.forth-ev.de/wiki/doku.php/projects:4e4th:start`
14. camelForth `http://www.camelforth.com`
15. Volksforth: `http://volksforth.sourceforge.net`
16. Atari Club ABBUC `http://www.abbuc.de`

Value–Manufaktur in AmForth

Matthias Trute

Values sind ein alter und wohlabgehangener Teil von Forth. Was gibt es da schon Neues zu entdecken? Braucht man die überhaupt noch?

Status quo

Values sind so alt, dass sich wohl nur die ganz alten Hasen entsinnen, warum die überhaupt existieren. Alle anderen suchen vielleicht eine Zeitlang nach deren Sinn und ziehen sich auf *"Schon immer da gewesen, wird schon zu was Nutze sein"* zurück [1].

Im Sprachkern gibt es einige Worte wie HERE, PAD, SOURCE und TIB, die man als Value implementieren *kann*. Nach der Lektüre dieses Artikels könnte man auf die Idee kommen, TIME&DATE ebenso umzusetzen.

In der Tat ist die Konkurrenz mit Variablen nah. Beiden ist gemein, dass sie einem (kleinen) Speicherbereich einen Namen geben und dieser Speicherbereich dann nicht anderweitig benutzt wird. Der alles entscheidende Unterschied liegt darin, dass eine Variable die Adresse ihres zugewiesenen Speichers offenlegt, wohingegen ein Value diese Information nicht preisgibt.

Den Wert einer Variablen zu lesen, erfordert so zwei Schritte: Zuerst die Adresse besorgen und dann den Speicher lesen. Ebenso wird in eine Variable geschrieben: Adresse besorgen und dann reinschreiben. Die Adresse eines Values ist dagegen wenig erhellend. Der Standard sagt nichts darüber, was man da bekommt.

Den Wert des Values erhält man durch Aufrufen des Names. Das sollte schneller gehen als bei den Variablen und erfordert weniger Code. Um den Wert eines Values zu ändern, ist TO das Mittel der Wahl. Das ist ein parsing word, das zudem noch für ganz andere Sachen eingesetzt wird (Stichwort Locals).

Auf den ersten Blick ist so ein Value erst mal anders zu handhaben und das Fehlen einer nutzbaren Speicheradresse, die man nach Belieben ver- und bearbeiten kann, kann ein Nachteil sein. Interessant wird es, wenn man die ausgetretenen Pfade des einheitlichen Adressraums verlässt, wo es Adressen gibt, die für @ und ! unerreichbar sind.

Hinzu kommt ein weiterer Aspekt: Values sind nicht auf einen einzelligen Speicherbereich beschränkt. Vielmehr gibt es 2VALUE[1] und FVALUE, die ein mehr an Daten speichern. Das TO kommt damit problemlos zurecht, solange genug Bytes auf dem Datenstack liegen.

Das vielleicht unwichtigste Merkmal ist, dass ein Value immer die *richtige* Menge an Daten transferiert. Eine 2Variable muss man immer mit einem 2@ lesen. Blöd, wenn man da einen Fehler macht.

Value-Manufaktur

Die Eigenschaft, dass es unbekannt ist, *wo* die Daten gespeichert werden, kann inspirieren. Anwendungsfälle für den PC fallen mir keine ein (Forth für die Cloud muss noch erfunden werden), für Mikrocontroller hingegen eröffnen sich interessante Perspektiven.

In AmForth wird ein Value grundsätzlich im EEPROM abgelegt. Das ist eine sehr alte Entscheidung, die sich auch bewährt hat. Damit wird dem Umstand Rechnung getragen, dass das Lesen eines Values einfach, das Schreiben umständlich(er) ist, resp. sein soll. Ebenso ist EEPROM: Oft lesen, da einfach, aber nur selten schreiben. Nichtsdestotrotz gab und gibt es den Wunsch, Values auch im RAM abzulegen.

Daneben kennt AmForth Values, die völlig neben allen Standards liegen: Byte-Values, die nur ein Byte speichern können (und darauf aufpassen, wirklich nur ein Byte zu schreiben) oder eines, welches seine Daten im EEPROM ablegt, aber durch den Einsatz eines RAM–Caches trotzdem hohe Änderungsraten verkraftet[2].

Für alle Value–Typen existiert eine generische Laufzeitumgebung, die unabhängig vom konkreten Typ ist. Das senkt u.a. auch die Komplexität beim TO ganz erheblich. Damit man eigene Valuetypen definieren kann, ist nur sehr wenig an Systemwissen erforderlich. Dieses Wissen kann sich vollständig in einem kurzen erzeugenden Wort befinden. Die damit definierten Values funktionieren wie alle anderen auch.

Dieses Systemwissen ist auch der Grund, warum es (noch) keine Value–Fabrik gibt, sondern (nur) eine Manufaktur. Es ist nicht durchautomatisiert und der Nutzer muss sich drauf einlassen.

Wie gehts

Am Beispiel des Byte-Values soll der Einsatz der Manufaktur beschrieben werden.

Zunächst werden zwei Hilfsworte benötigt: Eines zum Lesen und eines zum Schreiben des eigentlichen Datums. Diese beiden Worte werden in das vordefinierte Schema für Values eingebaut, damit sie zur Laufzeit aufgerufen werden können:

[1] Ubuntus gforth 0.7 kennt die nicht, Forth2012 schon.

[2] EEPROMs sind zwar häufig neu beschreibbar, erreichen aber trotzdem irgendwann ihre Grenzen. RAM hingegen ist da nicht eingeschränkt.

```
: c@v ( f-addr -- c ) @i c@ ;
: c!v ( c f-addr -- ) @i c! ;
: cvalue ( n "name" -- )
     (value)           \ create a new wordlist
                       \   entry with value runtime
     here 1 allot      \ allocate RAM and...
     dup , c!          \ keep the address and
                       \   initialize it
     ['] c@v ,         \ method for read operation
     ['] c!v ,         \ method for write (TO)
                       \   operation
;
```

Das interne Wissen besteht darin, dass Values in AmForth drei Pointer im Body des Dictionaryeintrags umfassen: Eine Adresse gefolgt von zwei Execution Tokens für die Lese– bzw. die Schreiboperation. Die Worte hinter den beiden Execution Tokens erhalten zur Laufzeit die Adresse des ersten Felds als Parameter. Die Laufzeitumgebung aller Values greift auf diese drei Pointer in gleicher Weise zurück.

Sprachexperten werden sofort erkannt haben, dass das eine OO–Architektur ist, die nicht zu Ende geführt wird. Sie haben Recht. Dafür ist sie einfach, sowohl in der Programmierung als auch in der Nutzung. Ob daraus mal eine "richtige" OO–Umgebung entsteht, wird sich zeigen.

Andere Systeme implementieren VALUE und TO auf eine völlig andere Art und haben vermutlich noch nie den Bedarf gehabt, überhaupt neue Valuetypen zu definieren. Zudem ist das TO als state smart parsing word ohnehin nicht sehr wohlgelitten.

Alternative DOES> ?

DOES> hat den Charme, dass man den Runtimeeffekt selbst definieren kann. Für ein Value ist das jedoch zu wenig. Da nicht nur *eine* Runtimeaktion als vielmehr deren zwei implementiert werden müssen. Eine standardisierte Möglichkeit, mehrere Methoden an ein Objekt zu binden, bietet erst die Objektwelt, die bei Forth bislang ein Randthema darstellt.

Syntaktisches

Für das eingangs erwähnte Caching-Value sind zwei weitere Methoden unabdingbar. Zum einen muss der Inhalt des Caches vor der ersten Nutzung auf den aktuellen Stand gebracht werden. Das nennt man Aufwärmen. Und zum anderen ist es sinnvoll, den Cache zumindest gelegentlich zurückzuschreiben, damit die Änderungen dauerhaft gespeichert sind.

Hier entsteht ein kleiner Bruch bei der Syntax. Die zusätzlichen Methoden müssen irgendwie auf die Interna der Values zugreifen können. Das geht klassisch via ' (tick):

```
' value-name warm-cache
....
  content TO value-name
.....
' value-name flush-cache
```

Da ist die eingangs erwähnte Seltsamkeit, dass die Adresse eines Values keine sonderlich interessante Information per se darstellt. Ebenso ist bei der Nutzung innerhalb von COLON–Worten zu beachten, dass man dann ['] nutzen muss. Da erscheint es naheliegend, die Idee des TO auszubauen:

```
         warm-cache  value-name
...
         content  TO value-name
...
         flush-cache value-name
```

Damit wird die explizite Adressermittlung vermieden und eine Verwechslung mit Variablen kann nicht mehr stattfinden. Mindestens ebenso wichtig ist, dass damit auch keine stilistischen Brüche mehr vorhanden sind. Zu beachten ist, dass die Methoden zwar gleich heißen, aber nicht die gleichen sind. Im letzteren Fall sind es state smart parsing words, die, wie eingangs erwähnt, nicht wohlgelitten sind[3].

Ausblick

Zuallererst ist das alles eine spleenige Gedankenspielerei für gelangweilte Systemprogrammierer. Nebenbei ist es eine Spielwiese für weitere neue und ungewohnte Sprachkonstrukte wie Quotations, die noch spleeniger sind:

```
: cvalue ( n "name" -- )
     (value)           \ create a new wordlist entry
                       \   with value runtime
     here 1 allot      \ allocate RAM and...
     dup , c!          \ keep the address and
                       \   initialize it
     [: @i c@ ;] ,     \ method for the read operation
     [: @i c! ;] ,     \ method for the write (TO)
                       \   operation
;
```

Noch weiter gehen Kombinationen mit noch mehr neuen Sachen:

```
08:08:91:92:91:01 to eth::mac
192.168.1.2/24 to eth::ip
1.1.2014 00:02:12 to TIME&DATE
```

Die Zahlen werden natürlich von einem Recognizer [rec11] im Interpreter erkannt und passend gemacht. Die eth:* Values sind natürlich direkt im PHY-Modul untergebracht, welches extern via SPI am Microcontroller hängt. Die Zeitinformation geht natürlich direkt in das RTC–Modul (Real Time Clock) des Rechners, z.B. via I2c. Eine passende VALUE–Writemethode macht es möglich. Die Zeit lesen geht genauso direkt aus der RTC, oder einem regelmäßig fortgeschriebenen Cache desselben.

[3] Warum das so ist, wäre interessant zu erfahren.

Referenzen

1. TO – Ein Mechanismus mit vielen Möglichkeiten. VD 2002/03
2. Recognizer — Interpreter dynamisch verändern. VD 2011/02

Listings

Die Caching–Variante des Standard-Values. Hier wird sicher deutlich, dass das alles eigentlich OO zu Fuß ist. Die ersten drei Felder im Value– Dictionaryeintrag sind vorgegeben, hinzu kommt ein viertes Feld, das die RAM–Adresse des Caches enthält. Die Lese– und Schreibmethoden greifen nur auf den Cache zu. Einzig die warmup– und die cache–flush–Worte arbeiten mit beiden Speichern.

```
\ 2 is a magic number
: @cache 2 + @i @ ;
: !cache 2 + @i ! ;

: flush-cache 1+ dup 2 + @i @ swap @i !e ;
: warm-cache  1+ dup @i @e swap 2 + @i ! ;

: cache-value
    (value)
    dup edp dup , dup cell+ to edp !e
    ['] @cache  ,
    ['] !cache  ,
    here 2 ( 1 cell ) allot dup , !
;
```

AmForth ist ein 16Bit–Forth. RAM und EEPROM sind dabei byteorientiert, das Dictionary benutzt hingegen 16 bit pro Adressunit, das mag verwirren.

. . . Fortsetzung der Meldung zum 30c3 von Seite 6

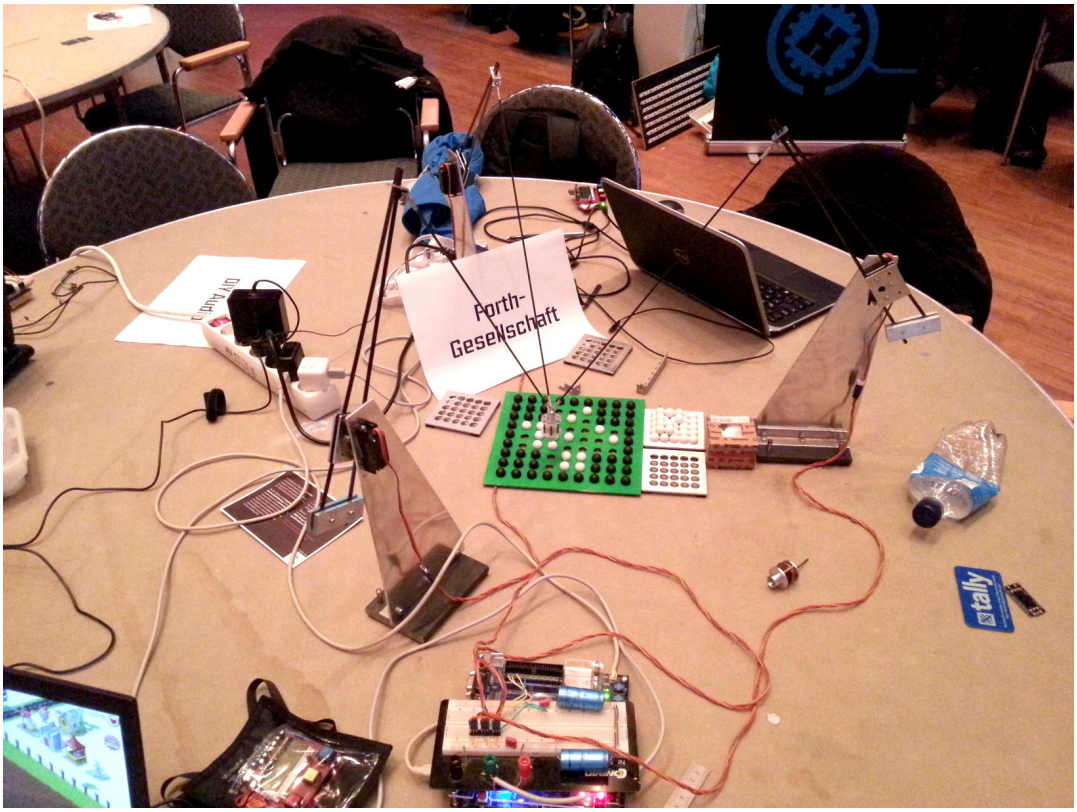


Abbildung 1: Die Forth–Gesellschaft auf dem 30c3

Fortsetzung auf Seite 31

Forward–Deklarationen

Matthias Trute

Es gibt Fälle, bei denen muss man ein Wort benutzen, bevor es definiert werden konnte. Ein Standardfall für `DEFER`*, scheint es.*

Hintergrund

Auf der AmForth–Mailingliste wird gelegentlich ganz Grundsätzliches diskutiert. So kam neulich das Thema auf, dass die bestehenden Defer–Varianten zu langsam seien und es doch irgendwie schneller gehen müsste. Nach einigem Hin und Her gab es die Möglichkeit, einen Defer zu versiegeln, so dass er nicht mehr verändert werden kann, dafür aber schneller ist. Das war schon gut, aber wenn man Nutzern einmal einen kleinen Finger reicht, wollen sie noch mehr, ergo war die Lösung nicht gut genug.

Es sollte eine Forward–Deklaration geben, die sich automatisch auflöst, sobald alle erforderlichen Definitionen vorhanden sind und die keinen zusätzlichen Runtime–Overhead mehr hat.

Umsetzung

Defers basieren darauf, dass sie ein Execution Token (XT) speichern, welches sie zur Ausführung bringen können. Dieses XT kann per `IS` geändert werden. Genau diese Indirektion ist es, was es zu ändern galt. Bei AmForth sind damit 5 Forth-Worte involviert (2 mal Fetch, 2 mal Execute plus ein zusätzlicher Exit am Ende). Andere Systeme mögen mit weniger auskommen, dafür ist der AmForth-Defer sehr flexibel, was den Speicherort angeht. Darüber hinaus gibt es starke Überschneidungen zu Values, was die Programmgröße reduziert.

Eine Forward–Deklaration umzusetzen, die den Anspruch erhebt, schneller als Defers zu sein, darf eines nicht haben: eine Indirektion. Das heißt im Umkehrschluss, dass der Programmcode zur Laufzeit verändert werden muss. Solcher self–modifying Code ist spätestens in der Steinzeit der IT als schlecht und untauglich verworfen worden. Die damit verbundenen Gefahren sind groß. Ebenso reagiert die Architektur moderner Prozessoren sehr unwirsch mit einem unglaublichen Performanceverlust, wenn man den Inhalt der Codesegmente einfach mal so ändert. Von Sicherheitsüberlegungen wie Virenangriffen und dergleichen mehr mal ganz abgesehen.

Nun besteht die Welt nicht nur aus den dicken Hobeln, die Welt der Mikrocontroller ist (noch) erheblich einfacher gestrickt. Da ist self–modifiying Code zwar auch nicht so toll, aber noch in Reichweite.

Wie bringt man den Wunsch nach einem schnellen Defer auf den Boden der Realität? Der erste Anlauf sah folgendermaßen aus (Vorschlag aus der Mailingliste)

```
variable back&forth
\ use forward referenced word
: forth&  dp back&forth ! -1 , ; immediate
\ use forward referenced xt
: &forth
  postpone (literal) -1 , dp 1- back&forth !
;  immediate
\ forward reference resolver
\ (put anywhere in referenced word)
: &back& latest @ back&forth @ !i ; immediate

\ 1st calling 2nd examples:
\
\ : first1  ." first1 " forth& ;
\ : second1  ." second1" &back&  ;
\
\ : first2  ." first2 " &forth execute  ;
\ : second2  ." second2" &back&  ;
```

Das funktioniert, nur hat es keiner verstanden. Abgesehen von den ganzen technischen Herausforderungen muss man neue Technik auch so verpacken, dass sie benutzbar wird. Stephen Pelc hat den richtungsweisenden Slogan *Notation matters* geprägt. Mit dieser Strategie im Hinterkopf wird eine Notation gesucht, die allen eventuellen Gurucode verstecken kann und trotzdem les– und wartbar ist.

```
> forward: foo
> : bar foo ;
> bar
Unresolved forward foo
> : foo ." hi" ;
> bar
hi
>
```

Das sieht schon verständlicher aus. Bleibt die Kleinigkeit, es auch genauso zu realisieren. Wie kommen die beiden `foo`'s zusammen und was passiert mit dem `foo` in `bar`? Zu dem Zeitpunkt, an dem `bar` definiert wurde, ist die endgültige Definition von `foo` (also dessen XT) noch unbekannt. Es muss ein Mechanismus entstehen, der alle forward–Referenzen findet und durch den endgültigen Code ersetzt.

Wie findet man alle Referenzen? Allen seit der Deklaration erzeugten Code durchzugehen, dürfte scheitern. Zahlen sieht man nicht an, ob sie ausführbarer Code sind. Es

[1] Mit dem Begriff JIT ist auch der self–modifying Code wieder opportun. Ein JIT–Codeveränderer macht nichts anderes. Dabei ist es unerheblich, ob die Codeveränderung nur zur Laufzeit im RAM geschieht oder bis in die Programmdatei hinein reicht.

gibt jedoch eine Stelle, wo diese Information mit Sicherheit verfügbar ist: In der CPU. Sobald die Programmausführung von `bar` bei `foo` angekommen ist, ist es Zeit, den Code Just–In–Time[1] zu verändern. Ein Nebeneffekt ist, dass man nicht vergisst, den Codeveränderer zu starten, was bei externer Lösung mindestens die Fehlersuche erschwert.

Das von `forward:` angelegte Wort hat damit eine komplexe Laufzeitaktion. Es muss zunächst den eigenen Namen im Dictionary suchen. Wenn der gefunden wurde, gibt es zwei Möglichkeiten. Es handelt sich entweder um die Forward– Deklaration selbst oder um eine andere Definition. Im letzteren Fall kann man unterstellen, dass die endgültige Definition erfolgt ist und ein neuer XT zur Verfügung steht. Dieser neue XT wird genutzt, um den bisherigen XT damit auszutauschen. Darüber hinaus wird er natürlich auch ausgeführt. Wenn `bar` noch einmal ausgeführt wird, wird sofort der neue XT ausgeführt, von dem alten ist keine Spur mehr vorhanden.

Wenn der gefundene XT der Gleiche wie der von der Forward–Deklaration ist, bedeutet dies, dass es noch keine Neudefinition gibt, eine Fehlermeldung *Unresolved forward declaration* weist den Programmierer darauf hin. Wenn der Name gar nicht gefunden wurde, hat jemand an der search-order geschraubt. Der Effekt ist der gleiche, nur die Fehlermeldung kann eine andere sein.

Variationen

Der oben skizzierte JIT Compiler mit self–modifying Code ist etwas extrem. Wenn man dem JIT Compiler so auslegt, dass er den neuen XT zwar erkennt und ausführt, es aber unterlässt, ihn auch in den Code einzubauen, wird er bei jedem Aufruf von `foo` aktiviert. Das bewirkt eine erneute Dictionary–Suche, die möglicherweise zu einem anderen Ergebnis führt (`foo` wurde wieder mal neu definiert). Das ist natürlich nicht mehr mit Vorteilen bezüglich der Laufzeit verbunden. Hier ist die analoge Defer–Lösung mit einem passenden `IS` eindeutig vorzuziehen. Andererseits kann man die Wortliste, in der gesucht werden soll, so gestalten, dass sie kurz genug ist. Damit hat man ein neues Werkzeug geschaffen, das auszuprobieren lohnen kann.

```
> forward: foo
> : bar foo ;
> : foo ."hi" ;
> bar
 hi
> : foo ."ho" ;
> bar
 ho
>
```

Die `forward:` Deklaration wird damit zum *late binding*. Zusammen mit ein wenig Wordlist–Management kommen dann Aktionen zustande, die kontextsensitiv sind. Die Komplexität kann grenzenlos sein ...

Implementierung

Die nachfolgende Forward–Deklaration ist für AmForth entstanden.

```
: forward:
  \ keep data to make live easier.
  dp create ,
  does>
  \ get the XT of the current word
  dup 1- swap
  \ copy the current name to RAM for
  \ find-name
  @i here iplace here count
  \ lookup the dictionary. get the XT
  find-name if
    tuck =
    abort" found only forward declaration."
    \ replace the XT in the caller and
    \ execute it *afterwards*
    dup r@ 1- !i execute
  else
    \ can only happen if search wordlist has
    \ changed
    true
    abort" unresolved forward declaration"
  then
;
```

Die Lösung ist nicht allzu sehr gegen Fehler gesichert. So wird nicht abgefangen, dass das erzeugte Wort nur im Rahmen eines anderen, später definierten Wortes ausgeführt wird. Wird das erzeugte Wort direkt aufgerufen, wird der find–recognizer im Interpreter zerstört. Ebenso ist die Fehlermeldung nicht ganz die Gleiche, wie im Konzept skizziert. Alles Übungen für den geneigten Koch. Rezepte sind keine Befehle, sie sklavisch zu befolgen, ist nicht erwünscht.

Referenzen

AmForth Kochbuch via `http://amforth.sourceforge.net/`

Stephen Pelc “Es kommt auf die Notation an.” VD 3/2012

Buchbesprechung: Retro–Computing: Simulation — Emulation — Projekte, “Exotic Flavor”

Ulrich Hoffmann

Retro-Computing ist ein auch für Forth interessanter Zweig der Computertechnik, da Forth auf vielen alten Computern zur Verfügung steht und so auch immer wieder Erwähnung findet. Das rege Interesse, auf das Forth auch auf dem alljährlich stattfindenden Vintage Computer Festival [2] trifft, belegt das eindrucksvoll. In seinem Buch Retro–Computing: Simulation — Emulation — Projekte, “Exotic Flavor”*[1] geht Peter Sieg auf verschiedenste Projekte ein, die auf unterschiedlichste Weise auch unbekannte alte Computer wieder zum Leben erwecken. Aber auch Modernes hat sich in sein Buch verirrt, wie zum Beispiel 4€–Forth auf dem Launchpad* `:-)`

Retro–Computing

Auf 116 DIN–A4–Seiten beschreibt Peter Sieg die verschiedensten Retro–Computing–Projekte, die sich mit einer Unzahl an alten Computern beschäftigen. Darunter finden sich bekannte Computer, die man anheben kann (Macintosh auf seiner Vorstellung durch Steve Jobs in 1984: *Never trust a computer you can't lift* [3]), wie Apple Lisa, 68K Macintosh Computer, Atari ST oder Sinclair ZX-81 aber auch bekannte schwere Modelle wie PDP-10 oder Cray-1. Vor allem gilt Peter Siegs Augenmerk aber weniger bekannten Systemen, wie dem Xerox Alto, MITS Altair, Kenbak-1, zahlreichen Computern der DDR (K1000er Reihe, EAW P8000, Robotron 1715, etc.), Kosmos Logikus oder der Vektrex-Konsole, um eine kleine Auswahl zu nennen. C64, Schneider CPC oder Apple-2 spart er bewusst aus, da Emulatoren für diese an anderer Stelle ausgiebig beschrieben sind.

Simulation — Emulation — Projekte

Jedes der Projekte stellt Peter Sieg mit farbigen Fotos, Screenshots, Listings oder Schaltplänen vor und vermittelt so einen guten Eindruck, was das Projekt leistet. Wichtig natürlich auch die Links hin zur Webseite des jeweiligen Projekts, wo man dann auch die Software bekommt. So fällt es nicht schwer, den Einstieg in die Projekte zu finden. Dabei werden neben den Software–Emulator–Platzhirschen Mess [4] und Mame [5] eben insbesondere auch zahlreiche weniger bekannte Spezial–Emulatoren vorgestellt wie SimH, Salto, JKCEmu und viele andere.

Außer reinen Software–Emulatoren geht Peter Sieg aber auch auf Hardware–Nachbauten ein, wie etwa den Kenbak-1–Emulator auf AVR–Basis, CP/M auf dem ATmega88 oder auch einen ZX-81 im AVR. Das zugehörige AX81–Projekt [6] ist auch in der Lage, einen Jupiter Ace zu emulieren. Leider spricht Peter Sieg weder diesen Sachverhalt an noch einen der zahlreichen Jupiter Ace Software-Emulatoren [7].

Ein weiterer Teil widmet sich der Hardware–Emulation von Systemen in FPGAs. Hier finden sich Neurealisierungen vom DDR–Z1013–Computersystem, vom Minimig (Mini Amiga 500) und von 6809–Microcomputern, insbesondere auch Rekonstrukt [8], einem 6809–Micro mit Mais–Forth, über das wir ja auch schon in Heft 2009–02 der Vierten Dimension berichtet haben (5 Jahre alt — na gut, lassen wir als Retro durchgehen `:-)`. Außerdem taucht hier auch das TI–Launchpad mit 4€–Forth auf, das nun eigentlich wirklich ganz und gar nicht retro ist, sondern ein ganz modernes Forth, mit dem man prima in die Controller–Programmierung einsteigen kann. Auf vielen der vorgestellten Rechnern läuft Forth: volksForth unter CP/M oder so wie bigForth auch auf dem Atari ST, JForth auf dem Amiga... Hier mehr zusammenzutragen, wäre wünschenswert gewesen.

Fazit

Vielleicht fehlt der Emulator Emulith [9] des Modula–2–Computers Lilith oder auch Lisp–Maschinen–Emulatoren, aber ansonsten bietet dieses Buch eine recht umfangreiche und vollständige Sammlung. Einen Anspruch auf Vollständigkeit erhebt es auch gar nicht. Gedruckt wurde das vorliegende Exemplar bei Amazon in Leipzig und ist zu einem Preis von 17.07 € erhältlich. Zu diesem Preis kann man auch über die teilweise unscharfen Fotos hinwegsehen.

Fazit: Für Retro-Interessierte eine gute Zusammenstellung interessanter Projekte. Die Auswahl der vorgestellten Projekte ist vielfältig, offenbart aber einige Lücken. Ein schöner Startpunkt für eigene Recherchen im Netz.

1. Peter Sieg: Retro–Computing: Simulation — Emulation — Projekte, “Exotic Flavor”, ISBN-13: 978-1494709983, Selfpublishing via Createspace.de, Januar 2014
2. `http://www.vcfe.org/D/`
3. `http://www.youtube.com/watch?v=2B-XwPjn9YY`
4. `http://www.mess.org`
5. `http://www.mamedev.org`
6. `http://www.jcwolfram.de/projekte/avr/ax81b/main.php`
7. `http://www.jupiter-ace.co.uk/`
8. `http://code.google.com/p/rekonstrukt/`
9. `http://pascal.hansotten.com/index.php?page=emulith`

Arduino Controlled Digital FM Radio

Craig A. Lindley

After reading about AmForth [2] on hack-a-day [5] I looked around for a project to use it with. I decided to build an Arduino controlled Digital FM radio with LCD display to gain some experience with AmForth and to bring back my Forth chops that I hadn't used in a very long time. The result is an FM radio that is fully remote controllable that I listen to with headphones while I am working. This project uses relatively simple hardware that can be built by anyone with basic electronic assembly experience. The software however is somewhat complex and took me some time to get working. More on the software a little later.

If you add a cool feature to the radio, please send me a note at: `calhjh@gmail.com` *and let me know so I can incorporate it into my radio as well.*

Hardware

The hardware is built around an Arduino Uno board running at 16 MHz and 5 volts. Other Arduino's could be used but some of the I/O pin assignment would need to change. Using an Arduino running at a different speed will also impact the design especially in the IR (Infra Red) detection area. I use an IR remote control from Adafruit [6] (see Figure 1) to control the radio. There are no other controls at all besides a display contrast trimmer adjustment for the LCD which should be set once and left alone. A schematic of the hardware is shown in Figure 4. I built my radio using point to point wiring but a prototyping shield could be used for a cleaner build.

I packaged my radio using two 4" x 6" pieces of clear 1/8" acrylic in a sandwich arrangement held together with wooden 1 1/2" dowel spacers. I like the look of the naked electronics. The LCD display is mounted to the front acrylic piece and the Arduino Uno and receiver board are mounted to the rear piece. See the photos for details.

The LCD display provides a 4-bit parallel data interface to the Arduino while the Si-4703 FM receiver is connected to the Arduino via an i2c serial interface. Since the Arduino Uno is a 5 volt part and the FM receiver is a 3.3 volt part, a bi-directional level converter must be used between them. The LCD display runs on 5 volts. The backlight for the LCD display is controlled by an output pin from the Arduino.

Figure 1: IR Mini Remote Control

The stereo audio output cable is used as the antenna for the FM receiver so you must pay attention to the length of the interconnect. The cord on the headphones I use works fine as an antenna as I can pick up all of the FM stations in my area. When strong stations are tuned in, the audio quality is top notch. When the receiver is receiving a stereo broadcast, the stereo indicator on my build lights up. Weaker stations are received in mono and the stereo indicator remains dark. A red power on LED lights when the radio is on.

An IR receiver (which is what Radio Shack calls it) is used to detect IR codes from the remote control. It filters out the 38 KHz IR carrier frequency thereby making detection of the key codes easier (though not easy).

The radio is powered via a USB cable and USB power supply. Alternatively the radio can be powered by connection to a USB port on your computer.

Software

Development Environment

I developed all of the AmForth software on my MacBook Pro using amforth-shell.py described in the AmForth documentation. Using the shell makes software development fast and convenient. All of the AmForth source files I use have a marker placed at the beginning of the file of the form

```
marker _markerName_
```

That way I can easily unload/forget faulty code before I load a newer version. I installed amforth-shell.py as an alias called `4thterm` via the following entry in my `.profile` file (the alias is one line without the trailing backslashes!).

```
AMFORTH=~/Documents/dev/AmForth
AMFORTH_LIB=$AMFORTH
export AMFORTH_LIB
alias 4thterm="$AMFORTH/amforth-5.1/tools/amforth-\
shell.py -p /dev/tty.usbmodem1411 --no-error-onout\
put"
```

Then I can bring up amforth-shell by typing `4thterm` in any shell window. Quite convenient.

Software Components

Although the AmForth software spans many files, there are really three blocks of functionality that need to be discussed.

IR Detection

IR detection was by far the trickiest part of the software to get running. The code is contained in the file `irDetect.frt`. The software is tricky because of the real time nature of IR transmissions. Here, the Forth code polls the output of the IR receiver looking for transitions called `MARK`s and `SPACE`s. I won't go into the details here but the codes broadcast by a IR remote control form a serial protocol with the lengths of the MARKs determining where the codes start and which bits are to be considered ones and zeros. This code doesn't try to actually decode the codes being sent, instead it creates an identifier via hashing that uniquely identifies the keys on the remote. Consult the code for the details. If you want to use a different remote with your radio, changes to this code will be necessary.

LCD Display

As mentioned a 4 bit parallel interface is used between the 16 character, 2 line LCD display and the Arduino. The software must first put the LCD display controller into 4 bit mode by executing a series of instructions which it does in its `initLCD` method. Numerous Forth words exists for controlling the display, positioning the cursor and writing text to the display. See the file `lcd_16x2.frt` for details. The backlight of the display is controlled via an output pin on the Arduino and can be switched on and off without affecting the display content.

Si-4703 Receiver Control

The Si-4703 FM receiver is controlled using a series of 16-bit registers. In a typical operation the registers are read from the chip, values are changed and the updated register values are sent back causing the receiver chip to react. The Si-4703 chip has some idiosyncrasies that made code development interesting. For example all registers must be read each time and the Si-4703 provides the register content in the following order: `reg 10 .. 15` followed by `reg 0 .. 9`. The code for controlling the Si-4703 is contained in the file `si4703.frt`.

There are Forth words available for setting the volume, setting the channel, reading the channel, seeking upwards or downwards, etc. The code I developed just scratches the surface of what the Si-4703 can do. I'm still working on code to read the RDS data provided by the receiver which identifies the radio station, the song being played and even the current date and time. In the future I hope to be able to display this information on the LCD display.

The file `ui.frt` brings all of the software modules together and forms the user interface for the radio.

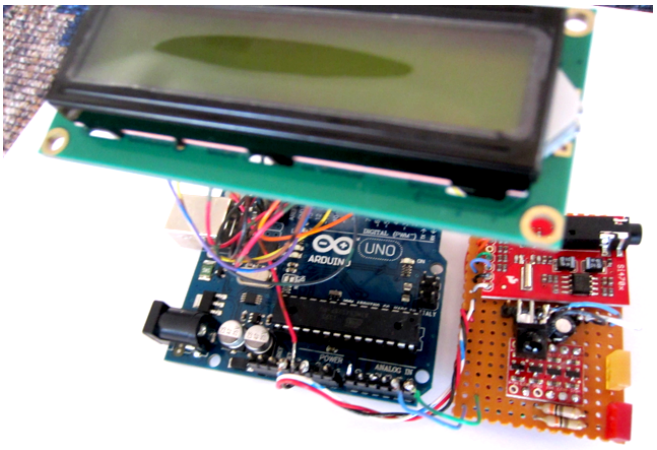


Figure 2: Three Components — LCD, Arduino Uno and Receiver Board

Installation

Starting with the stock Arduino Uno installation I use amforth-shell.py to load the required Forth files. I call the file who's content is shown below `loadall.frt`. If you `#include` this file after a new AmForth install it will load up all the Forth files in the correct order.

```
\ Load up all pre-requisite forth files
#include postpone.frt
#include marker.frt

marker _EMPTY_

#include case.frt
#include buffer.frt
#include bitnames.frt
#include twi.frt
#include si4703.frt
#include irDetect.frt
#include lcd_16x2.frt
#include ui.frt
```

If all is well and everything loaded/compiled successfully, executing the top level word `runFMRadio` should start the radio. NOTE: the radio will appear off until you click the play/pause button on the remote. At this time the display will light up and the radio becomes functional. Clicking play/pause again makes the radio appear off though it is still listening for the IR code to turn it back on. Figure Three details the remote control key assignments.

I want to run this app as a turnkey application but at the present time this is not working for me.

Editor: this has been solved using a 500 ms delay at start in order to allow all hardware/firmware to settle. (from email on amforth mailing list)

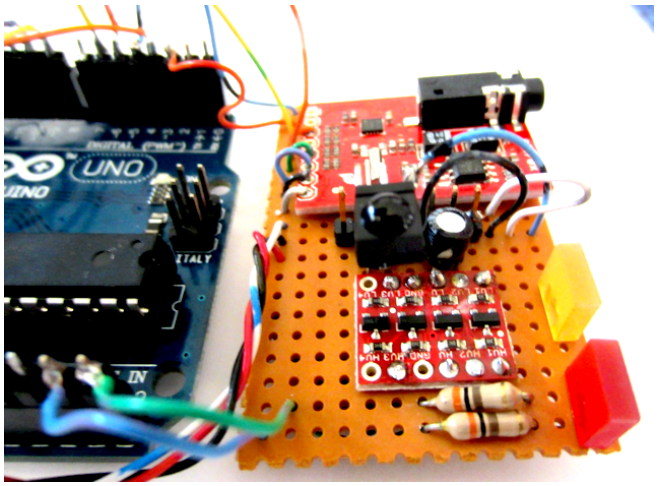


Figure 3: Receiver Board Close Up — At the top is the evaluation board with the black 1/8" stereo output jack, in the middle is the IR receiver facing upward, towards the bottom is the four channel level converter, the yellow rectangular LED is the stereo indicator and the red LED is the power on indicator.

Resources

Information about the Si-4703 digital FM receiver can be found at the manufactures website [7]. AN230, AN231, AN243 are application notes concerning the Si-470x series parts. AN332 provides example code useful for programmers.

The original version of this article and auxiliary files can be found at the authors website [1]. The original announcement dated 2013-12-05 can be found at amforths mailing list archive [3,4].

The complete code of this project can be found at [1].

Links

1. `http://craigandheather.net/misc/Lindley_ArduinoFMRadio_AmForth1.0.zip`
2. `http://amforth.sourceforge.net`
3. `http://amforth.sourceforge.net/faq.html`
4. `http://sourceforge.net/mailarchive/message.php?msg_id=31720020`
5. `http://hackaday.com`
6. `http://www.adafruit.com`
7. `http://www.silabs.com`

Schematic and Tables

Item	Part Number	Source
Arduino Uno	16Mhz 5 volt part	
Bi-directional Level Converter	BOB-12009	SparkFun
Evaluation Board for Si4703 FM Tuner	WRL-10663	SparkFun
16x2 line LCD display		
IR Receiver	#2760640	Radio Shack
Red LED		
Yellow LED		
2 x 300 ohm 1/4 w resistor		
10K ohm 10 turn trimmer		
Mini Remote Control	ID: 389	AdaFruit
USB cable for Arduino to USB power supply connection		
USB power supply 500 mA or greater		
.1" male break away header pins for Arduino connectors		

Table 1:

Key	Function
Vol-	Mutes the audio
Play Pause	Turns the radio off and on
Vol+	Un-mutes the audio
Up Arrow	Volume Up
Down Arrow	Volume Down
Left Arrow	Scan Down for a station/channel
Right Arrow	Scan Up for a station/channel
Enter Save	Sets up for saving a preset. Tune a station, press Enter Save and then a key 1 .. 9 to save a preset.
Keys 1 .. 9	Tunes station if preset set, otherwise does nothing

Table 2: Key mapping

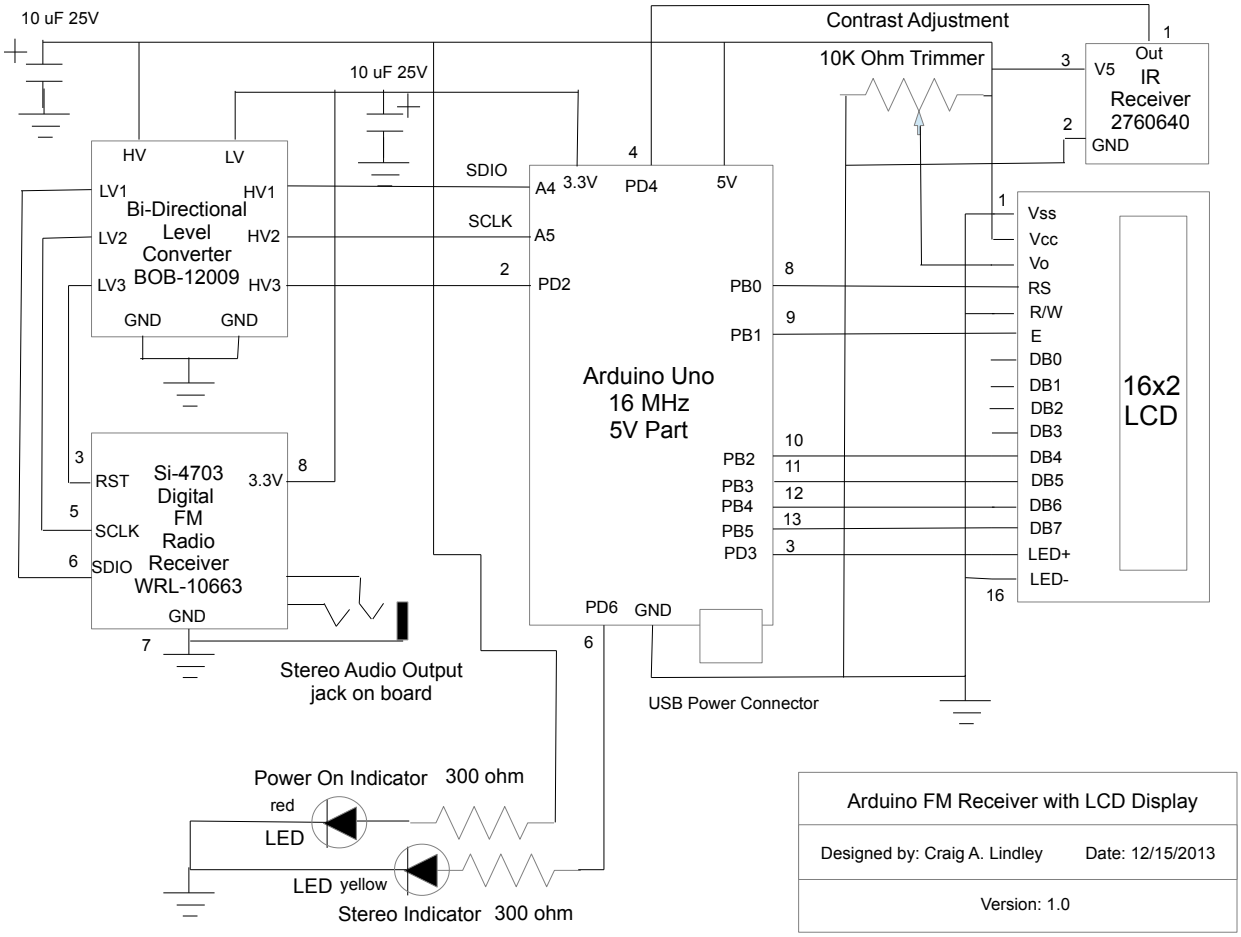


Figure 4: The schematics ...

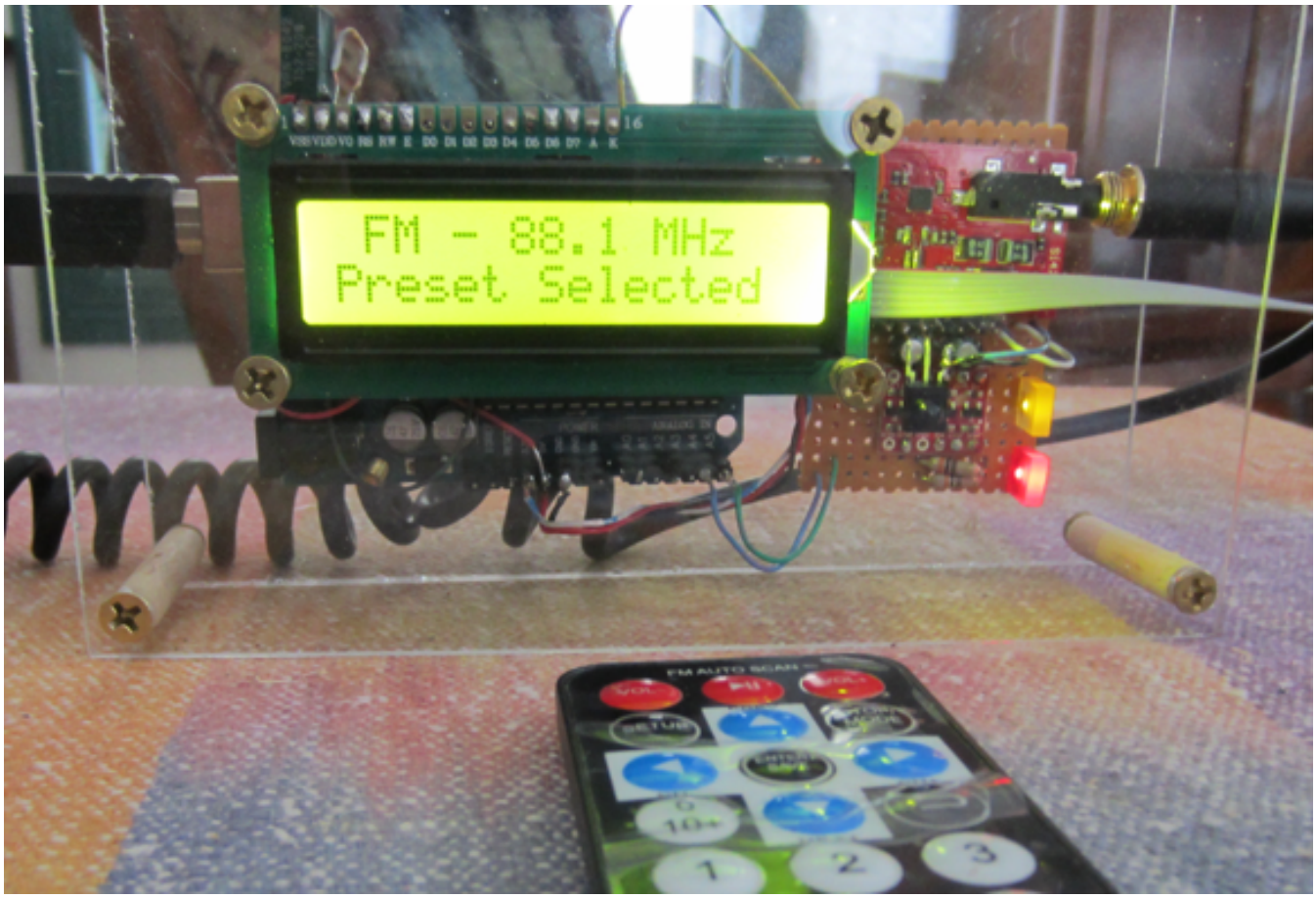


Figure 5: The Working Receiver in its prototype packaging — On the left is the USB cable powering the receiver. On the right I have my headphones plugged onto the receiver. The contrast adjusting trimmer can be seen on the left top of the LCD display.

Fortsetzung von Seite 6

naken_asm

Tool change for CamelForth/MSP430
Datum: 2. März 2014 17:40:24 MEZ

Gentlemen, ich dachte, ich sollte Euch wissen lassen, dass ich plane, die CamelForth/MSP430–Kernel–Entwicklung auf ein neues Werkzeug zu verlegen, `naken_asm`.

Anwender *bSquare* informierte mich über diesen Assembler. Der arbeitet auf allen Plattformen (Linux, Windows und Mac), und so kann ich komfortabel auf meinem Linux–Desktop entwickeln, anstatt den Windows–Laptop hervorholen zu müssen. Und er ist auch Multi–Target fähig, was wahrscheinlich für mich wichtiger ist als für Euch. `:-)`

Ich habe die größte Hürde erfolgreich genommen: Der MSP430–CamelForth–0.5–Kernel wurde bereits mit `naken_asm` generiert. Ich hoffe den aktuellen Quellcode bald herausbringen zu können — ich muss das F1611–Target noch konvertieren, und dann die Dokumentation für die neuen Werkzeuge überarbeiten.

Grüße, Brad

`http://www.mikekohn.net/micro/naken_asm.php`
`http://www.camelforth.com/news.php`

mk

DCF77–Funkuhr

Rafael Deliano

Nützlich, wenn man auf Datenlogger timestamps benötigt. Aber auch klassisches Anfängerprojekt. Benötigt nur einen Portpin und eine 1–Millisekunde–Verzögerungsschleife, Rest kann in FORTH programmiert werden.

Hardware

Typisch verwendet man Empfängermodule, die bereits ein digitales Signal mit Logikpegel liefern. Hier wurde eine alte Baugruppe von Auerswald verwendet, die für Außenmontage gedacht ist, und ein 5V–Logiksignal liefert (Abb. 1,2). Der Oldtimer ist intern diskret aufgebaut und heute noch sporadisch auf ebay erhältlich. Liefert sehr sauberes Signal (Abb. 3) und stellt damit minimale Anforderungen an Software.

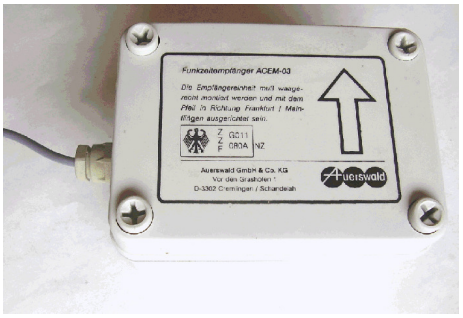


Abbildung 1: DCF–Baugruppe von Auerswald

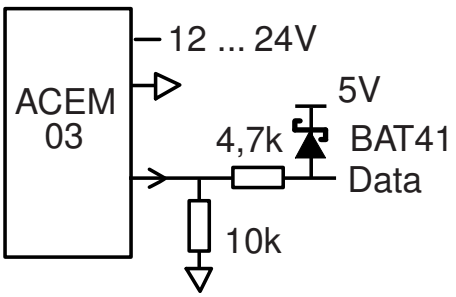


Abbildung 2: Schaltung Auerswald

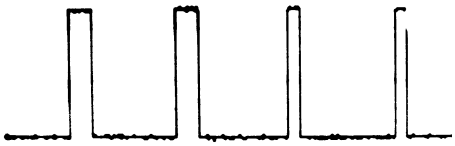


Abbildung 3: Signal Auerswald

Billiger ist das Modul von Pollin (Abb. 4,5), das für Controller mit 2.5 ... 3.3 V Versorgung passt und einen Enable–Eingang hat. Es eignet sich nicht direkt für das Programm im Listing. Man braucht zusätzliche Routinen gegen Prellen und Spikes (Abb. 6). Bisweilen ist auch ein LC–Filter in der 3 V–Versorgung nützlich. Sowie Kabel, um Abstand zum Computer und Nähe zum Fenster zu erhalten.

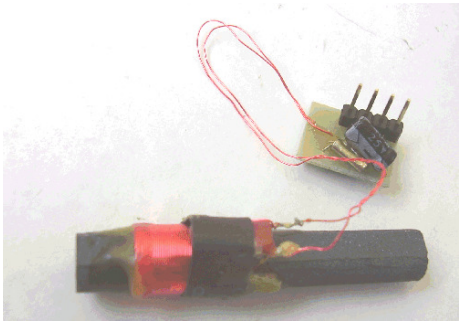


Abbildung 4: DCF–Modul von Pollin

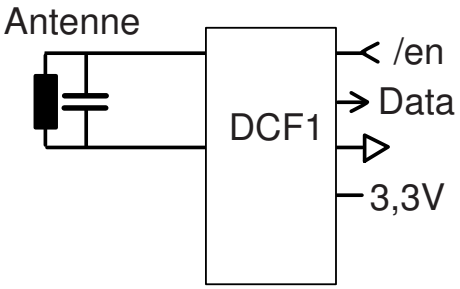


Abbildung 5: Schaltung Pollin

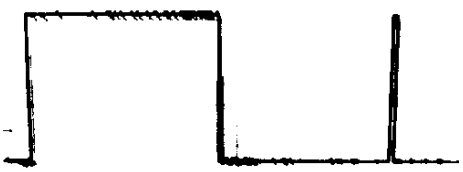


Abbildung 6: Signal Pollin

Datenformat

Die Trägerfrequenz von 77,5kHz wird pro Minute 59 mal im Sekundentakt auf 25% der Amplitude abgesenkt (Abb. 7). Ist die Dauer dieses Einbruchs 100 ms, signalisiert das ein Bit 0. 200 ms entspricht Bit 1. Allerdings werden diese Low–Pulse von schmal–bandigen Empfängern meist deutlich verschliffen, so dass die Dauer des Pulses etwas variiert. Bei Übergang von Sekunde 59 zu Sekunde 0 erfolgt keine Absenkung. Das ermöglicht dem Empfänger die Synchronisation. Man erhält somit pro Minute ein 59–Bit–Datenwort. Bit 0 ... 20 enthält entweder keine Information oder Daten, die selten benötigt werden. Die Nutzinformation steckt in Bit 21 ... 59. Die Ziffern sind in BCD kodiert. Bei den Wochentagen gilt: 1

= Montag. Die Prüfbits P1 — P3 ergänzen die vor ihnen liegenden Blöcke von Datenbits auf gerade Parität.[1]

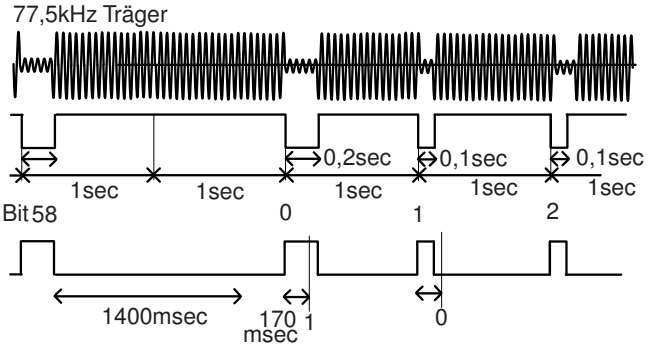


Abbildung 7: Timing

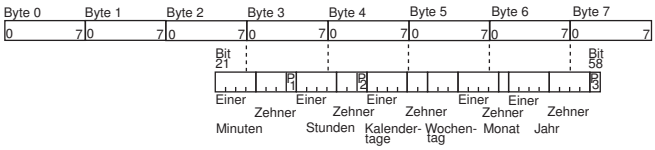


Abbildung 8: Datensatz

Programm

Erster Schritt ist Synchronisation, also Warten auf den fehlenden Puls. Was bis zu einer Minute dauern kann. Die Schaltschwelle 1400 ms (Abb. 7) ist ziemlich unkritisch. Dann folgt das Einsammeln der 59 Bits, die in 8 Bytes abgespeichert werden. Der Abtastzeitpunkt 170 ms (Abb. 7) dafür muss eventuell an das verwendete Empfängermodul angepasst werden. Bei prellender Hardware kann es sinnvoller sein die Pulsbreite zu messen.

Hat man das komplette Paket zusammen, prüft man anhand der Paritybits, ob Ausgabe sinnvoll ist. Bei erfolgreichem Empfang druckt das Programm die Rohdaten und Zeit am Terminal aus (leicht editiert, Ed.):

```
02 0D F4 5A 92 5E 4E 04 12:57: 0 Sonntag... 29.12.13
44 68 14 5B 92 5E 4E 04 12:58: 0 Sonntag... 29.12.13
C2 1B 34 4B 92 5E 4E 04 12:59: 0 Sonntag... 29.12.13
```

Mit 9600 Baud macht das Timing noch keine Probleme, da die Ausgabe mit dem recht unkritischen Synchronisatonspuls zusammenfällt. Auch hier ist Geduld nötig, es dauert eine Minute bis eine neue Zeile kommt.

Sekunden

Es werden hier keine Sekunden dargestellt, da diese ja nicht im Datenblock enthalten sind. Die 60. Sekunde ist auch nicht über den Start eines Bits aus dem Funksignal abzuleiten. Will man Uhrzeit im Sekundentakt auf LED-Display anzeigen, braucht man also in jedem Fall Hilfstakt aus Timer des Controllers.

Links

`http://www.embeddedforth.de/`
`http://de.wikipedia.org/wiki/DCF77` Die Anlage in Mainflingen. Reichweite: über 2000 km, also ganz Mitteleuropa. Zeitzeichen für rund 100 Millionen Funkuhren

Listing

```
1   <|  \ DCF77.txt
2
3   \ DCF77 for GP32
4
5   HEX
6
7   : BYTE-SWAP     \ ( UN1 --- UN2 )
8                   \ exchange bytes
9   DUP 8<SHIFT SWAP 8SHIFT> OR ;
10
11  : LXOR \ ( flag1 flag2 --- flag3 )
12         \ logical XOR
13  IF IF 0 ELSE 1 THEN
14  ELSE IF 1 ELSE 0 THEN THEN ;
15
16  8 ZVARIABLE DCF-REG
17  2 ZVARIABLE THRES1
18  1 ZVARIABLE THRES2
19
20  : CD.     \ ( C  ---  ) C = 0 ... 99
21            \ print 2 digits decimal
```

[1] Anmerkung der Redaktion: In den Bits 1 ... 14 werden seit Ende 2006 Daten zum Wetter und zum Katastrophenschutz von der Schweizer Firma MeteoTime übertragen.

```
22  DUP 9 U> IF
23    9 1 DO
24    0A - DUP
25    0A U< IF LEAVE I THEN
26    LOOP 30 + EMIT
27  ELSE SPACE
28  THEN 30 + EMIT ;
29
30  : BIT->ADDR \ ( C1 --- N1 C2 )
31              \ C1 = 0 ... 58 = number of bit
32              \ N1, C2 = addr of bit
33  DUP 1SHIFT> 1SHIFT> 1SHIFT> DCF-REG
34  + SWAP 07 AND ;
35
36  : (CHECKSUM?)   \ ( C3 F1 C2 C1  --- F2 )
37  DO I BIT->ADDR B@ LXOR LOOP
38  SWAP BIT->ADDR B@ LXOR LOR ;
39
40  : CHECKSUM?     \ ( --- F1 ) F1 = 1 : ok
41  0 D% 28 0 D% 27 D% 21 (CHECKSUM?)
42    D% 35 0 D% 34 D% 29 (CHECKSUM?)
43    D% 58 0 D% 57 D% 36 (CHECKSUM?) LNOT ;
44
45  : 10*           \ ( C1 --- C2 )
46  DUP DUP 4<SHIFT 1SHIFT> + + ;
47
48  : BCD>BIN   \ ( C1 --- C2 ) BCD digits to bin
49  DUP 0F AND SWAP 4SHIFT> 10* + ;
50
51  : DCF-REG+@  \ ( C1 --- C2 )
52  DCF-REG + @
53  BYTE-SWAP    \ for bigendian 68HC08  CPU
54  ;
55
56  : TIME@        \ ( --- SEC MIN HOURS )
57  0                                          \ seconds
58  2 DCF-REG+@ 4SHIFT> 1SHIFT> 7F AND BCD>BIN   \ minutes
59  3 DCF-REG+@ 4SHIFT> 1SHIFT> 3F AND BCD>BIN ; \ hours
60
61  : DATE@        \ ( YEAR MONTH DAY WEEKDAY --- )
62  6 DCF-REG+@ 1SHIFT> 1SHIFT> FF AND BCD>BIN   \ year
63  5 DCF-REG+@ 4SHIFT> 1SHIFT> 1F AND BCD>BIN   \ month
64  4 DCF-REG+@ 4SHIFT>         3F AND BCD>BIN   \ day
65  5 DCF-REG+@ 1SHIFT> 1SHIFT> 07 AND           \ weekday
66  ;
67
68  TABLE DAY-TXT
69  ," ........." ," Montag...." ," Dienstag.." ," Mittwoch.."
70  ," Donnerstag" ," Freitag..." ," Samstag..." ," Sonntag..."
71
72  : DAY.                        \ ( C1 ---  )
73  10* DUP 9 + SWAP DO DAY-TXT I + C@ EMIT LOOP 2 SPACES ;
74
75  : TIME.       \ (  ---  ) print
76                \ HH:MM:SS  day-of-week  DD.MM.YY
77  CHECKSUM? IF
78  TIME@ CD. ." :" CD. ." :" CD. 2 SPACES
79  DATE@ DAY. CD. ." ." CD. ." ." CD.
80  THEN ;
81
```

```
82  : PULSE?       \ ( --- Flag ) DCF77: high-pulse
83  PD 4 B@        \ PD 4 input pin
84  ;
85
86  : SYNC              \ (  ---  )
87  BEGIN               \ wait for a missing pulse
88    1                 \ timeout flag
89    THRES1 @ 0 DO
90    1 MSEC
91    PULSE?             \ rising edge ?
92    IF DROP 0 LEAVE THEN
93    LOOP
94  UNTIL ;
95
96  : REG.      \ (  ---  ) print raw data
97  CR 7 0 DO DCF-REG I + C@ CH. LOOP ;
98
99  : READ-BITS                  \ (  ---  )
100 0                       \ number of bit
101 BEGIN
102 BEGIN PULSE? LNOT UNTIL
103 BEGIN PULSE? UNTIL     \ wait for rising edge
104 THRES2 C@ MSEC         \ delay for sample point
105 DUP BIT->ADDR
106 PULSE? IF B1! ELSE B0! THEN   \ store bit
107 DUP D% 58 =
108   IF DROP 1
109   REG. 2 SPACES TIME.
110   ELSE 1+ 0 THEN
111 UNTIL ;
112
113 : RUN                        \ (  ---  )
114 D%  170 THRES2 C!
115 D% 1400 THRES1 !
116 BEGIN
117   SYNC
118   READ-BITS
119 AGAIN ;
120
121 : TEST1  \ (  ---  )
122         \ print inputpin
123 BEGIN
124 PULSE? IF 31 ELSE 30 THEN EMIT D% 30 MSEC
125 AGAIN ;
126
127 : TEST2      \ (  ---  )
128              \ test data
129    \ 23:52:00 Montag 10.07.78
130 40 DCF-REG 2+  C!
131 7A DCF-REG 3 + C!
132 0C DCF-REG 4 + C!
133 E5 DCF-REG 5 + C!
134 E0 DCF-REG 6 + C!
135 05 DCF-REG 7 + C!
136 TIME. CR ;
137
138 |>
```

Grafik für Gforth

Hannes Teich

Um es gleich vorweg zu sagen: Der Plotter ist ein Android-Tablet, in meinem Fall der Nexus-7 von Google/Asus. Die Verbindung dorthin übernimmt eine Datei. Ich habe oft bedauert, dass Gforth keine Grafik-Schnittstelle bietet. Das ist seiner Portabilität geschuldet, denn jede Plattform will grafisch anders bedient werden. Bisher habe ich mir mit Basic-256 geholfen, das unter Linux und Windows läuft. Mit einer Bildschirmauflösung von 1820x1024 (Full HD) kriege ich damit 1200x666 als Grafikfenster.

Android und Mobile Basic

Nun bin ich kürzlich in die Android-Welt eingestiegen, und weil ich dort zwar Gforth vorfinde, aber nicht Basic-256, so habe ich gesucht – und etwas noch Besseres gefunden, nämlich *Mobile Basic*, sehr tüchtig und gut dokumentiert, mit vielen Beispielen. Ich habe mir ein Dateiformat ausgedacht und eine entsprechende Grafik-Datei zunächst mit Basic erzeugt, mit Basic gelesen und mit Basic auf den Bildschirm gezeichnet. Als das funktionierte, war es nicht mehr schwer, solch eine Datei mit Gforth zu erzeugen. Als Android-Neuling habe ich mich schwer getan (und tue es noch), das Dateisystem zu durchschauen. Eine gute Übersicht habe ich noch nicht entdeckt. Deshalb ist vieles durch Versuch, Irrtum und Heureka zustande gekommen. Ob es elegantere Lösungen gibt, weiß ich nicht; ich beschreibe einfach, wie es bei mir funktioniert.

Gforth auf Android

Gforth läuft auf Android im Ordner `/storage/emulated/0/gforth/site-forth`, ist zwar auch anders adressierbar, aber das halbe Dutzend File-Manager, die ich mir geholt habe, halten `/storage/emulated/0` für die gängige Ebene. Gforth legt seine Ausgabe-Dateien bevorzugt in `/storage/emulated/0/gforth/home` ab, und deshalb verwende ich diesen Ordner im Folgenden, obgleich ich den Weg vom Desktop zum Tablet schildern möchte. Die Tastatur des Desktop ist einfach leichter zu bedienen.

Zu den Listings

Es folgen zwei Listings, eines in Gforth, das andere in Mobile Basic geschrieben. Zusammen erzeugen sie das Bild (Abb. 1), freilich ohne die Gimp-Verfremdung, vielmehr in zarten weißen Linien auf schwarzem Grund. Vorher liste ich noch den Zyklus der Handgriffe auf, um von der Formulierung der Grafik bis zum Ergebnis zu kommen. Ich verwende mit Erfolg die zwei Apps: *wifi file transfer pro* von smarterDroid zur drahtlosen Datenübertragung sowie den *Android File Manager* von SmartWho.

- Plotdaten im Gforth-Programm bearbeiten
- Terminal: `~/Desktop$ gforth plotter.fs` (plot.dat entsteht)
- Am Tablet wifi file transfer pro starten (http://192.168.1.100:1234)
- Browser am Desktop starten
- Adresse http://192.168.1.100:1234 im Adressfeld eingeben
- Ordner `/storage/emulated/0/gforth/home` wählen
- Älteres `plot.dat` löschen (sonst wird neues umbenannt)
- „ Dateien auswählen“ anklicken
- `plot.dat` im Desktop anwählen
- „ Upload starten“ anklicken
- Am Tablet wifi file transfer pro beenden
- Mobile Basic aufrufen
- plotout.bas starten (holt sich plot.dat aus home)
- Voila! — Die Grafik erscheint.

Und um das Ergebnis zu retten:

- Einen Snapshot machen (Austaste und Leiser-Taste zugleich)
- Den Plot in der Galerie aufsuchen
- Wenn zufrieden, mit wifi file transfer pro zum Desktop holen
- Dazu in `/storage/emulated/0/Pictures/Screenshots` nachsehen

So, das war's. Weitere Informationen in den Listings. Runterladen kann man das hier: `http://www.forth-ev.de/filemgmt/singlefile.php?lid=525`

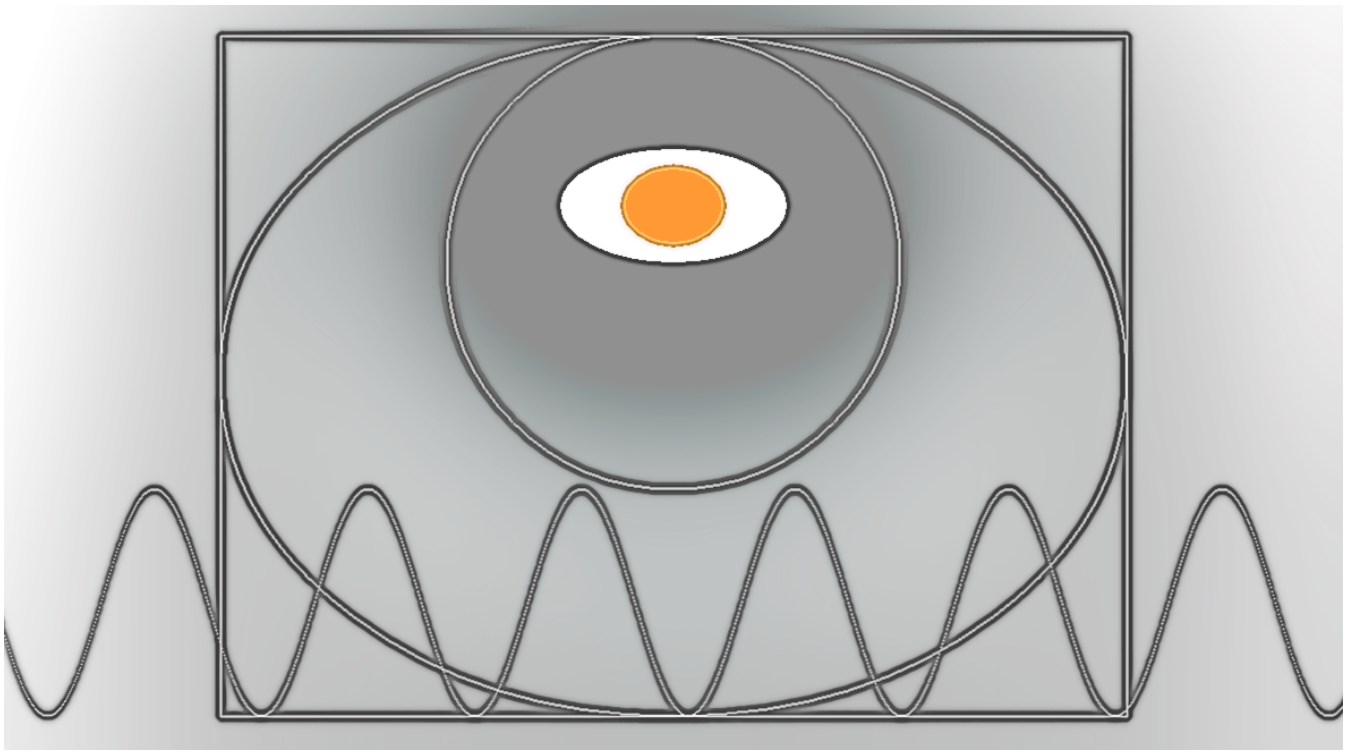


Abbildung 1: Grafik, mit Gforth erzeugt, nachträglich mit Gimp verfremdet

Das Gforth-Listing

```
\ ===+=== 1 ===+=== 2 ===+=== 3 ===+=== 4 ===+=== 5 ===+=== 6 ===+===
\ plotter.fs - last edit: 19-feb-2014 21:00 -jgt
\
\ ######################################################################
\ Generieren einer durch Mobile Basic zu interpretierenden Plot-Datei
\ ######################################################################
\
\ Fuer die Grafik-Datei sind (vorerst nur) 10 Befehle vorgesehen, mit
\ denen in beliebigen Farben Linien, Kreise, Ellipsen, Rechtecke und
\ vor allem einzelne Pixel gesetzt werden koennen. Die Befehle werden
\ jeweils durch Kennnummern eingeleitet. Kennnummern und Parameter
\ haben 16 bit Breite (Bytes-Anordnung: big endian). Fuer den Befehl
\ plot sind als Parameter beliebig viele xy-Paare erlaubt, die durch
\ eine Ende-Kennung abgeschlossen werden. clear fuellt den Bild-
\ schirm mit vorgewaehlter Farbe, show macht die gesetzte Grafik
\ sichtbar.
\
\   "_color"    | $AA01 |  <r>  |  <g>  |  <b>  |  <a>  |
\   "_clear"    | $AA02 |
\   "_line"     | $AA03 |  <x1> |  <y1> |  <x2> |  <y2> |
\   "_rect"     | $AA04 |  <x>  |  <y>  |  <w>  |  <h>  |  <f>  |
\   "_oval"     | $AA05 |  <x>  |  <y>  |  <w>  |  <h>  |  <f>  |
\   "_circle"   | $AA06 |  <x>  |  <y>  |  <r>  |  <f>  |
\   "_plot"     | $AA07 |  <x1> |  <y1> | . . . | . . . | $AA00 |
\   "_pause"    | $AA08 |  <s>  |
\   "_show"     | $AA09 |
\   "_end"      | $AA0A |
\
\ <r><g><b> = rot gruen blau (0 0 0 = schwarz, 255 255 255 = weiss)
\ <a> = alpha (0 = transparent, 255 = deckend)
\ <x><y><r> = Kreismittelpunkt und -radius
\ <x><y> = linke obere Ecke (Rechteck und Rahmen fuer Ellipse)
\ <w><h> = Breite und Hoehe (Rechteck und Rahmen fuer Ellipse)
\ <f> = ausfuellen (true/false fuer Kreis, Ellipse und Rechteck)
```

```
35  \ <s> = Sekunden (fuer Pause)
36
37    s" deskplot.dat" 2constant  wfile       \ Dateiname
38    0                value      wfileID     \ Dateikennung
39    2000             constant   bufsize     \ Puffergroesse
40                     variable   bufcnt      \ Pufferzaehler
41                     variable   tmp         \ Fluechtige Variable
42
43  create wbuffer bufsize allot              \ Schreibpuffer
44
45  \ Ausgabedatei erzeugen
46  : create-wfile  ( --)
47              wfile R/W BIN  ( c-addr u fam)
48              CREATE-FILE      ( fileid ior)
49          IF    ." couldn't create " wfile type ."  - quit" quit
50          THEN  to wfileID ;
51
52  \ Daten in Wave-Datei schreiben.
53  : >file  ( addr len)  wfileID  WRITE-FILE ( ior) drop ;
54
55  \ Schreibpuffer leeren und in Datei schreiben.
56  \ FLUSH-FILE leert auch den unsichtbaren Zwischenspeicher.
57  \ Am Programmende ist der Puffer mit bufflush zu leeren.
58  : bufflush  ( --)
59              wbuffer bufcnt @ >file
60              0 bufcnt !
61              wfileID FLUSH-FILE drop ;
62
63  \ Schreibpuffer laden und ggf. in die Wave-Datei uebertragen.
64  \ So lange die Daten nicht in den Puffer passen, wird der Puffer
65  \ randvoll gefuellt und sodann in die Wave-Datei entleert.
66  \ Dann wird der Datenrest in den Puffer geladen.
67  : $>buf  ( addr len --)
68          BEGIN   bufsize bufcnt @ - >r
69                  dup r@ >=
70          WHILE   over wbuffer bufcnt @ + r@ move
71                  swap r@ + swap  r> -
72                  wbuffer bufsize >file
73                  0 bufcnt !
74          REPEAT  r> drop  dup
75          IF      >r wbuffer bufcnt @ + r@ move
76                  bufcnt @ r> + bufcnt !
77          ELSE    2drop
78          THEN ;
79
80  \ 2 niederwertige Bytes vertauschen
81  : byte-swap  ( u1 -- u2 )  dup 8 rshift 255 and
82                             swap 255 and 8 lshift or ;
83
84  \ 2-byte-Wert in den Schreibpuffer schreiben.(big endian)
85  : 2>buf  ( u --)  byte-swap  tmp !  tmp 2 $>buf ;
86
87
88  \ ##############################
89  \ Grafikdaten in Datei schreiben
90  \ ##############################
91
92  \ Dateibeginn
93  : _begin  ( -- )  cr ." begin "
94          bufflush ;
95
96  \ Farbe waehlen
97  : _color  { r g b a -- }  ." color "
98          $AA01 2>buf  r 2>buf  g 2>buf  b 2>buf  a 2>buf ;
99
100 \ Bildschirm loeschen
```

```
101  : _clear  ( -- )  ." clear "
102          $AA02 2>buf ;
103
104  \ Gerade zeichnen
105  : _line  { x1 y1 x2 y2 -- )  ." line "
106          $AA03 2>buf  x1 2>buf  y1 2>buf  x2 2>buf  y2 2>buf ;
107
108  \ Rechteck zeichnen (hohl oder gefuellt)
109  : _rect  { x y w h f -- }  ." rect "
110          $AA04 2>buf  x 2>buf  y 2>buf  w 2>buf  h 2>buf  f 2>buf ;
111
112  \ Ellipse zeichnen (hohl oder gefuellt)
113  : _oval  { x y w h f -- }  ." oval "
114          $AA05 2>buf  x 2>buf  y 2>buf  w 2>buf  h 2>buf  f 2>buf ;
115
116  \ Kreis zeichnen (hohl oder gefuellt)
117  : _circle  { x y r f -- }  ." circle "
118          $AA06 2>buf  x 2>buf  y 2>buf  r 2>buf  f 2>buf ;
119
120  \ Ein oder mehrere Pixel setzen
121  : _plot-on   ( -- )        $AA07 2>buf  ." plot-on " ;
122  : _plot      { x y }  x 2>buf  y 2>buf  ." plot " ;
123  : _plot-off  ( -- )        $AA00 2>buf  ." plot-off " ;
124
125  \ Pause (Sekunden)
126  : _pause  { s -- }  ." pause "
127          $AA08 2>buf  s 2>buf ;
128
129  \ Gesetzte Grafik sichtbar machen
130  : _show  ( -- )  ." show "
131          $AA09 2>buf ;
132
133  \ Dateiende
134  : _end  ( -- )  ." end " cr cr
135          $AA0A 2>buf  bufflush ;
136
137
138  \ ########################
139  \ Eine Grafik formulieren
140  \ ########################
141
142    create-wfile                    \ Ausgabe-Datei erzeugen
143
144  : go   bufflush
145                          _begin      \ Dateibeginn
146       0   0   0 255     _color      \ schwarz deckend
147                          _clear      \ Bildschirm fuellen
148    255 255 255 255       _color      \ weiss deckend
149    400 300 800 600   0   _rect       \ Rechteck
150    400 300 800 600   0   _oval       \ Ellipsen
151    800 500 200   0       _circle     \ Kreis
152                          _show       \ Grafik aktualisieren
153                      2   _pause      \ Pause 2 Sekunden
154    700 400 200 100   1   _oval       \ Ellipse ausgefuellt
155    255  40  40 127       _color      \
156    750 410 100  80   1   _oval       \ Ellipse ausgefuellt
157                          _show       \ Grafik aktualisieren
158                      2   _pause      \ Pause 2 Sekunden
159    255 180 180 255       _color      \ Rot fuer Plot
160                          _plot-on    \ Plot
161    1200 0 do i 200 + i s>f 30e f/ fsin 100e f* 800e f+ f>s
162                          _plot       \ Sinuskurve
163           loop           _plot-off
164                          _show       \ Grafik aktualisieren
165                     10   _pause      \ Pause 10 Sekunden
166                          _end ;      \ Dateiende
```

```
167
168     go    \ Autostart
169
170   \ ====================================================================
171
```

Das Basic-Listing

```
1   \ ===+=== 1 ===+=== 2 ===+=== 3 ===+=== 4 ===+===
2   plotout.bas - Datei steuert Plotter
3   (19-feb-2014 21:00 -jgt)
4
5   Dies ist ein Plotter-Programm fuer Mobile Basic
6   unter Android. Eine Datei mit Grafik-Befehlen
7   steuert die Bildschirm-Grafik. Die Datei plot.dat
8   wird in /storage/emulated/0/gforth/home erwartet.
9   Dieser Ordner wird von Gforth fuer die Ausgabe
10  benutzt.
11
12  Folgende Befehle werden erkannt:
13     COLOR, CLEAR, LINE, RECT, OVAL, CIRCLE, PLOT,
14     PAUSE, SHOW, END
15
16
17  function int(x as short) as short
18      int=integer(x) & 0xFF
19  end function
20
21  sub gocolor      // {1} r g b a
22      print "color"
23      dim r,g,b,a as short
24      get #1,r
25      get #1,g
26      get #1,b
27      get #1,a
28      setcolor r,g,b,a
29  end sub
30
31  sub goclear      // {2}
32      print "clear"
33      cls
34  end sub
35
36  sub goline       // {3} x1 y1 x2 y2
37      print "line"
38      dim x1,y1,x2,y2 as short
39      get #1,x1
40      get #1,y1
41      get #1,x2
42      get #1,y2
43      drawline x1,y1,x2,y2
44  end sub
45
46  sub gorect       // {4} x y w h f
47      print "rect"
48      dim x,y,w,h,f as short
49      get #1,x
50      get #1,y
51      get #1,w
52      get #1,h
53      get #1,f
54      if integer(f)=0 then
55          drawrect x,y,w,h
56      else
57          fillrect x,y,w,h
58      end if
59  end sub
60
61  sub gooval       // {5} x y w h f
62      print "oval"
63      dim x,y,w,h,f as short
64      get #1,x
65      get #1,y
66      get #1,w
67      get #1,h
68      get #1,f
69      if integer(f)=0 then
70          drawoval x,y,w,h
71      else
72          filloval x,y,w,h
73      end if
74  end sub
75
76  sub gocircle     // {6} x y r f
77      print "circle"
78      dim x,y,r,f as short
79      get #1,x
80      get #1,y
81      get #1,r
82      get #1,f
83      if integer(f)=0 then
84          drawcircle x,y,r
85      else
86          fillcircle x,y,r
87      end if
88  end sub
89
90  sub goplot       // {7} x1 y1 [x2 y2 [...]] {0}
91      print "plot"
92      dim x,y as short
93      dim endflag as boolean
94      endflag=false
95      while endflag=false
96          get #1,x
97          if x=short(0xAA00) then
98              endflag=true
99          else
100             get #1,y
101             plot x,y
102         end if
103     end while
104 end sub
105
106 sub gopause      // {8} s
107     print "pause"
108     dim s as short
109     get #1,s
110     sleep integer(s)*1000
111 end sub
112
113 sub goshow       // {9}
114     print "show"
```

```
115         repaint
116     end sub
117
118     sub main
119         dim home,file as string
120         dim ding as short
121         dim item,cmd as integer
122         dim basta,eflag as boolean
123
124         graphics
125
126         home="/storage/emulated/0/"
127         file="gforth/home/deskplot.dat"
128
129         open #1,home+file,"r"
130
131         repeat
132             repeat
133                 get #1,ding
134                 item=integer(ding)
135             until item<0xAA00
136             cmd=item & 0xFF
137             if cmd=1 then
138                 call gocolor
139             elseif cmd=2 then
140                 call goclear
141             elseif cmd=3 then
142                 call goline
143             elseif cmd=4 then
144                 call gorect
145             elseif cmd=5 then
146                 call gooval
147             elseif cmd=6 then
148                 call gocircle
149             elseif cmd=7 then
150                 call goplot
151             elseif cmd=8 then
152                 call gopause
153             elseif cmd=9 then
154                 call goshow
155             elseif cmd=10 then
156                 print "end"
157                 eflag=true
158                     end if
159         until eflag=true
160
161         close #1
162
163      // basta=delete(home+file)
164         if basta then
165             print "file deleted"
166         else
167             print "file not deleted"
168         end if
169     end sub
170
```

. . . Fortsetzung der Meldung zum 30c3 von Seite 13

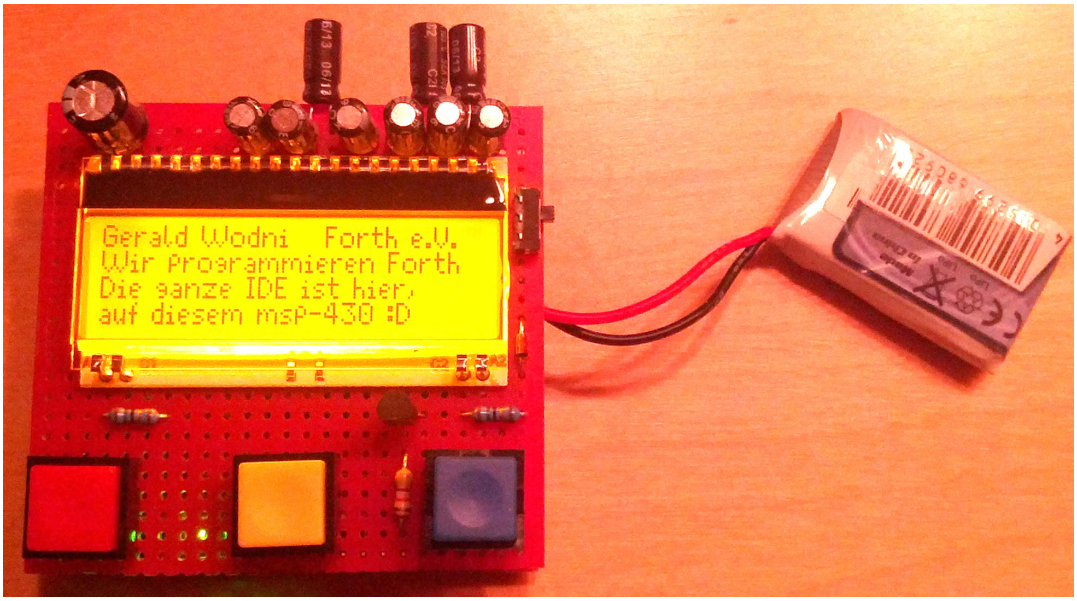


Abbildung 2: Das Namensschild mit dem msp430

4€4th–IDE unter Linux und MacOS X

Carsten Strotmann

4€4th

Das 4€4th [1] ist ein kleines Forth für das TI Launchpad MSP-EXP430G2 Board. Der Name kommt vom ursprünglichen Einstiegspreis des MSP430 Launchpad Boards, 4.30 US$. Derzeit liegt der Preis für das Board um die 10 Euro, teurer, aber immer noch günstig.

Michael Kalus und Dirk Brühl haben das kleine 4€4th für dieses Board entwickelt, und Dirk Brühl hat eine IDE (grafische Entwicklungsumgebung) für Windows für dieses Forth geschrieben. Für diesen Artikel habe ich die Version der 4€4th von Anfang Februar benutzt. Bei älteren Versionen lassen sich die Zeichensätze nicht wie unten beschrieben anpassen.

Die IDE [2] ist für Windows geschrieben, kann aber mittels WINE [3] auch unter Linux und MacOS X ausgeführt werden[1]. WINE ist ein spezielles Programm, welches Windows–Programmen unter Linux oder MacOS X eine Windows–Laufzeitumgebung bereitstellt. Das Programm wird nativ auf dem Prozessor ausgeführt, die Betriebssystemaufrufe für Windows werden abgefangen und in die entsprechenden Linux oder MacOS X Systemaufrufe übersetzt. Daher der Name WINE = *WINE Is Not an Emulator*. Mittels WINE können viele Windows–Programme unter Linux und MacOS X ausgeführt werden, so auch die 4€4th–IDE.

Im Folgenden beschreibe ich, was dabei zu beachten ist:

[1] auf Intel–x86–Prozessoren

Linux

Unter Linux befindet sich WINE in den Programm–Repositories der Linux–Distributionen. WINE wird über die Paketverwaltung der Linux Distribution installiert (`yum` bei Fedora, `apt-get` oder `aptitude` bei Debian oder Ubuntu, `slackpkg` bei Slackware etc.).

Nun muss für die WINE–Umgebung die serielle Schnittstelle von Linux auf eine serielle Schnittstelle von Windows abgebildet werden. Das TI Launchpad meldet sich bei Linux unter dem Namen `/dev/ttyACM0`. Um diese Unix–Schnittstelle in die Windows–Umgebung zu bringen, legen wir einen symbolischen Verweis (Link) auf eine freie COM-Schnittstelle im Verzeichnis `~/.wine/dosdevices` an:

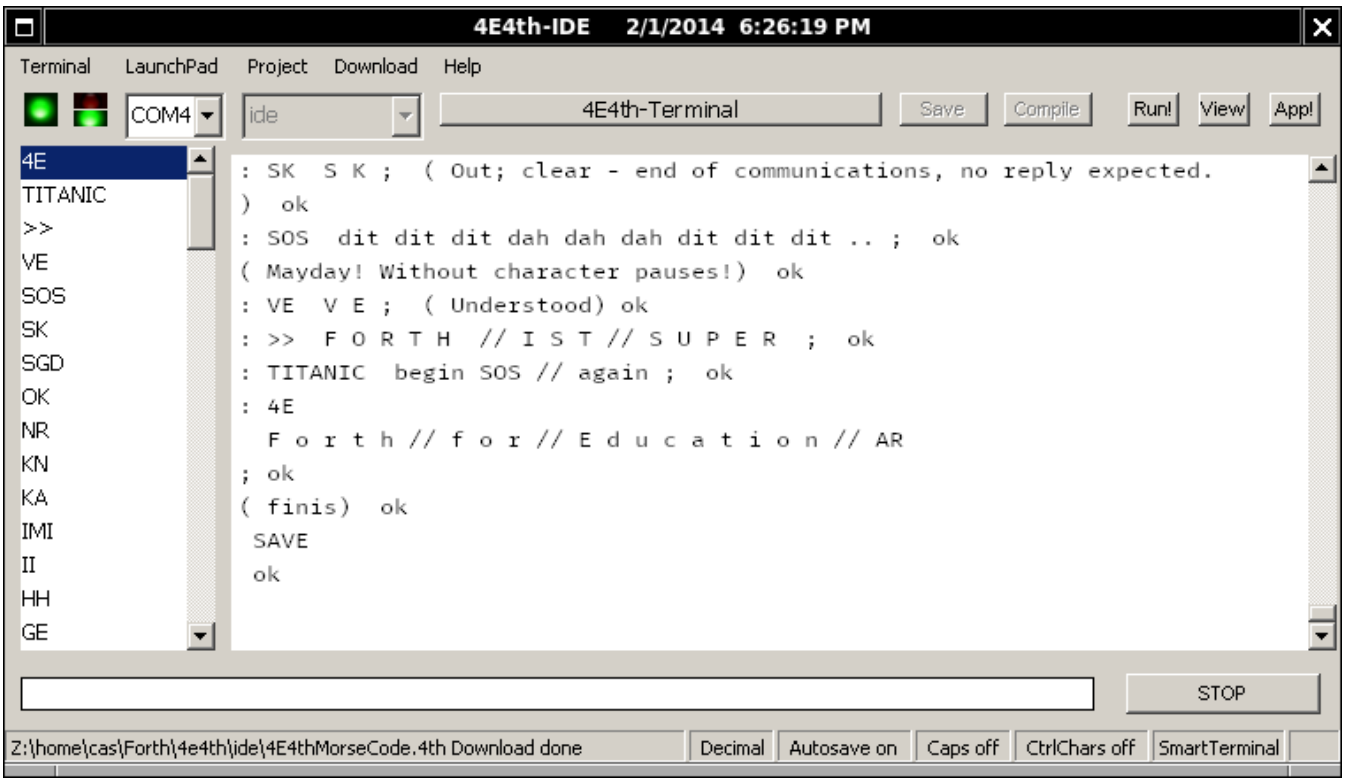


Abbildung 1: 4€4th–IDE unter Linux

```
shell> ln -s /dev/ttyACM0 ~/.wine/dosdevices/com4
```

Nun die 4€4th–IDE von der Webseite laden und in ein Verzeichnis auspacken. Die IDE kann von einem Datei–Manager–Programm oder von der Kommandozeile gestartet werden:

```
shell> cd Forth/4e4th/ide
shell> ls
4E4th-IDE.exe
4E4thMorseCode.4th
Backup
COM4Session.4th
Engine
Liesmich.txt
SLEEP.4th
stars.4th
shell> wine 4E4th-IDE.exe
```

Die Schnittstelle auswählen und loslegen.

Die Standard–Zeichensätze (Fonts) in der 4€4th–IDE kommen vom darunterliegenden Win32Forth und sehen unter WINE oft unansehnlich aus. Dirk Brühl hat eine Plugin–Schnittstelle in die 4€4th–IDE eingebaut, mit welcher das Aussehen, und damit auch die Zeichensätze der IDE, angepasst werden können. Dazu muss die Datei `Plugins.f` in das Verzeichnis `Engine/Plugs` unterhalb der IDE kopiert werden. In der Datei `Plugins.f` können die Zeichensätze eingestellt werden. Unter Linux benutze ich *Fira Mono* [4] aus dem Firefox–OS–System:

```
\ Changing the Appearance
140 zoom4E4th-IDE  46  zoom4E4th-IDE
680 zoom4E4th-IDE 305  zoom4E4th-IDE
    Move: EditWindow
 10 zoom4E4th-IDE  41  zoom4E4th-IDE
120 zoom4E4th-IDE 310  zoom4E4th-IDE
    Move: ListOfWords
70 to c/l    \ characters per line

\ Changing of Fonts
Font Test-font
: Set-Test-font ( -- )
   ." Set-Test-font "
     -12               Height: Test-font
       8                Width: Test-font
       0           Escapement: Test-font
       0          Orientation: Test-font
     400               Weight: Test-font
       0              CharSet: Test-font
       3         OutPrecision: Test-font
       2        ClipPrecision: Test-font
       1              Quality: Test-font
      18       PitchAndFamily: Test-font
       s" Fira Mono" SetFacename: Test-font
   Create: Test-font
   Handle: Test-font SetFont: EditWindow
;

Set-Test-font
  Handle: Test-font SetFont: ProjectCmbList
  Handle: Test-font SetFont: CmbList1
  Handle: Test-font SetFont: ListOfWords
  Handle: Test-font SetFont: EditWindow
  Handle: Test-font SetFont: Label1
  Handle: Test-font SetFont: TerminalInput
  Handle: Test-font SetFont: ModusLabel

  ' Set-Test-font is Set-Welcome-font
  ' Set-Test-font is Set-Text-font
  ' Set-Test-font is Set-Terminal-font
```

MacOS X

Um das MSP430–Launchpad am MacOS X–Rechner betreiben zu können, muss der FTDI–USB–Seriell–Treiber installiert werden. Eine Anleitung zur Installation des Treibers findet sich im Wiki der Forth-Gesellschaft [5].

WINE benötigt unter MacOS X die X11 Grafikschnittstelle. Bei älteren MacOS X–Versionen ist die X11–Schnittstelle auf den Installations–DVDs zu finden, bei MacOS X 10.8 und 10.9 kann die jeweils aktuelle Version aus dem Internet geladen werden [6].

Unter MacOS X gibt es mehrere Möglichkeiten, die WINE–Umgebung zur Ausführung von Windows–Programmen zu installieren. Ich bevorzuge WineBottler [7], welches es erlaubt, Windows–Programme so umzuwandeln, dass sie wie MacOS X–Programme (Apps) funktionieren und als vollständige Einheiten verschoben und kopiert werden können.

Nachdem WineBottler installiert ist, kann die 4€4th–ZIP–Datei entpackt und das `4e4th-ide.exe` per Doppelklick aus dem Finder gestartet werden.

Ich habe für die 4€4th–IDE auf meinem System eine eigene *Bottle* (WineBottler Begriff für eine abgeschottete Windows–Ausführungs–Umgebung) unter dem Verzeichnis `/Users/cas/Documents/4e4th` erstellt. Damit die USB–serielle Schnittstelle für die 4€4th–IDE sichtbar und benutzbar wird, muss ein Link (Symlink) von der Gerätedatei auf eine COM–Schnittstelle innerhalb der *Bottle* erstellt werden (hier COM4:):

```
shell> sudo ln -s \
  /dev/tty.uart-B6FF46762582362C \
  /Users/cas/Documents/4e4th/dosdevices/com4
```

(die Zeichenkette nach `tty.uart-` ist bei jeder Installation anders, daher bitte vor dem Erstellen des Links nachschauen. Die Zeile ist hier mit \ zum maskieren der Zeilenenden umgebrochen.).

Die Standard–Zeichensätze sind, wie auch bei Linux, unter MacOS X nicht optimal. Wie oben für Linux–Systeme beschrieben, können die Zeichensätze mittels der Datei `Plugins.f` angepasst werden.

Alternativen

WINE bietet eine sehr leichtgewichtige Lösung, um 4€4th unter nicht–Windows–Systemen auszuführen. Eine andere Alternative sind *System–Emulatoren*: Programme, welche einen ganzen PC simulieren (oder emulieren), um dort das Windows–Betriebssystem auszuführen. Diese Lösungen benötigen jedoch zwingend eine Lizenz des Microsoft–Windows–Betriebssystems, sind daher immer mit zusätzlichen Kosten verbunden.

Die System–Emulatoren sind als OpenSource, kostenlos oder auch als kostenpflichtige Programme erhältlich:

- KVM — Linux Kernel Virtual Machine: `http://www.linux-kvm.org`
- VirtualBox — Machine Emulator für Linux, Windows, Solaris und MacOS X: `http://virtualbox.org`
- Qemu — kann x86 Systeme auch auf anderen Prozessoren emulieren: `http://wiki.qemu.org/Main_Page`
- VMWare Player — kostenlos und closed–source `http://www.vmware.com/products/player/`
- VMWare Workstation — kostenpflichtig `http://www.vmware.com/products/workstation/`
- Parallels — kostenpflichtig `http://www.parallels.com/de/`

Referenzen

1. `http://4e4th.eu`
2. `http://www.4e4th-i.de`
3. `http://www.winehq.org`
4. `http://www.mozilla.org/en-US/styleguide/products/firefox-os/typeface`
5. `http://forth-ev.de/wiki/doku.php/projects:4e4th:start`
6. `http://xquartz.macosforge.org/landing`
7. `http://winebottler.kronenberg.org`

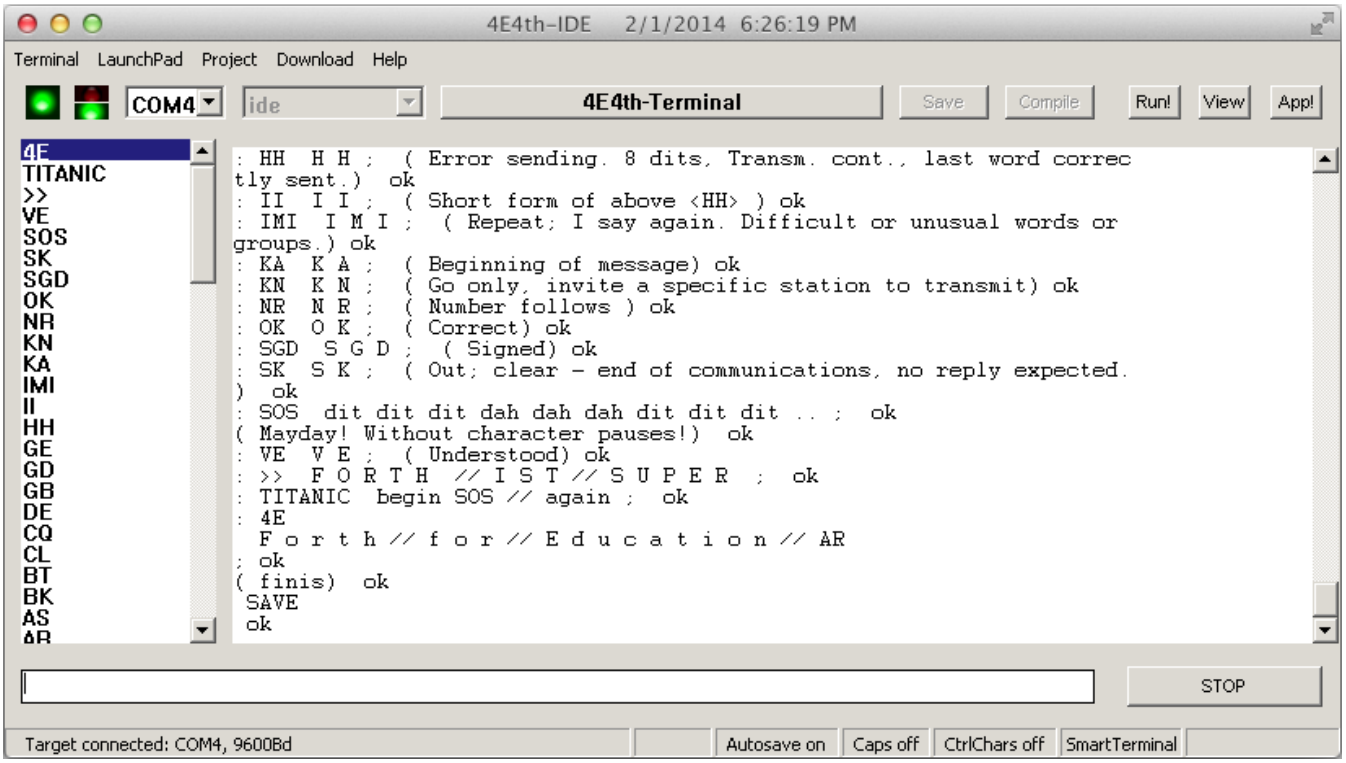


Abbildung 2: 4€4th–IDE unter MacOS X

Forth–Gruppen regional

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

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

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

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

Gruppengründungen, Kontakte

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

μP–Controller Verleih

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

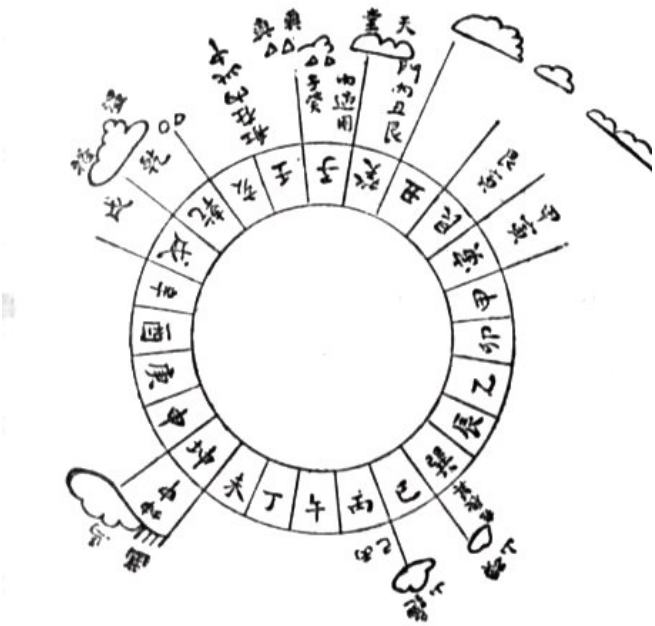
Spezielle Fachgebiete

FORTHchips (FRP 1600, RTX, Novix)	**Klaus Schleisiek–Kern** Tel.: **(0 40) – 37 50 08 03** (g)
KI, Object Oriented Forth, Sicherheitskritische Systeme	**Ulrich Hoffmann** Tel.: **(0 41 03) – 80 48 41** (g) Email: `uho@forth-ev.de`
Forth–Vertrieb volksFORTH ultraFORTH RTX / FG / Super8 KK–FORTH	Ingenieurbüro **Klaus Kohl–Schöpe** Tel.: **(0 82 66)–36 09 862** (p)

Termine

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

27.–30. März 2014: Forth–Tagung, Bad Vöslau bei Wien
8.–10. Mai 2014: LinuxTag, Berlin
`http://www.linuxtag.de`
28.–29. Juni 2014: Maker World (zusammen mit der Ham Radio), Messe Friedrichshafen
`http://www.maker-world.de`
5.–6. Juli 2014: MakerFaire Hannover
`http://www.makerfairehannover.com`



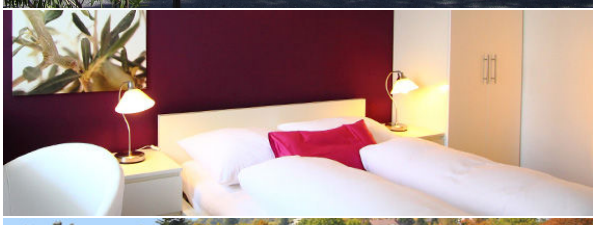
Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen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

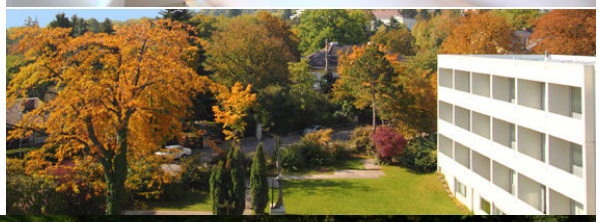
Hinweise zu den Angaben nach den Telefonnummern:
Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeitszeiten
g = geschäftlich
Die Adressen des Büros der Forth–Gesellschaft e.V. und der VD finden Sie im Impressum des Heftes.

Einladung zur

Forth–Tagung 2014 vom 27. bis 30. März in Bad Vöslau

Unterbringung und Tagung im College Garden Hotel in der Johann–Strauß–Straße 2, 2540 Bad Vöslau, Österreich.





Anreise

Bad Vöslau liegt südlich von Wien und ist mit dem Auto über die A2 erreichbar. Das Hotel bietet einen Taxiservice vom Bahnhof, oder auch vom Flughafen.

Anmeldung

Auf `http://tagung.forth-ev.de` finden sich noch viele weitere Informationen, das aktuellste Programm sowie die elektronische Anmeldung.

Programm

Donnerstag

ab 13:00	Frühankommer(–Workshops)

Freitag

vormittags	Frühankommer(–Workshops)
nachmittags	Begin der Tagung, Vorträge und Workshops

Samstag

vormittags	Vorträge und Workshops
nachmittags	Exkursion

Sonntag

09:00	Mitgliederversammlung
13:00	Ende der Tagung