Vierte Dimension

3/2006

Das Forth–Magazin

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

In dieser Ausgabe:

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische Informatik

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

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

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

LEGO RCX–Verleih

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

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

Inhalt:

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

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

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

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

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

KIMA Echtzeitsysteme GmbH

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

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

FORTecH Software
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

Budapester Straße 80 a D–18057 Rostock
Tel.: (0381) 46 13 99 10 Fax: (0381) 4 58 34 88

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

Ingenieurbüro
Dipl.–Ing. Wolfgang Allinger

Tel.: (+Fax) 0+212–66811
Brander Weg 6
D–42699 Solingen

Entwicklung von µC, HW+SW, Embedded Controller, Echtzeitsysteme 1–60 Computer, Forth+Assembler PC / 8031 / 80C166 / RTX 2000 / Z80 ... für extreme Einsatzbedingungen in Walzwerken, KKW, Medizin, Verkehr / >20 Jahre Erfahrung.

Ingenieurbüro
Klaus Kohl

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

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

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

Forth–Gesellschaft e. V.
Postfach 19 02 25
80602 München
Tel: (0 89) 1 23 47 84
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,

Bei sommerlichen Temperaturen und WM–Spektakel fließen VD–Beiträge erfahrungsgemäß eher spärlich. Trotzdem haben wir die VD wieder füllen können. Die entscheidende Lücke hat diesmal Erich Wälde gefüllt — ein Forth–Neuling, der erst auf dem Linux–Tag mit Forth Bekanntschaft gemacht hat. Seine ersten Experimente sehen nicht nach Anfängerwerk aus, und zeigen, dass man Forth in kurzer Zeit lernen kann — entsprechende Motivation vorausgesetzt.

Die Umfrage auf www.forth-ev.de unter unseren Lesern lässt darauf schließen das nur wenige Leser auch auf die Website schauen und dort dann auch abstimmen. Schade eigentlich, das Ihr unser weblog so wenig benutzt. Ob es beim nächsten Mal mehr Stimmen werden?

Artikel	Stimmen
Ankündigung EuroForth 2006	0
Vorschauf auf das Programm der Forthtagung im Mai 2006	0
Umwandlung von HEX nach BCD	0
Forth am Stil — Teil 1	1
Forth von der Pike auf — Teil 3 und 4	2
R8C-Forth	2
Forth erzeugt automatisch Hexadoku-Rätsel-Vorgaben	4
Lebenszeichen	0
Gehaltvolles	0
Format-Hinweise für Autoren	0

Deshalb, liebe Mitglieder: Wenn ihr der Meinung seid, es sollte über dies, das und jenes berichtet werden, aber das alles nicht in der VD zu finden ist, teilt es wenigstens mit. Uns oder dem weblog. Vielleicht findet sich dann sogar ein Autor, der dazu etwas weiß.

Und: Vielleicht wird das Wetter auch wieder schlechter, und ihr findet Zeit, eure Tagungsbeiträge ins Reine zu schreiben! Denn nach wie vor ist eines wahr: So eine Zeitung ist immer nur in einer Art Fließgleichgewicht der eingehenden Beiträge. Einen Vorrat gibt es nicht. Also spitzt die Griffel und legt los.

Ob vielleicht auch Intellasys (`http://www.intellasys.net/`) bis dahin Informationen über den neuen Forth–Prozessor von Chuck Moore heraus rückt? Dann können wir auch darüber berichten.

Bernd Paysan und Ulrich Hoffmann

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
`http://www.forth-ev.de/filemgmt/index.php`

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

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

Hexadoku von Martin Bitter

Lieber Fred,

Das war mal ein Artikel! Da zeigt ein Mathematiker, wie man Mathematik betreiben kann — im Hexadoku (Sudoku) bekannte mathematische Regeln entdecken, überprüfen und anwenden. So soll es gemacht sein! Logisch, fortschreitend, nachvollziehbar und glasklar!

Angeblich soll der *Erfinder* des Sudokus Wayne Gould einige Jahre an seinem Sudokuprogramm geschrieben haben — Du bist zu deiner Lösung wesentlich schneller gekommen. Bravo!

Noch offene Fragen: Die Zahl möglicher Sudokus ist bekannt $9! \cdot 722 \cdot 27 \cdot 27.704.267.971$ bzw. 5.472.730.538. (`http://de.wikipedia.org/wiki/Sudoku`). Kannst Du angeben, wieviele Sudokus mit deinen Methoden erzeugbar sind?

Gruß Martin

Obfuscate Email Addresses

Hallo Redaktion. Die neue Spammer-Sperre in der PDF-Version der VD finde ich prima. Ihr könnt ja zaubern! Für Menschen lesbar, für Maschinen nicht. Ich frage auch nicht, wie das geht...

Michael.Kalus@onlinehome.de

Und ist das Werk auch gut gelungen...

...bestimmt verträgt es Änderungen.

Lieber Forthfreund,

geh beherzt ran an das Forth–eV–Wiki. Als ich Bernd Paysan neulich vorsichtshalber per mail informieren wollte, dass mir an dem R8C Text auf dem Forth Wiki etwas unklar vorkam, bevor ich da im Wiki herum ändere, war er gar nicht so erbaut von meiner Mitteilung. Nicht, weil er inhaltlich anderer Meinung war, sondern wegen des Weges den ich eingeschlagen hatte — ich war viel zu vorsichtig da ran gegangen und damit zu umständlich. Hier der (fast, leicht gekürzte) O–Ton der mail:

> On Monday 01 May 2006 02:00, Kalus Michael wrote:
>
> > Wir mailten zu deinem Text im forth wiki. Ich will da nicht weiter eigenmächtig drin herumändern...
>
> Doch, mach' das — genau das ist der Zweck eines Wikis. Das Herumschicken von E–Mails an den, der die Seite pflegt, ist viel zu arbeitsaufwändig. Und *was* du genau geändert hast, kann ich mit dem diff–Tool des Wikis am besten sehen. Und wenn mir davon etwas nicht gefallen hätte, hätte ich das eben wieder so geändert, wie mir das passt.
>
> > Daher hier mein Vorschlag...
>
> Ja. Aber beim nächsten Mal einfach im wiki ändern. Ich habe das Forth–eV–Wiki in meinem RSS–Feed und sehe sowieso, wenn sich da was ändert. Bernd

Also schaut selbst was im forth wiki geändert oder ergänzt gehört — und macht es dann einfach.

Gruß, Michael

Chuck Moores jüngster Spross: SEAforth–Prozessoren.

Scalable Embedded Array™ (SEA) Platform nennt IntellaSys ihren Multicore–Processor–Chip. Es sind 24 C18–Forth–Prozessorkerne auf dem Chip, wovon jeder eine Milliarde Forth–Operationen in der Sekunde ausführen kann, und das bei nur 150mW in einer typischen Applikation.

Charles Moore steckt hinter dieser Firma, zusammen mit Chester Brown, Dave Guzeman, Yu–Ping Cheng, Jeff Fox, Allan L. Swai, Dennis R. Miller und Greg Peel. Die Gruppe hat viel vor und wendet sich an die Forth User Community:

> SEAforth processors execute 24 BILLION Forth instructions per second! We plan on shaking some things up in the embedded controller environment.

Auf ihrer Website kündigen sie den Chip SEAforth–24A an und beschreiben ihn näher. Die C18–Kerne sind 18–bit stackorientierte Maschinen in denen als Instruktionen VentureForth ausgeführt wird, eine VentureForth–Instruction je Nanosekunde. Man denkt dabei vor allem an Anwendungen aus den Bereichen: Consumer audio processing, Wireless and USB devices, Home automation, Remote data collection and processing, Security applications.

Also schaut mal rein bei:
`http://www.intellasys.net/users/index.php`

Fort(h)bildungsplattform Forth–Gesellschaft

Liebe VD, hier noch ein paar Zeilen zwischen M. und U. die ich kürzlich aus einer Mail gefischt habe:

> *Ich experimentiere mit Geeklog und dem Forth–Wiki zur Zeit sehr gerne und bin froh und dankbar, dass die Forth–Gesellschaft das ermöglicht hat, auf eigenem Server — finde ich toll.*
>
> Ja — das war schon immer eine ihrer Leistungen. Schade, dass so wenige das nutzen...

In der Tat. Dabei ist Platz genug vorhanden.

Viele Grüße, Michael

Ein neuer Direktor stellt sich vor

Ewald Rieger

Am 14. Mai 2006 wurde ich als Nachfolger von Prof. Dr. Fred Behringer zusammen mit den beiden amtierenden Direktoren Bernd Paysan und Dr. Ulrich Hoffmann einstimmig in das neue Direktorium gewählt. Zuerst möchte ich an dieser Stelle meinen herzlichen Dank an Fred richten. Über viele Jahre hat er sich engagiert für die Belange unseres Vereines eingesetzt. Ganz besonders möchte ich auf seine zahlreichen Kontakte zu den weltweiten Forth–Gruppierungen hinweisen. Ich wünsche ihm weiter viel Gesundheit und hoffe, dass er uns weiter tatkräftig zur Seite steht.

Ich möchte nun die Gelegenheit nützen, mich ein wenig vorzustellen. Denn die wenigsten unserer Mitglieder hatten Gelegenheit, mich auf den vielen Forth–Tagungen kennen zu lernen, an denen ich regelmäßig seit meinem Eintritt in die Forth–Gesellschaft im Jahre 1985 teilnahm. Nach meiner Geburt am 20.03.1954 verbrachte ich meine frühe Jugend in der ländlichen Umgebung eines kleinen 500–Seelen–Dorfes in der Nähe von Vaihingen an der Enz. Nach meinem Schulabschluss erlernte ich in einer kleinen Papierfabrik den Beruf des Chemielaboranten. Besonders interessierte mich aber die organische Chemie, weshalb ich nach meinem Wehrdienst zu Boehringer Mannheim (heute Roche Diagnostics GmbH) wechselte, um dort neue Pharmawirkstoffe zu synthetisieren. Mit diesem Wechsel nach Mannheim ging aber auch ein alter Wunsch in Erfüllung, sich wie ein Vogel in die Lüfte zu erheben, indem ich 1975 in den Badisch–Pfälzischen–Luftsportverein in Mannheim eintrat, um Segelflieger zu werden. Bald beschäftigte ich mich mit der Technik, die man zum Fliegen braucht, wartete Fluginstrumente und bildete mich zum Werkstattleiter weiter. Aber wer seine Sache gut und richtig macht, landet in einem Verein früher oder später im Vorstand, wo ich von 1984 bis 1987 zum Technischen Leiter gewählt wurde. In meiner damaligen Amtszeit modernisierten wir unter anderem die Heizung des Vereinsheims und den zugehörigen Werkstatträumen. Auch damals waren Energiekosten schon ein wichtiges Thema, so entstand der Plan, die Heizung durch einen Mikrocontroller Kosten sparend zu steuern. Mit meinen anfänglichen Erfahrungen von Fig–Forth auf dem AIM–65 ging es nun ans Werk, Forth auf einen 6502–EMUF zu portieren, der die Heizung steuern sollte. Ich erinnere mich noch genau, dass ich damals ein Fig–Forth–Listing bei der Firma Forth–Systeme–Flesch bestellte und der Lieferung ein Mitgliedsantrag der Forth–eV beilag. Im November 1985 war ich dann auf meiner ersten Forth–Tagung in Bad–Bergzabern mit dabei. Viele neue Anregungen — Multitasker, Metacompiler, Volksforth usw. — nahm ich mit nach Hause, sie beschleunigten mein Projekt enorm. Bis 1997 regelte der 6502–EMUF unsere Heizung, ein Hardwareschaden führte sein Ende herbei. Aber auch heute steuert ein EPAC–68k die Heizung unter Bigforth.

Das gesammelte Wissen um Forth — eine Fachsprache zu entwickeln und sie als Scriptsprache für Steuerungssequenzen in nebenläufigen Prozessen einzusetzen — führte mich ab 1987 beruflich zur Entwicklung von präparativen Chromatographieanlagen, um mit deren Hilfe Substanzgemische aus der organischen Synthese im Maßstab von bis zu mehreren hundert Gramm in seine Bestandteile aufzutrennen. Mit dem Aufkommen einer neuartigen Synthese–Strategie der „Kombinatorischen Chemie“ entstand der Wunsch, die Synthese von potentiellen Pharmawirkstoffen zu automatisieren. In einem neuen Projekt entwickelten wir ab 1995 ein eigenes, in Forth programmiertes Robotersystem zur automatischen Synthese. Teilnehmer der Forth–Tagung 1997 in Ludwigshafen konnten damals in die Anfänge dieses Projekts Einblick nehmen. Durch den Verkauf unseres Unternehmens an eine große Aktiengesellschaft mit einer damit verbundenen Strukturbereinigung endete das Projekt im Jahre 2001. Seit meinem Abteilungswechsel wirke ich bis heute an der Entwicklung von Glucosetestsystemen mit. Auch wenn ich heute eher selten beruflich in Forth programmiere, ist mir meine Faszination an Forth erhalten geblieben. Die einfache und unkomplizierte Interaktion mit dem System verleitet mich immer wieder zu neuen Basteleien. Dabei ist die lokale Forth–Gruppe Rhein–Neckar seit ihrer Gründung nach der Forth–Tagung 1986 stets ein gut funktionierendes Team. Besonders unter Mitwirkung von Friedel Amend entstanden dort der „Pflock–Solitär spielende Triceps“ und im letzten Jahr das „Kugel–Balance–Spiel.“ Beide Projekte machten uns viel Freude. Faszinierte Besucher bestaunten unsere Modelle an den Forth–Jahres–Tagungen und ganz besonders bei den Linuxtagen in Karlsruhe und Wiesbaden. Der Lohn: ein erster Kontakt zum Publikum um über Forth ins Gespräch zukommen. Gerade in Wiesbaden zeigte sich, wie leicht die Überleitung dann zum R8C–Projekt gelang.

Als Direktor liegt mir neben den allgemeinen Tätigkeiten deshalb besonders am Herzen, unsere Mitglieder zu ermutigen, anderen Forth näher zubringen. Werben an der richtigen Stelle ist wirksam und heute unverzichtbar. Wir können und wollen sicherlich damit nicht die derzeitige Konstellation unter den Computersprachen verändern, aber wenigstens versuchen, Forth als Bereicherung zu erhalten. Nur wenn es uns gelingt neue aktive Mitglieder zu werben, sind wir auch in Zukunft in der Lage unser

Forth–Magazin „Vierte Dimension“ ausreichend mit lesenswerten Artikeln regelmäßig zu füllen. Ich möchte an dieser Stelle ausdrücklich betonen, dass ich unsere gut gepflegte Web–Präsenz für sehr wichtig halte, sie aber keinen Ersatz für die gedruckte Zeitschrift darstellt.

Uns geht es besonders um den Austausch von Wissen, aber wo soll es denn nun herkommen, wenn nicht von uns selbst. Oft schlummern gute Ideen in den Köpfen, meistens fehlt aber dann der Mut, sie zu äußern, in der Annahme sie seien geringwertig oder in den Augen anderer gar falsch. Deshalb Hemmungen bei Seite legen und ein wenig über eure Projekte und Erfahrungen mit Forth berichten oder aber mit seinen Ideen einfach mitgestalten! Ein Verein kann auf Dauer nicht nur von immer der gleichen Hand voll Aktiven getragen werden. Wir brauchen deshalb immer wieder neue engagierte Mitglieder, um uns zu stärken oder Ausgebrannte zu ersetzen. Aus meinen langjährigen Erfahrungen als Segelflug–Werkstattleiter und Technischer Leiter weiß ich, dass nur gezieltes Ansprechen und Einfordern weiter hilft. In einem beharrlichen Prozess müssen wir ständig Mitglieder mit der richtigen Selbsteinschätzung und etwas Verantwortung finden, um unsere Aufgaben zu meistern.

Ich wünsche uns allen ein erfolgreiches Zusammenwirken und Spaß, gemeinsame Ziele zu erreichen.

Euer Direktor Ewald Rieger

Protokoll der ordentlichen Mitgliederversammlung der Forth–Gesellschaft e.V. am 14.05.2006 im Universitätskolleg Bommerholz in Witten/Ruhrgebiet

Jens Storjohann

Begrüßung der anwesenden Mitglieder

Ulrich Hoffmann begrüßt im Namen des Direktoriums die versammelten Mitglieder.

Wahl des Schriftführers

Zum Schriftführer wird einstimmig Jens Storjohann gewählt.

Wahl des Versammlungsleiters

Heinz Schnitter wird einstimmig (21 ja, keine Enthaltungen, keine Gegenstimmen) zum Versammlungsleiter gewählt. Rolf Schöne berichtet, dass die Gesellschaft zur Zeit 128 Mitglieder hat, von denen 126 den Jahresbeitrag gezahlt haben. Bei (anfänglich) 21 anwesenden Mitgliedern ist damit die Beschlussfähigkeit der Versammlung gegeben.

Ergänzungen zur Tagesordnung

Alle Ergänzungen der Tagesordnung werden unter dem Punkt *Verschiedenes* behandelt.

Verleihung des Swap–Drachens

Ulrich Hoffmann überreicht stellvertretend für Johannes Reilhofer, der leider verhindert ist, den Swap–Drachen an Carsten Strotmann in Anerkennung seiner Arbeit um die Verbreitung von Forth. Er hat das Volksforth wiederbelebt, Forth im Rahmen des Vintage Computer Festivals und auf Linux–Veranstaltungen verbreitet.

Bericht des Direktoriums (mit Redaktion und Kassenbericht)

Rund um die VD

Ulrich Hoffmann berichtet über die VD. Er dankt unserem zurückgetretenen Redakteur Friederich Prinz für seine Arbeit. Provisorisch wurde die Redaktionsarbeit vom Direktorium übernommen.

Mitglieder–Entwicklung, Kassenstand und Kassenprüfung

Rolf Schöne berichtet über die Finanzen und die Entwicklung der Mitgliederzahlen. Das Vereinsvermögen hat sich um 1745,55 Euro vom 06.04.2005 bis 08.05.2006 leicht erhöht. Das liegt u. a. daran, dass im letzten Jahr nur 3 Hefte der VD erschienen sind. Im Jahr 2005 sind 6 Mitglieder ausgeschieden (5 Kündigungen, 1 Nichtzahlung des Beitrags). Neue Mitglieder sind 2005 3 und 2006 4 dazugekommen. Damit kann man vorläufig von einem Ende des Mitgliederschwundes sprechen.

Der durch Rolf Lauer erstellte Kassenprüferbericht wird durch Heinz Schnitter verlesen. Er enthält die uneingeschränkte Bestätigung *die Kasse stimmt*.

Einige Teilnehmer der Forth–Tagung 2006 mit einer schwer zu fotografierenden halben Birgit Prinz

Internet–Präsenz

Ulrich Hoffmann trägt über die Internet–Präsenz der Forth–Gesellschaft vor. Ein neuer root server ist in Betrieb genommen worden. Die Arbeit an der Internet Präsenz der Forth–Gesellschaft verteilt sich nun auf mehrere Schultern. Es ist nun ein System auf technisch höchstem Niveau implementiert, das viel mehr als das Verwalten statischer Internet–Seiten erlaubt. Beispielsweise wird die VD über Subversion Repository erstellt.

Der Mikrocomputer Verleih läuft auch über die Website.

Auslandsarbeit

Fred Behringer berichtet über die Aktivitäten der Forth–Anhänger im Ausland. In der Silicon Valley Gruppe treffen sich regelmäßig 10 bis 30 Teilnehmer. In der VD lesen wir oft die Berichte, die uns Henry Vinerts übermittelt. Es gibt keine Zeitschrift mehr. Die Gruppe in Großbritannien konnte ihre Zeitschrift auch nicht weiterführen. Die Verbindung zur niederländischen Gruppe ist sehr gut. Willem Ouwerkerk und Albert Nijhof sind als Gäste zur Jahrestagung gekommen. Wenn die holländische Gruppe 180 Mitglieder zählt, liegt dies an der Struktur der HCC als Dachorganisation.

Arbeitsbericht und Vorhaben des Direktoriums

Bernd Paysan berichtet über Projekte. Das R8C–Forth wurde auf dem Linux–Tag gezeigt. Es handelt sich um ein von der Zeitschrift Elektor begonnenes Projekt einer Mikroprozessor–Platine, die über den Verlag erhältlich ist. Über die Forth–Gesellschaft gibt es ein Forth System dafür. An der USB–Verbindung hierfür wird noch gearbeitet.

Die Bücher von Leo Brodie *Starting Forth* und *Thinking Forth* sollen wieder zugänglich werden. Das erste steht schon im Netz. Im Augenblick wird daran gearbeitet, die Beispiele des zweiten Buches auf den ANSI–Standard umzustellen.

Auf dem Linux–Tag in Wiesbaden war der Stand der Forth–Gesellschaft gut besucht. Ein echter Hingucker war die von Ewald Rieger gezeigte Anordnung, die mechanische Bewegungen von rollenden Kugeln über Bildverarbeitung regelte.

Entlastung des Direktoriums

Eine Teilsumme der Ausgaben (1421 Euro) für die letzte Jahrestagung wurde als kategorielos gebucht. Das Direktorium wird eine formale Korektur vornehmen lassen, wodurch die Ausgeglichenheit von Einnahmen und Ausgaben der Jahrestagung sichtbar wird.

Die Entlastung des Direktoriums wurde einstimmig mit 24 Ja–Stimmen (0 Enthaltungen, 0 Gegenstimmen) beschlossen.

Wahl des Direktoriums

Ulrich Hoffmann, Bernd Paysan und Ewald Rieger stellen sich als Gruppe zur Wahl. Sie werden einstimmig (24 Ja–Stimmen, 0 Enthaltungen und 0 Gegenstimmen) gewählt.

Prozessor–Projekte (R8C etc.)

Die Mitglieder zeigen Interesse am MicroCore von Lattice und am R8C. Der R8C soll einen USB Controller von Cypress bekommen. Carsten Strotmann wird sich um Treiber und Dokumentation kümmern.

Verschiedenes

Der Ort der nächsten Jahrestagung und Mitgliederversammlung wird diskutiert. Es stehen zur Wahl Wien (Anton Ertl), Ulm (Gerd Franzkowiak) und Lübeck (Jens Storjohann).

Forth von der Pike auf — Teil 5 und 6

Ron Minke

Die hier mit freundlicher Genehmigung der HCC–Forth–gebruikersgroep wiederzugebende achtteilige Artikelserie erschien in den Jahren 2004 und 2005 in der Zeitschrift *Vijgeblaadje* unserer niederländischen Forth–Freunde. Übersetzung: Fred Behringer.

Hier sind nun der fünfte und sechste Teil des Versuchs, ein Forth–System auf die Beine zu stellen, dessen Voraussetzung überhaupt nix, oder auf gut Deutsch *from scratch*, lautet.

Wir kommen jetzt zum fünften Teil der Geschichte über das Hochziehen eines AVR–Forth–Systems mit dem Ausgangspunkt from scratch.

In der vorigen Folge hatten wir IP und SP je zu je den AVR–Registerpaaren X und Y zugeordnet. Im vorliegenden Teil gehen wir auf den verwendeten Pseudocode zur Umsetzung von High–Level–Worten nach Befehlen in AVR–Maschinensprache ein. Werfen wir nochmal einen Blick auf Teil 4.

Legen wir gleich mit dem Code los:

DOCOL–Pseudo		DOCOL–Assembler	
PUSH IP	(1)	`Push XL`	Bewahre IP auf dem Returnstack auf
	(2)	`Push XH`	
INC IP	(3)	`Adiw WL,2`	Zeig auf das nächste Wort
MOV IP,W	(4)	`Mov XL,WL`	Kopiere Pointer
	(5)	`Mov XH,WH`	
NEXT			Führe den Code für NEXT aus (siehe Teil 4); Fahre mit dem nächsten Wort fort

Zur Aufbewahrung von IP auf dem Returnstack benötigen wir zwei Befehle: Unser Prozessor ist ja nur acht Bit breit. Aus demselben Grund erhöhen wir W um zwei, damit es auf das nächste Wort (= 2 Bytes weiter oben) zeigt.

EXIT–Pseudo		EXIT–Assembler	
POP IP	(1)	`Pop XH`	Stelle IP vom Returnstack aus wieder her
	(2)	`Pop XL`	
NEXT			Führe den Code für NEXT aus (siehe Teil 4); Weiter dort, wo wir verblieben waren

Auch hier zwei Befehle, um IP wiederherzustellen. Man beachte, dass es nicht nötig ist, W wiederherzustellen. W bekommt in NEXT einen neuen Wert.

Die richtige Wahl getroffen?

Wir müssen uns noch eine Antwort auf die Frage verschaffen, ob die Wahl des AVR–Registerpaares W für das Forth–Register W die **richtige** Wahl gewesen ist. Das W–Register wird zur Markierung des Speicherplatzes für das nächste Wort verwendet.

In diesem AVR–Forth verwenden wir W ausschließlich in High–Level–Definitionen zur Anzeige des nächsten Wortes. Dazu muss W innerhalb des Codes für `NEXT` (siehe Teil 4) einen Wert bekommen und dieser Wert muss beim Verarbeiten von `DOCOL` zur Verfügung stehen. Zur Verfügung stehen muss er, wie wir später sehen werden, auch in `DOCOL`–artigen Konstruktionen wie `DOCON` und `DOVARIABLE`. Außerhalb dieser in Maschinensprache gehaltenen Teile wird weder das Forth–Register W noch das AVR–Registerpaar W benötigt. Wir können also das Registerpaar W als ein ganz *gewöhnliches* Registerpaar verwenden. In den Maschinensprachteilen machen wir von der Möglichkeit Gebrauch, dass wir bei Registerpaaren auf einen Schlag gleich noch einen konstanten Wert (hier 2, die Anzahl von Bytes in einem Wort) hinzuaddieren können, um das nächste Wort zu erreichen. Das brauchen wir also nicht in zwei getrennten Byte–Additionen zu tun. Der verwendete Befehl, `Adiw`, ist auf die Registerpaare W, X, Y und Z und nur auf diese anwendbar. Da wir die Registerpaare X, Y und Z bereits zugeordnet haben, ist das W–Registerpaar das letzte Paar, bei welchem dieser `Adiw`–Befehl möglich ist. (Befehlssatz: Siehe Atmel–Website.)

Die Wahl des AVR–Registerpaares W für das Forth–Register W ist sicher eine gute Wahl.

Entscheidung 8: Das Forth–Register W wird dem AVR–Registerpaar W zugeordnet.

So, die Basis ist damit gelegt. Wir zählen die getroffenen Entscheidungen noch einmal auf:

Forth–Register	AVR–Register
RP	SP
W	WL und WH (= R24 und R25)
IP	XL und XH (= R26 und R27)
SP	YL und YH (= R28 und R29)
Hilfsregister für indirekte Sprünge	ZL und ZH (= R30 und R31)

Struktur hineinbringen

Um dem Ganzen etwas Struktur zu verleihen, treffen wir folgende Vereinbarung über die Verwendung von Registern innerhalb des Datenstacks. Alle Datenstack–Aktionen ordnen wir ab R23 (gleich unter dem Registerpaar W) nach unten zu an, wobei R23 das obere Byte eines Wortes enthält und R22 das untere und so weiter. Benötigt ein Wort beispielsweise zwei Stack–Einträge, dann kommen wir zu folgendem Bild (wir nehmen das Wort `AND` als Beispiel):

```
AND ( n2 n1 -- n3 )
```

	Input		Output	AVR–Register
	0		0	R19
	1 n2 unteres Byte		1 n3 unteres Byte	R20
	2 n2 oberes Byte	SP →	2 n3 oberes Byte	R21
	3 n1 unteres Byte		3	R22
SP →	4 n1 oberes Byte		4	R23
	5		5	

Der Maschinencode wird nun:

```
Code_Andd:
   Ld R23,Y+      Hole oberen n1–Teil herein
   Ld R22,Y+      Hole unteren n1–Teil herein
   Ld R21,Y+      Hole oberen n2–Teil herein
   Ld R20,Y+      Hole unteren n2–Teil herein

   And R21,R23    AND oberer Teil
   And R20,R22    AND unterer Teil

   St -Y,R20      Speichere unteren n3–Teil ab
   St -Y,R21      Speichere oberen n3–Teil ab

   Next
```

Benötigen wir einen Vorgang, bei dem Daten hereingeholt werden müssen (indirekt, also über einen Pointer), dann verwenden wir dafür das Registerpaar Z. Eine Zuordnung dieses einen Stackeintrags zu einem Registerpaar aus der Reihe R23 ... entfällt dann.

Nehmen wir als Beispiel den Code für das Wort `@`

```
@ ( adresse -- wert )

Code_At:
   Ld ZH,Y+       Hole oberen Adressteil herein
   Ld ZL,Y+       Hole unteren Adressteil herein

   Ld R21,Z+      Hole oberen Teil des Wertes
                  von dieser Adresse herein
   Ld R20,Z+      Hole unteren Teil des Wertes
                  von dieser Adresse herein

   St -Y,R20      Speichere unteren Teil des
                  Wertes ab
   St -Y,R21      Speichere oberen Teil des Wertes ab

   Next
```

Wir reservieren vorläufig R23 ... R12 für sechs Stack–Einträge. Im Übrigen sind Forth–Worte, die mehr als sechs Stack–Einträge benötigen, nicht mehr einfach zu nennen!

Inzwischen sind wir einem guten Teil von AVR–Maschinencode begegnet. Bevor wir jedoch zu einem funktionierenden Forth–System kommen, müssen wir auch die Hardware–Seite noch unter die Lupe nehmen.

Hardware–Umsetzung

Der AVR–Prozessor von Atmel ist ein Prozessor, der mit der Harvard–Speichereinteilung arbeitet. Das bedeutet, dass der Speicherplatz für das Programm vom Speicherplatz für die Daten ganz und gar getrennt ist (man vergleiche die Datenblätter der verschiedenen Prozessoren auf der Website von Atmel). Das stellt an den Entwurf unseres Forth–Systems spezielle Anforderungen. Die Codierungsmöglichkeiten für Forth sind:

- Indirekte Fädelung (klassisches Modell)
- Direkte Fädelung
- Unterprogramm–Fädelung

Die **einzig** mögliche Art der Implementation auf dem AVR–Prozessor ist das klassische Forth–Modell, die indirekt gefädelte (indirect threaded) Version.

Warum ist das so? Die indirekt gefädelte Version geht von einer Wortliste aus, die ausschließlich Zeiger (Pointer) enthält, welche auf einen Platz (die CFA) verweisen, der wiederum einen Zeiger auf den Maschinencode (im internen Flash–Speicher) enthält. Neue Worte, die an die Wortliste angefügt werden, bestehen ausschließlich aus Zeigern. Nirgends wird direkt auf Maschinencode verwiesen. Sehen Sie sich den in Teil 4 gebrachten Codeteil für `NEXT` daraufhin noch einmal an. Die Werte, welche im Datenspeicher landen, sind **echte** Daten (die Pointer). Das Ablegen von Daten und das anschließende Ausführen dieser Daten so, als wäre es Maschinencode, ist **nicht** möglich; Maschinencode kann einzig und allein vom internen Flash–Speicher aus ausgeführt werden.

Im folgenden wird auf die Pointer–Struktur der drei Fädelungstypen näher eingegangen. Bei einem Prozessor der Serie 8051, das werden Sie danach begreifen, sind alle drei Typen möglich, indem man nämlich die beiden Speicherbereiche gewissermaßen *aufeinander* legt. Bei der AVR–Prozessorserie geht das ganz bestimmt nicht und wir bleiben an das indirekte Verdrahtungsmodell gebunden. (Das finden wir aber auch nicht sonderlich schlimm. Das ist das wahre *Basismodell*.)

Wir sehen uns nun (im ursprünglichen Teil 6 dieser Artikelserie) die Hardware–Ausstattung des Systems an. Das System hat eine Standard–Ausführung und besteht aus einem AVR–Prozessor, einem Adress–Latch und einem RAM–Speicher. Man beachte, dass das übliche EPROM fehlt. Das sitzt als Flash–Speicher im Prozessor selbst. Und in diesem Flash–Speicher sitzt nun wiederum unser Vorhaben, das AVR–Forth. Wir gehen davon aus, dass ein serieller Anschluss in Form eines im Prozessor–Chip vorhandenen UARTs zur Verfügung steht. Auch müssen wir natürlich einen AVR–Prozessortyp wählen, der externes RAM ansteuern kann (das können sie durchaus nicht alle).

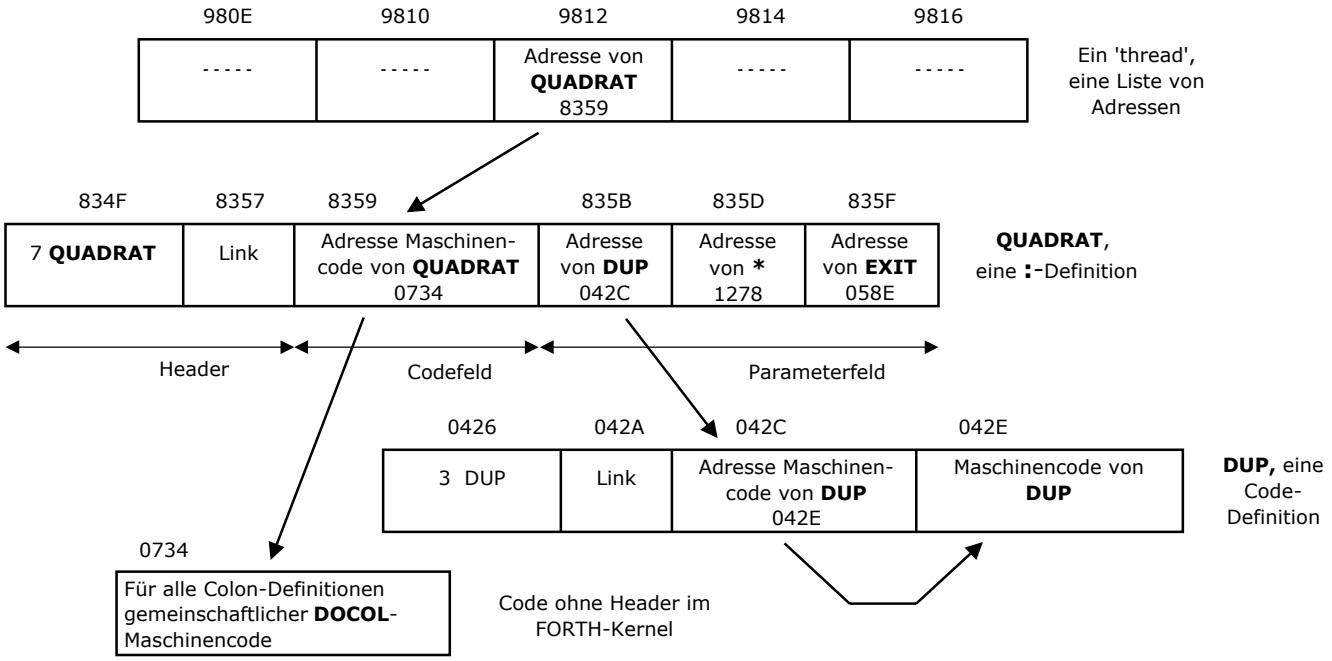


Abbildung 1: Indirekte Fädelung

Indirekt gefädelt, direkt gefädelt, unterprogrammgefädelt???

Bevor wir die verschiedenen Fädelungsarten (threads) besprechen, brauchen wir noch eine Methode, :–Definitionen in Forth ins RAM zu setzen. Wie wir das tun, ist im Augenblick nicht so wichtig. Wir gehen davon aus, dass es ganz *normal* möglich ist. Ausgangspunkt ist die Definition

```
:    QUADRAT   DUP   *   ;
```

Irgendwo in unserem Forth–System verwenden wir das Wort `QUADRAT`. Um das zu Sagende besser ins Bild zu bringen, nehmen wir ein paar fiktive Speicheradressen an, wo die verwendeten Worte abgelegt sind. Den **Aufruf** von `QUADRAT` finden wir beispielsweise an der Hex–Adresse 9812. Die **Definition** von `QUADRAT` finden wir an Adresse 834F. Das Wort `DUP` ist ein Low–Level–Codewort an Adresse 0426 im Flash–Speicher und `*` ist eine High–Level– (eine :–) Definition an Adresse 1278 im Flash–Speicher. Für ein traditionelles indirekt gefädeltes Forth können wir jetzt den folgenden Speicherauszug zeichnen. Abbildung 1 zeigt ein Beispiel bei dem je zwei Bytes eine Zelle bilden.

Wir nehmen an, dass der IP jetzt auf die Stelle 9812 zeigt, so dass also bei Aufruf von `NEXT` die Definition `QUADRAT` ausgeführt wird.

Was geschieht nun genau? Wir verwenden den einfachen Befehlssatz, den wir in Teil 1 eingeführt hatten (man achte auf die Zahlenbeispiele):

`NEXT:`	IP zeigt auf das als nächstes auszuführende Wort an Adresse 9812
`MOV W,(IP)`	Kopiere den Inhalt von IP (=8359), die CFA des als nächstes auszuführenden Wortes, ins Register W.
`INC IP`	Setze den IP so, dass er auf das Wort NACH dem momentan bearbeiteten zeigt, um da unmittelbar weiterzuarbeiten (=9814).
`MOV Z,(W)`	Führe den Maschinencode aus, dessen Adresse jetzt im W–Register (=0734) sitzt. Verwende Z als Zwischenregister.
`JMP (Z)`	Gehe über einen indirekten Sprung in den Maschinencode.

Wäre `QUADRAT` eine Maschinencode–Definition gewesen, wären wir nun fertig. Das Stückchen Maschinencode wird ausgeführt und wir springen auf das nächste `NEXT` zurück, das uns zu Platz 9814 leitet. `QUADRAT` ist jedoch ein High–Level–Wort. Es enthält keinen Maschinencode, sondern einen thread, eine Liste von Adressen. Um diese Definition ausführen zu können, muss der Interpreter an Adresse 835B, dem Parameterfeld von `QUADRAT`, aufs Neue gestartet werden. Wir müssen aber auch den alten Wert von IP sichern, um da dann fortfahren zu können, wenn das Wort `QUADRAT` abgearbeitet ist. Da wir gerade einen indirekten Sprung ausgeführt haben, müssen wir hierzu auf ein Stückchen Maschinencode stoßen. Das ist der Code für `DOCOL` (sehen Sie noch einmal kurz bei Teil 5 dieser Serie nach). Dieses Stückchen Maschinencode ist

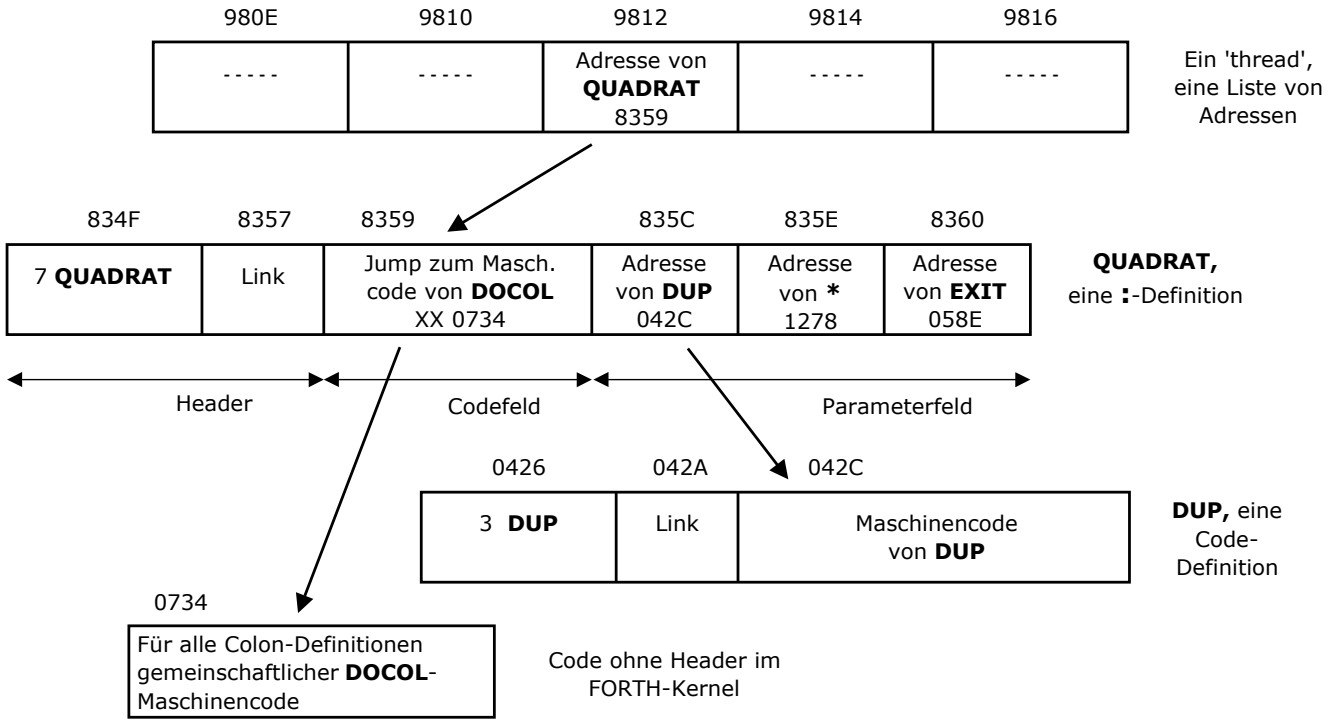


Abbildung 2: Direkte Fädelung

für jede High–Level–Definition dasselbe. Wir wiederholen den Pseudocode:

```
DOCOL:
    PUSH IP     Sichere den momentanen IP
                auf dem Returnstack (=9814)
    INC W       Zeige auf das nächste Wort
                (=835B)
    MOV IP,W    Kopiere diesen Pointer als neu-
                en IP

    NEXT        Fahre beim nächsten Wort fort
```

Wir sind nun bei der Ausführung der Definition `QUADRAT` eine Ebene tiefer gerutscht. Die Situation, in der wir uns jetzt befinden, ist eigentlich dieselbe wie an unserem Ausgangspunkt, das Ausführen einer Reihe von Definitionen (Worten). Wir sind auf indirekte Weise hierher gelangt, über Zeiger (Pointer). Diese Vorgehensweise ist die traditionelle altmodische Art von Forth. Das ist die indirekt gefädelte Methode.

Es ist klar, dass wir auch noch zurück müssen. Haben wir das Wort `*` auf dieselbe Weise ausgeführt, so stoßen wir auf das Wort `EXIT`. Dieses Wort ist der Returnbefehl, der in den Speicher gesetzt wird, wenn der Forth–Compiler auf das Wort `;` (Ende der Definition) trifft. `EXIT` macht das Umgekehrte von `DOCOL`. In Pseudocode sieht das so aus:

```
EXIT:       Die Rückkehradresse steht auf
            dem Returnstack

    POP IP  Stell die Adresse des als nächs-
            tes auszuführenden Wortes
            wieder her (=9814)
    NEXT    Fahre dort fort, wo wir nach
            dem Ausführen des Wortes
            QUADRAT verblieben waren
```

Die charakteristischen Merkmale eines indirekt gefädelten Forths: Jedes Forth–Wort hat ein Codefeld von genau einer Zelle (hier 2 Bytes). High–Level– (:–) Definitionen compilieren für jedes Wort genau eine Zelle in die Definition. Der Forth–Interpreter muss doppelt indirekt arbeiten, um die Adresse des schließlich auszuführenden Maschinencodes zu finden, erst über IP, danach dann über W.

Direkt gefädelter Code

Der Unterschied ist klein: Beim direkt gefädelten Code steht im Codefeld keine Adresse, sondern Maschinencode. Oft hat der Code die Form von `JMP` adresse oder `JSR` adresse, aber es kann natürlich auch eine vollständig ausgeschriebene Routine sein, die auf `NEXT` endet (siehe Abbildung 2).

Da `NEXT` nun eine Indirektionsstufe weniger auszuführen braucht, wird es ein Stückchen einfacher:

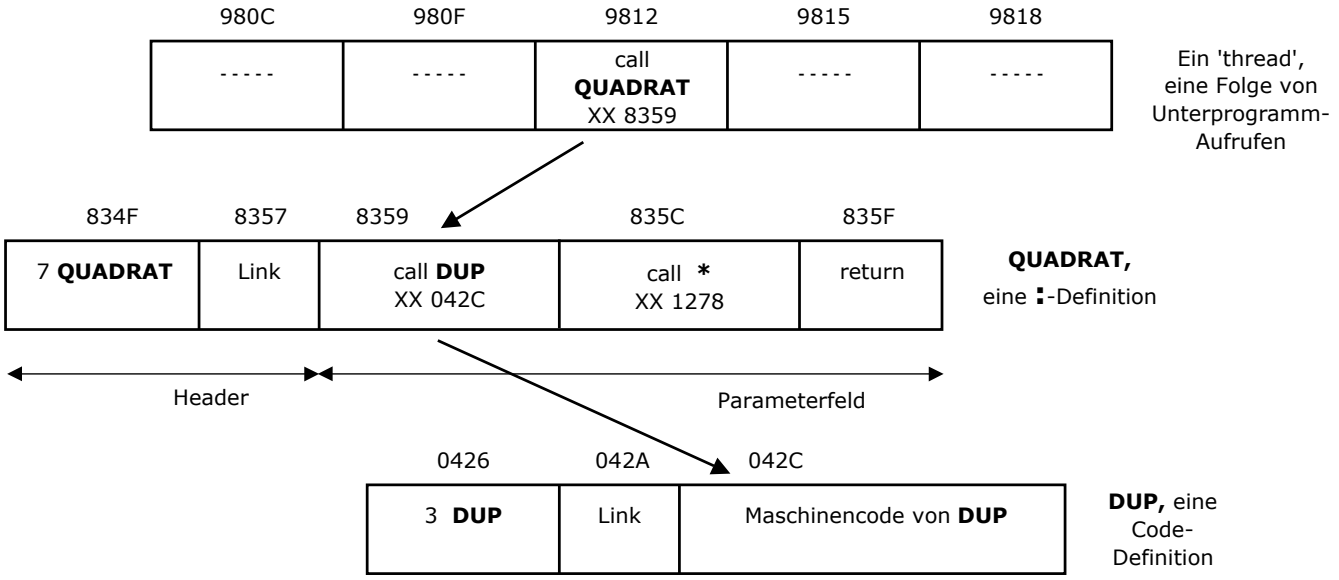


Abbildung 3: Unterprogramm–Fädelung

`NEXT:`	IP zeigt auf das als nächstes auszuführende Wort auf Adresse 9812
`MOV W,(IP)`	Kopiere den Inhalt von IP (=8359), die CFA des als nächstes auszuführenden Wortes, ins W–Reg.
`INC IP`	Setze den IP so, dass er auf das Wort nach dem momentan bearbeiteten zeigt (=9814), um da unmittelbar weiterarbeiten zu können.
`JMP (W)`	Springe direkt zum Maschinencode (=0734)

Der letzte Befehl in diesem Beispiel ist ein `JMP`. Und hier liegt zugleich das Problem unseres AVR–Forth–Systems: Der Prozessor wurde so entworfen, dass im RAM–Speicher nichts anderes als Daten liegen können. Der auszuführende Maschinencode findet sich einzig und allein im FLASH–Programmspeicherbereich des Prozessors. Und sollten Sie die Maschinencodebefehle ins RAM setzen, dann könnten diese auf keine einzige Art ausgeführt werden. Der Trick des Aufeinanderlegens der beiden Speicherplatzarten (Programmbereich und Datenbereich), so wie er bei der Prozessorserie 8052 üblich ist, greift hier nicht. Schade, mit dem AVR–Prozessor ist keine direkt gefädelte Forth–Version möglich!

Unterprogrammgefädelter Code

Eigentlich können wir es nun schon erraten: Ein unterprogrammgefädeltes Forth besteht aus einer Anzahl von hintereinander platzierten Unterprogrammaufrufen (Calls) der verwendeten Worte.

Aber wenn wir die Calls im RAM unterbringen, können wir auch hier leider keinen Maschinencode ausführen. Abbildung 3 zeigt der Deutlichkeit halber einen Speicherauszug.

Man beachte, dass sich durch die Verwendung von Unterprogrammaufrufen die Adressen, an denen die Calls stehen, verändert haben! Der `CALL`–Befehl selbst nimmt einen gewissen Platz ein.

Unsere Wahl ist einfach:

Entscheidung 9: Wir wählen ein indirekt gefädeltes AVR–Forth!

— Wird fortgesetzt —

Gehaltvolles

zusammengestellt und übertragen von *Fred Behringer*

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 55, April 2006

Gespot op het web — Ron Minke

„Inzwischen hat jeder DSL–Anschluss und es ist möglich geworden, im Internet nach Herzenslust nach Forth–Anwendungen Umschau zu halten. Da wir in der Forth–groep viel mit AVR–Prozessoren von Atmel arbeiten, informiere ich mich regelmäßig bei `http://www.avrfreaks.com`. In einem der Forumsdiskussionen las ich etwas über ein Webserver–Projekt. Erstaunlich, mit wie einfachen und wenigen Mitteln man sowas in Forth bewerkstelligen kann.“ Es folgen Hinweise auf verschiedene Prozessoren, mit denen Webserver verwirklicht wurden: 8052 — RTL8019 — Anpassung einer alten ISA–Netzwerkkarte — AVR–Mega–Prozessor — ...

Gespot op het web II — Ron Minke

„Auf der oben genannten Website der AVR–Freaks fand ich noch etwas, was für die Forth–groep interessant sein könnte: Es betrifft das Programmieren von AVR–Prozessoren über die ISP–Anschlüsse. Für die Forth–groep kann man dazu die Dongle–Karte von Willem (Rez.: Ouwerkerk?) verwenden.“ ... Kabel vom Dongle zur Prozessorkarte so kurz wie möglich halten, um Übersprechen zu vermeiden...

Een nieuwe ster — Albert v.d. Horst/Coos Haak

Mit Alberts ciForth vor Augen wird auf das R8C–Platinchen eingegangen, das auch von der Zeitschrift Elektuur, der niederländischen Ausgabe der Elektor, angeboten wurde (12,– Euro): 3 Maschinenbefehle für ROT, 1 Befehl für M*, für FILL und MOVE Register poppen und dann 1 Maschinenbefehl, ein einziger Maschinenbefehl, um 32 Bits durch 16 Bits zu teilen... Datenstack, Returnstack, User–Variablen und Terminal–Buffer je 128 bytes an RAM, bleiben 512 Bytes für neue Definitionen, der Forth–Kernel natürlich im Flash–ROM. „Übrigens gibt es auch schon ein GForth für das Ding. Es ist uns gelungen, die benötigten Dateien aus dem Internet zu laden und das Projekt zu bauen. Wir wissen jedoch noch nicht genau, wie das angefertigte Image in den Prozessor zu laden ist.“

Das Vijgeblaadje erscheint um den Ersten eines jeden geraden Monats herum.

VIJGEBLAADJE der HCC Forth–gebruikersgroep, Niederlande Nr. 56, Juni 2006

Lichtkrant deel 2 — Paul Wiegmans

Das ist der zweite und letzte Teil von Pauls Laufschrift–Artikel. Pauls Einleitung: „Diesmal geht es über den Pixelbuffer im Speicher und die Abbildung von Zeichen aus dem Pixelbuffer heraus. Anhand von einigen Stückchen Forth lass ich sehen, wie Text auf dem Laufschrift–Gerät sichtbar gemacht wird. Das Ganze wird von einem SAB80C535–Prozessor auf einem ATS–Board ausgeführt.“ Die Überschriften der Artikelabschnitte: Der Pixelbuffer — Die Font–Tabelle — Das Abbilden — Das Hauptprogramm.

Glyn evaluation board — Albert v. d. Horst

Albert berichtet: „Seit der Ausstellung von Bits&Chips habe ich guten Kontakt zur Firma Glyn. Sie haben für 50,– Euro ein Evaluation–Board, das sie zu einem guten Preis an uns verkaufen würden. Sie versprechen sich von einer Werbeaktion Nutzen in Hobby–Kreisen, so ähnlich wie bei ihrem Elektor–Bravourstück mit dem R8C–Brettchen. Das Evaluationsboard enthält einen M30626FJPFP–Chip.“ Und dann zählt Albert die Eigenschaften des Chips auf. Näheres bei `www.glyn.nl`. Es ist von Forth die Rede (Rezensent: Wessen Forth?).

Hard– en software voor het ATS Board — Robert Henneke

Der Autor bezieht sich auf das Handbuch Forth auf dem 80C35–Prozessor mit dem ATS535 von At van Wijk. Er bespricht jeweils ganz kurz: 32K–IIC–EEPROM — 32KB–Data–RAM auf dem Speichererweiterungsanschluss — eine alternative Reset–Betätigung – Ein Real–Time–Clock — Monitor — Disassembler/Tracer — Downloader — I/O–Driver — RTC und Multitasker.

Adventures in Forth

Erich Wälde

Bevor ich vergesse, wie es am Anfang war, schreib ich mal ein bisschen auf, wie es mir ergangen ist, seit ich meine Nase in Forth gesteckt habe.

Am Linuxtag 2006 in Wiesbaden hatte ich den Renesas r8c/13–Mikrokontroller in Forth–Aktion gesehen. Den Kontroller auf dem Elektor–Platinchen hatte ich noch zu Hause, und so hatte ich endlich eine sinnvolle Verwendung dafür. Allerdings habe ich das Entwicklungsboard (von Elektor, 1) und ein 16x2–LCDisplay dazugekauft.

Installation

Das Wiki von Bernd Paysan (2) führte mich zu den nötigen Zutaten

1. `gforth`
2. `libffcall`
3. `m16c_flasher` von Thomas Fischl

Eine etwas ältere Version von gforth war auf meinem Rechner schon installiert:

```
$ dpkg -l gforth
ii  gforth         0.6.2-6        GNU Forth
Language Environment
```

Die Installation der libffcall war zwar einfach, aber wie gforth beim Installieren davon Wind bekommt, das war nicht ganz offensichtlich, weil ich diese Dinge in meinem HOME–Verzeichnis installiert hatte. Mit ein paar zusätzlichen Umgebungsvariablen lässt sich das aber lösen ('$' ist der shell prompt):

```
$ wget ftp://ftp.ilog.fr/pub/Users/haible/gnu/
ffcall-1.8.tar.gz
$ tar xvf ffcall-1.8.tar.gz
$ export CC=gcc-3.4
$ cd ffcall-1.8
$ ./configure --prefix=$HOME
$ make
$ make install
$ cd ..
```

```
$ cvs -d :pserver:anonymous@c1.complang.tuwien.
ac.at:/nfs/unsafe/cvs-repository/src-master co
gforth
$ export CFLAGS=-I$HOME/include
$ export LDFLAGS=-L$HOME/lib
$ cd gforth
$ export GFORTH=/usr/bin/gforth
$ export GFORTHPATH=$PWD:/usr/lib/gforth/0.6.2
$ ./BUILD-FROM-SCRATCH --prefix=$HOME
$ ./build-ec r8c
$ cd ..
```

```
$ wget http://www.fischl.de/thomas/elektronik/
r8c/m16c_r8c_flash.2005-11-20.tar.gz
$ tar xzvf  ~/Forth/m16c_r8c_flash.2005-11-20.
tar.gz
$ cd m16c_flash
$ make clean
$ make
$ cd ..
```

`BUILD-FROM-SCRATCH` blieb bei mir irgendwann hängen, aber `gforth` wurde davor erzeugt, so dass `./build-ec r8c` erfolgreich läuft.

Das frisch gebackene `rom-r8c.mot` muss jetzt auf den Kontroller. Dazu verbindet man den Kontroller entweder direkt mit der seriellen Schnittstelle (s. Elektor 01/2006) oder über das dazugehörige Entwicklungsboard. Eine Stromversorgung und das Ansteuern des MODE–Pins zum Programmieren gehört dazu, Details finden sich in den erwähnten Artikeln. Angenommen, die serielle Schnittstelle des Entwicklungsrechner wäre `/dev/ttyUSB0`, dann geht das Programmieren des Kontrollers so:

```
<Mode-Pin-erden>
<Reset-Taste-drücken>
$ ./m16c_flash/flash /dev/ttyUSB0 R8C ./gforth/
rom-r8c.mot ff:ff:ff:ff:ff:ff:ff
<Mode-Pin-befreien>
<Reset-Taste-drücken>
```

über die gerade eben schon verwendete serielle Schnittstelle kann man sich jetzt mit dem Kontroller verbinden, z.B. mit minicom. Die Einstellungen sind dann `/dev/ttyUSB0`, 38400 baud, 8 Datenbits, kein Paritätsbit, 1 Stopbit (8N1). Wenn alles funktioniert, erhält man auf Drücken der Return–Taste `<RET>` die Antwort „`ok.`“

```
minicom
... start message ...
<RET>
ok.
```

Siehe auch den Diskussionsthread unter 3 für Hinweise.

Erste Versuche

Für die ersten Versuche (blinken mit den LEDs, etwas auf das Display schreiben) finden sich Beispiele im Wiki (2).

Über Forth selbst sind die Dokumentation von gforth sowie die Bücher “Starting Forth” und “Thinking Forth” (Leo Brodie) sehr gute Quellen. Zu beiden Büchern gibt es Webseiten (4, 5). Auch der Wikipedia–Artikel zu Forth (7) ist meiner Meinung nach recht gut.

`rom` und `ram`

aus dem Wiki:

> Das Forth ... legt selbstdefinierte Programme per Default im (sehr knappen) RAM ab. Umschalten zwischen RAM und ROM mit den Wörtern ram und rom.

ROM ist das Flashrom des Kontrollers, es ist größer als das RAM. Daher kann man Forth–Programme ins ROM speichern:

```
rom
... hier kommt das programm hin
ram
savesystem
```

`savesystem` speichert eine Kopie des RAM–Inhalts ins ROM. Wenn das Programm im ROM überschrieben werden soll, dann muss man das ROM erst mit dem Wort `empty` löschen. Wenn man das versäumt, oder wenn es nicht richtig funktioniert hat, dann erhält man beim nächsten Befehl etwas irreführende Fehlermeldungen, etwa so:

```
: foo <<< Undefined word
```

`empty` und evtl. ein RESET hilft dann normalerweise. Den Erfolg kann man auch überprüfen mit dem Wort `dump`, welches den Speicherinhalt ausliest:

```
rom
... ein paar Befehle hier
ram
savesystem

$2000 $800 dump
...
empty
$2000 $800 dump
...
```

Nach dem `empty` stehen lauter `FF` im Speicher, vorher kann man mit etwas Phantasie das Programm wiederfinden.

Das Dollarzeichen '$' sagt Forth, dass die Zahl dahinter auf jeden Fall eine Hexadezimalzahl ist, egal welche Basis zur Eingabe gerade benutzt wird, z.B. $10 = 16. Entsprechend sagt das Prozentzeichen '%', dass die Zahl dahinter eine Binärzahl ist, z.B. %10001010 = $8a = 138. Auch für Dezimalzahlen gibt es ein Zeichen: '&', welches in den Beispielen im Wiki benutzt wird.

Ein i2c–Device ansteuern

Als erstes Bastelprojekt, bei dem sich richtig was tut, wollte ich einen i2c–Baustein ansteuern, welcher 8 digitale IO–Pins besitzt (PCF8574). Der Bus benutzt 2 Drähte, SDA und SCL genannt. Die Details stehen im Datenblatt zum PCF8574 oder beispielsweise in dem Buch "I2C Bus angewandt" (6) Der R8C–Kontroller soll als "i2c master" funktionieren, d.h., nur er darf den Bus ungefragt belegen und er gibt bei der Datenübertragung den Takt vor.

Als Erstes werden ein paar Konstanten definiert. Deren Namen sind aussagekräftiger als die blanken Werte. Sie machen das Programm besser lesbar. Pin 15 alias P1.0 des Kontrollers gehört zum ersten Bit von Port 1 und soll die SCL Leitung treiben:

```
0 Constant PinSCL
```

Pin 14 alias P1.1, also das zweite Bit von Port 1 soll die SDA Leitung treiben:

```
1 Constant PinSDA
```

Für den Port 1 gibt es schon ein Wort (`port1`), ansonsten hört der auch auf das Register mit der Adresse $E1

```
port1 Constant PortI2C
```

Da ich nicht nur auf den Bus schreiben will, sondern auch davon lesen, muss ich gelegentlich das „Data Direction Register" von Port 1 beschreiben. Deshalb bekommt es ebenfalls einen Namen statt seiner Registeradresse ($E3)

```
$E3 Constant PddrI2C
```

Die 7–Bit–Adresse des Portexpanders ist %010_0000 oder $20 hexadezimal. Das Schreiben/Lesen–Bit–w wird hintendran gehängt: %010_0000_w. Das ist $20 um eins nach links verschoben + w, also $40 + w. Zum Lesen vom Baustein wird diese Adresse um 1 erhöht ($41).

```
$40 Constant i2c_addr_portexpander
```

Die einfachsten Funktionen (Worte) setzen je einen der beiden Pins auf 0 oder 1:

```
: sda0 PinSDA PortI2C bclr ;
: sda1 PinSDA PortI2C bset ;
: scl0 PinSCL PortI2C bclr ;
: scl1 PinSCL PortI2C bset ;
```

Diese Worte kann man jetzt schon ausprobieren. Da an diesen beiden Pins auf der Ergänzungsplatine schon LEDs angeschlossen sind, kann man sofort sehen, ob es funktioniert. Nach `scl0` geht die rechte der 4 LEDs aus, nach `sda0` die links daneben.

Um ein Bit zu versenden, müssen wir das Datenbit setzen, ein bisschen warten, dann SCL auf 1 setzen, wieder ein bisschen warten und dann SCL zurück auf 0. Ich habe beschlossen, jeden Clock–Zyklus in 4 Teile zu zerlegen, Die kleinste Wartezeit nenne ich `tick` und sie soll eine Mikrosekunde betragen. Damit dauert ein Clock–Zyklus 4 Mikrosekunden, was lange genug sein sollte.

```
: tick 1 us ;
: 2tick tick tick ;
```

Man kann tick anfangs auch sehr viel länger wählen, z.B. `250 ms` (Millisekunden), dann kann man den Datenaustausch an den LEDs *sehen*. Um ein Bit zu verschicken, erwarten wir das Bit (0 oder 1) auf dem Stack, IF nimmt es vom Stapel und inspiziert es. Ist es 1 (true), dann wird die Datenleitung ebenfalls auf 1 gesetzt, andernfalls wird die Datenleitung auf 0 gesetzt.

```
IF sda1 ELSE sda0 ENDIF
```

Danach muss ich den Clock–Impuls erzeugen, und das vollständige Wort soll `bit>i2c` heißen:

```
\ sende 1 Bit per Clock-Zyklus
: bit>i2c ( bit -- )
    IF sda1 ELSE sda0 ENDIF
    tick scl1 2tick scl0 tick ;
```

Um ein bestimmtes Bit aus einem gegebenen Byte zu verschicken, muss ich dieses Bit aus dem Byte extrahieren. Um das Bit Nr. i aus dem Byte x zu erhalten, muss ich das Byte um i Stellen nach rechts verschieben (`x i rshift`) und evtl. vorhandene höherwertige Bits löschen (`$01 and`).

```
: getBit ( x i -- bit ) rshift $01 and ;
```

Damit kann ich jedes Bit aus dem Byte einzeln gewinnen. Das lässt sich schön überprüfen mit

```
%00001010 dup . 0 getBit . 10 0  ok
%00001010 dup . 1 getBit . 10 1  ok
%00001010 dup . 2 getBit . 10 0  ok
%00001010 dup . 3 getBit . 10 1  ok
%00001010 dup . 4 getBit . 10 0  ok
```

Um ein ganzes Byte zu verschicken, muss ich eine Schleife schreiben, die 8 mal durchlaufen wird.

```
8 0 DO ... LOOP
```

Dabei ist 8 der Grenzwert, 0 der Startwert, und das Wort `I` legt den aktuellen Zählerstand auf den Stapel. `I` produziert nacheinander die Werte 0 bis 7:

```
: 8mal 8 0 DO I . LOOP ;  ok
8mal <RET> 0 1 2 3 4 5 6 7  ok
```

Allerdings muss das höchstwertige Bit zuerst verschickt werden. Ich muß also statt I den Wert $8-(I+1)$ benutzen:

```
: 8mal 8 0 DO 8 I 1+ - . LOOP ; redefined 8mal
   ok
8mal <RET> 7 6 5 4 3 2 1 0  ok
```

Damit hätte ich alles beisammen, um ein ganzes Byte mit dem höchstwertigen Bit zuerst zu versenden. Beachte, dass das zu versendende Byte immer erst dupliziert werden muss, weil es sonst nach dem ersten Schleifendurchgang verloren ist. Am Ende der Schleife muss man dann das übriggebliebene Byte mit `drop` vom Stapel nehmen.

```
\ sende 1 Byte (8 Bit) MSB zuerst
: >i2c ( x -- )
    8 0 DO
        dup 8 I 1+ - getBit
        bit>i2c
    LOOP
    drop ;
```

Wenn man statt `bit>i2c` das Bit lediglich ausgibt mit `.`, dann erhält man folgende Ausgaben:

```
10 >i2c <RET> 0 0 0 0 1 0 1 0  ok
11 >i2c <RET> 0 0 0 0 1 0 1 1  ok
12 >i2c <RET> 0 0 0 0 1 1 0 0  ok
64 >i2c <RET> 0 1 0 0 0 0 0 0  ok
```

Mit diesen Worten kann ich 1 Byte auf dem Bus verschicken. Soweit so gut. Davor brauchen wir noch eine Startbedingung, und am Ende eine Stopbedingung.

```
\ sende START, STOP
: i2c_start ( -- ) tick sda0 2tick scl0 tick ;
: i2c_stop  ( -- ) tick scl1 2tick sda1 tick ;
```

Jetzt fehlt lediglich noch eine Kleinigkeit, nämlich, nach dem Versenden der Adresse zu überprüfen, ob sich der adressierte Baustein auch angesprochen fühlt. Dies tut er kund, indem er beim 9. Clock–Impuls die Datenleitung auf low zieht. Ich muss also die Datenrichtung an SDA umkehren, einen Clock–Impuls erzeugen und die Leitung lesen, während SCL high ist: *(Bei manchen Kontrollern muss man vorher SDA auf 1 setzen, bei diesem anscheinend nicht.)*

```
\ setze Pin SDA als Eingang / Ausgang
: sdaInput  ( -- ) PinSDA PddrI2C bclr ;
: sdaOutput ( -- ) PinSDA PddrI2C bset ;

\ lies SDA
: readSDA   ( -- bit )
    PinSDA PortI2C btst IF 1 ELSE 0 ENDIF ;

\ lies ACK oder NACK vom Empfänger
: ack<i2c ( -- bit )
    sdaInput
    tick scl1 tick readSDA tick scl0 tick
    sdaOutput ;
```

Mit diesen Worten kann ich dem Baustein ein Byte ‘x’ schicken.

```
: >portexpander ( x -- )
    i2c_start     \ START
    i2c_addr_portexander >i2c
                  \ Adresse schreiben
    ack<i2c drop  \ ACK lesen und verwerfen
    >i2c          \ Datenbyte schreiben
    ack<i2c drop  \ ACK lesen und verwerfen
    i2c_stop ;    \ STOP
```

Selbstverständlich kann ich das sofort testen:

```
15 >portexpander <RET> ok
```

Jetzt sollten 4 von den 8 LEDs am PCF8574 leuchten. Wenn man genau hinschaut, dann stellt man fest, dass die LEDs der 4 höherwertigen Bits leuchten. Das ist richtig, weil die LEDs gegen +5V geschaltet sind, und ein Pin mit einer Null den Strom durch die LEDs einschaltet. Oft will man das aber anders sehen, dafür muss man die Bits des zu verschickenden Bytes invertieren. Dafür gibt es auch ein Forth–Wort:

```
15 invert >portexpander <RET> ok
```

Jetzt ist alles so, wie's sein soll. Um länger in den Genuss blinkender LEDs zu kommen, hilft eine Schleife, in der eine Variable (`N`) jede Sekunde um 1 erhöht und dann invertiert auf dem i2c–Portexpander ausgegeben wird. Die Schleife soll unterbrochen werden, wenn ich eine beliebige Taste drücke. Die Forth–Worte `Variable`, `!` (store), `@` (fetch), `BEGIN ... UNTIL` (Schleife) und `key?` (ist eine Taste gedrückt worden?) kommen mir zu Hilfe:

```
Variable N
0 N !
: run
    0 >portexpander
    1000 ms
    BEGIN
        N @
        dup invert >portexpander
        1+ N !
        1000 ms
    key? UNTIL ;
```

Nach der Eingabe von `run` sollten auf den LEDs alle Sekunde ein neuer Wert erscheinen! Das Programm beendet sich, wenn man eine Taste drückt.

Natürlich kann man das alles von Hand eintippen, aber nach einem Reset ist erst mal alles verloren. Deshalb schreibe ich eine Datei `adv_01.fs`, in der alle Anweisungen stehen, eingeklammert mit den Worten `rom` und `ram`. Die Datei kann ich im terminal–Programm mit `include` an den Kontroller übertragen und danach die darin definierten Worte benutzen:

```
$ gforth terminal.fs
include adv_01.fs
run
```

and watch the show!

Das Programm zu diesem Teil befindet sich in der Datei `adv_01.fs`.

Verbesserungen

Einen i2c–Baustein in etwa 60 kurzen Zeilen Code anzusprechen, das finde ich schon bemerkenswert — auch wenn es bislang ganz ohne Fehlerbehandlung ist.

Adressierung

Der erste Schritt ist gemacht, ein i2c–Baustein lässt sich zum Schreiben ansprechen. Ein anderer Baustein lässt sich durch Auswechseln der Adresse ebenfalls addressieren. Allerdings würde ich dazu eine Kopie von `>portexpander` anlegen, in der nur die Adresse anders ist. Schön ist das nicht. Daher soll das Wort `>portexpander` so geändert werden, dass sich die Adresse auf dem Stapel befindet.

```
: send1byte>i2c ( x addr -- )
    i2c_start    \ START
    >i2c         \ Adresse 'addr' senden
    ack<i2c drop \ ACK lesen und verwerfen
    >i2c         \ DatenByte 'x' senden
    ack<i2c drop \ ACK lesen und verwerfen
    i2c_stop ;   \ STOP

15 i2c_addr_portexpander send1byte>i2c
```

Schreiben/Lesen

Bislang schreibe ich Daten an den i2c–Baustein. Um zu lesen, brauchen wir analog zu `>i2c` ein Wort `<i2c`, welches ein Byte vom Bus liest:

```
: <i2c ( -- x )
    sdaInput  \ SDA auf Input schalten
    0         \ Datenbyte auf 0 gesetzt
    8 0 DO    \ Schleife über 8 Bit
        1 lshift
              \ das bisherige Datenbyte um 1
              \ Stelle nach links verschieben
              \ einen Clock-Impuls erzeugen und
              \ das Bit lesen
        tick scl1 tick readSDA tick scl0 tick
        +     \ das Bit zum Datenbyte addieren
    LOOP      \ Ende der Schleife
    sdaOutput ;  \ SDA auf Output
```

Wenn ich vom Bus Lesen will, dann muss ich den Baustein adressieren und dabei das Read–Bit setzen. Die Adresse ändert sich von `($20 << 1) | 0 == $40` auf `($20 << 1) | 1 == $41`.

Wenn man den Baustein direkt lesen kann, dann ist das schon alles, und einmal lesen sieht etwa so aus:

```
: get1byte<i2c ( addr -- x )
    i2c_start    \ START
    >i2c         \ ADRESSE
    ack<i2c drop \ ACK lesen und verwerfen
    <i2c         \ DatenByte _lesen_
    1 bit>i2c    \ NACK _senden_
    i2c_stop ;   \ STOP
```

Allerdings muss man fast immer nach der Adresse erst einmal ein Byte schreiben: Das kann ein *Controlbyte* sein, oder die Adresse im Speicher des Bausteins, von der man die Daten will, oder die Kanalnummer bei einem AD–Wandler etc. Sogar bei dem simplen Portexpander muss ich erst mal eine 1 auf alle Pins schreiben und dann lesen, damit das Ergebnis zuverlässig ist.

Die Folge ist dann eben Start, Adresse senden, ACK lesen, das Controlbyte senden, ACK lesen, danach wird eine repeated–start–Bedingung gesendet, dann ein oder mehrere Datenbytes gelesen und nach jedem gelesenen Byte ein ACK *gesendet*. Nach dem letzten gelesenen Byte verschickt der i2c–Master ein NACK, also eine 1 statt einer 0. Damit wird dem adressierten Baustein mitgeteilt, dass keine weiteren Bytes gelesen werden. Und zum Abschluss wie immer eine Stopp–Bedingung.

```
\ Sende "repeated start"
: i2c_rstart ( -- )
    sda1 tick scl1 tick sda0 tick scl0 tick ;
```

Variable Anzahl von Bytes lesen/schreiben

Ein Lesevorgang auf dem i2c–Bus besteht also normalerweise aus dem Verschicken von mindestens einem Controlbyte an die Adresse, dem Lesen von einem oder mehreren Datenbytes, jeweils gefolgt von ACK oder NACK. Um das besser programmieren zu können, ändere ich die Worte `>i2c` und `<i2c` so ab, dass sie eine variable Anzahl von Bytes verschicken können.

```
\ verschicke N Bytes nach Adresse addr
\ das "most significant Byte" liegt oben
\ auf dem Stapel
: NB>i2c ( x1 .. xN.msB N addr -- )
    i2c_start \ START
    >i2c ack<i2c drop
              \ ADRESSE senden, ACK lesen
    0 DO      \ Schleife, nimmt N vom Stapel
        >i2c ack<i2c drop
              \   DATA_BYTE verschicken
    LOOP      \ Ende Schleife
    i2c_stop  \ STOP
;

\ Lies N Bytes von Adresse addr
\ das "least significant Byte" liegt
\ danach oben auf dem Stapel
\ Adresse und ControlByte sind schon verschickt
: NB<i2c ( N addr -- xN.msB .. x1 )
  i2c_rstart  \ REPEATED_START
  1+ >i2c ack<i2c drop
            \ ADRESSE | READ senden, ACK lesen
  1- dup 0 > IF
            \ Schleife über N-1 Byte
    0 DO    \   Datenbyte = 0 (initialisieren)
      <i2c 0 bit>i2c
            \   DATA_BYTE lesen, ACK schicken
    LOOP    \ Ende Schleife
  ENDIF
  <i2c 1 bit>i2c
            \ letztes Byte lesen, NACK schicken
  i2c_stop  \ STOP
;
```

Die Schleife geht hier über $n-1$ Byte, nicht n Byte, weil nur $n-1$ ACK verschickt werden. Man hätte das auch anders lösen können, aber mir gefällt es so. Ich hätte auch das Versenden von ACK und NACK getrennt definieren können:

```
: ack>i2c  ( -- ) 0 bit>i2c ;
: nack>i2c ( -- ) 1 bit>i2c ;
```

Aber da diese Stelle nur in `NB<i2c` vorkommt, erschien es mir nicht lohnend. Das Lesen vom Portexpander geht dann so:

```
: <portexpander ( -- x )
    $ff                       \ ControlByte
    1 i2c_addr_portexpander NB>i2c
    1 i2c_addr_portexpander NB<i2c
;
```

```
<portexpander cr . <RET>
255 ok
```

Jetzt ist es nett und übersichtlich. Zwar muss ich die LEDs durch eine Schalterleiste auswechseln, sonst kommt immer das gleiche Ergebnis (255), aber das ist kein Programmierproblem. Das Programm zu diesem Teil befindet sich in der Datei `adv_02.fs`.

Thermometer

In diesem Teil kommt ein Thermometer mit i2c–Schnittstelle an den i2c–Bus, welches vom R8C–Kontroller gelesen wird. Das Ergebnis wird auf dem LC-Display ausgegeben. Etwa 4 Messungen pro Sekunde sollen es werden. Der Temperatursensor ist ein LM75, die Auflösung beträgt 0.5°C, die Genauigkeit etwa ±2°C. Der Baustein hat die 7–Bit–Adresse $48, wenn alle Adressbits auf low sind, und $4f, wenn alle Adressbits gesetzt sind. `$4f << 1 == $9e`.

```
$9e Constant i2c_addr_lm75
```

Der LM75 liefert das Ergebnis in 2 Bytes, das höherwertige Byte (xh) wird zuerst übertragen. Im niederwertigen Byte (xl) ist nur noch ein Bit gültig, nämlich das höchste. (Beim LM75**A** sind es die *drei* höchsten Bits, was einer Auflösung von 0.125°C entspricht) Um die Temperatur zu lesen, ist eine $00 zu verschicken (pointer register, s. Datenblatt), das entspricht unserem Controlbyte. Mit den oben erarbeiteten Worten ist das Auslesen des LM75 ein Kinderspiel:

```
: get.T ( -- xh xl )
   $00                        \ ControlByte
   1 i2c_addr_lm75 NB>i2c     \ verschicken
   2 i2c_addr_lm75 NB<i2c     \ 2 Byte lesen
;
```

Um die beiden Bytes in eine Temperatur umzuwandeln, konsultiert man das Datenblatt. Die 9 Bits bilden eine Zahl mit Vorzeichen, die von −55.0 bis +128.0 reicht, in Schritten von 0.5.

xl liegt oben auf dem Stapel, die 7 ungültigen Bits lösche ich mit

```
$80 and
```

dann bringe ich xh nach oben

```
swap
```

und rücke das Byte um 8 Stellen nach links

```
8 lshift
```

dann werden die beiden Teile einfach addiert

```
+
```

Das Resultat ist die Temperatur, allerdings um den Faktor 256 zu groß. Wenn ich das Resultat einfach durch 256 teile, dann verliere ich die Nachkommastelle. In Forth skaliert man solche Zahlen mit einem geeigneten Wert. Um eine Nachkommastelle zu behalten, skaliere ich die Temperatur mit 10. Wenn ich jetzt aber einfach

```
10 * 256 /
```

hinschreibe, dann ist das Ergebnis dennoch unbefriedigend, weil das Ergebnis der Operation

```
10 *
```

nur die Größe einer Zelle hat. Ist das Zwischenergebnis größer, dann verliere ich die höchsten Bits. Zu Hilfe kommt das Forth–Wort `*/` (star–slash), welches ein Zwischenergebnis doppelter Breite benutzt. Also

```
10 256 */
```

oder

```
5 128 */
```

erzeugen die Temperatur in Einheiten von 1/10 °C.

```
: decode.T ( xh xl -- T*10 )
    $80 and     \ die ungültigen Bits löschen
    swap        \ xh nach oben holen
    8 lshift    \ xh << 8
    +           \ == (xh << 8) + ( xl & 0x80)
    5 128 */ ;  \ == T*10
```

Versuch macht kluch:

```
get.T cr .s decode.T cr . <RET>
<2> 24 255
245  ok
```

Das sind 24.5°C, soweit so schön. Um die Zahl mit Nachkommastelle auf das LCDisplay zu schreiben, muss man im Forth–Buch oder in der Dokumentation von gforth nachlesen. Hier das Ergebnis ohne große Erklärungen:

```
\ _immer_ das Vorzeichen anzeigen, '+' or '-'
: sign! 0 < IF 45 ELSE 43 ENDIF hold ;

\ formatiere T zur Ausgabe mit type or lcdtype
: format.T ( T*10 -- )
    dup >R  \ Kopie fuer das Vorzeichen aufheben
    abs s>d \ Betrag, single in double umwandeln
    \ 1 Ziffer, "." Dezimalpunkt, 2 Ziffern,
    \ Vorzeichen
    \ Zahlen >= 1000 werden VORNE abgeschnitten!
    <# # 46 hold # # R> sign! #> \ == "%+5.1f"
;
```

Ausprobieren mit

```
lcdpage
get.T decode.T format.T lcdtype
```

Das Thermometer soll ständig laufen, braucht also wieder eine Schleife. Eine Messung soll 4-mal pro Sekunde sein:

```
: thermometer
    BEGIN
        lcdpage
        get.T decode.T format.T lcdtype
    250 ms
    key? UNTIL
;

thermometer
```

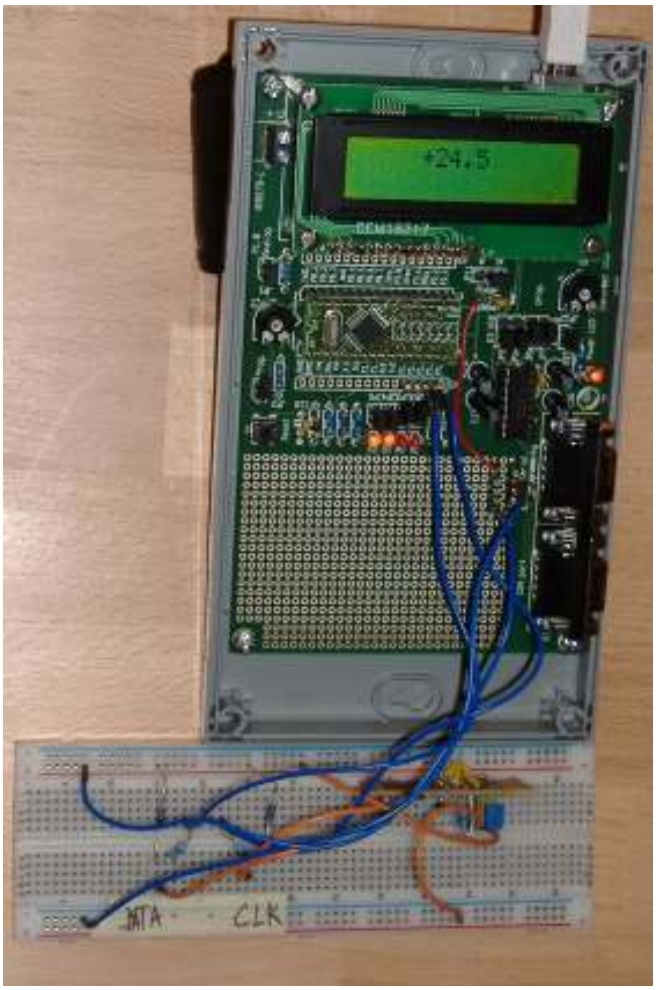


Abbildung 1: Thermometer–Aufbau

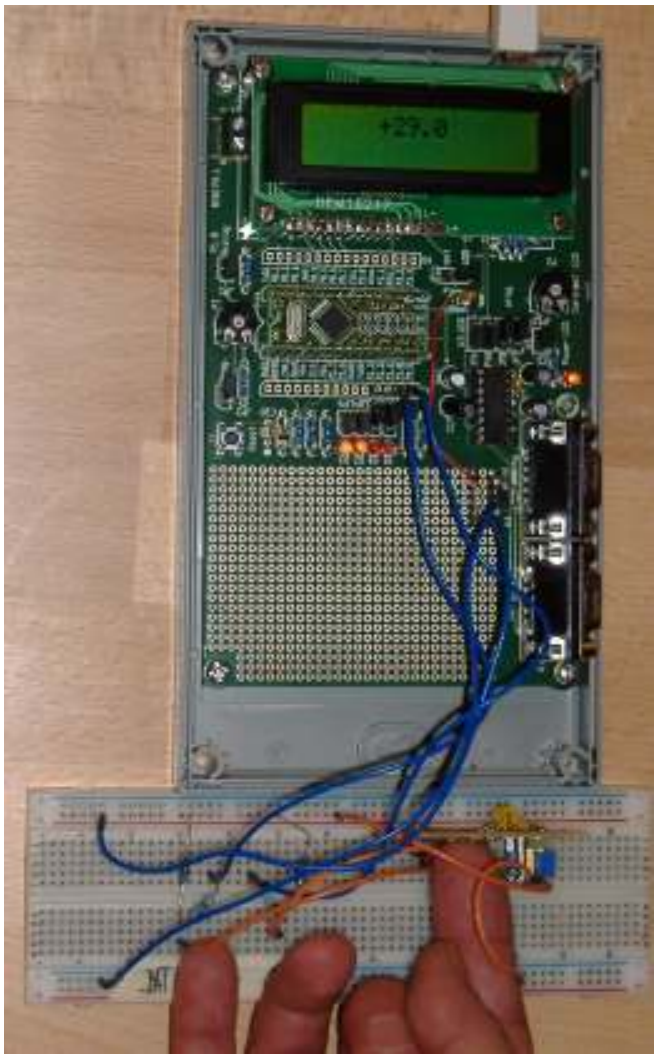
Auffällig ist, dass das LCDisplay recht unruhig *flimmert*. Das kommt durch die häufigen Aufrufe von `lcdpage`. Besser ist es, ein Wort `lcdpos` zu schreiben, welches den Cursor an eine bestimmte Stelle setzt.

```
: lcdpos ( row col -- )
    swap $40 * + $80 + lcdctrl! &1 ms ;
```

Damit kann man `thermometer` aufbessern:

```
: thermometer
    lcdpage
    BEGIN
        0 5 lcdpos
        get.T decode.T format.T lcdtype
        250 ms
    key? UNTIL
;
thermometer
```

Den Finger auf den LM75 ...



... and watch the show!

Das Programm zu diesem Teil befindet sich in der Datei `adv_03.fs`.

Referenzen

1. Elektor–Hefte 12/2005, 01/2006, 02/2006 `http://www.elektor.de`
2. `http://www.forth-ev.de/wiki/doku.php/projects:r8c:r8c_forth`
3. `http://groups.google.com/group/de.comp.lang.forth/browse_thread/thread/038d092aab9c558c/80aa79ab4b8624cd#80aa79ab4b8624cd` (thread zur Installation)
4. `http://home.iae.nl/users/mhx/sf.html` Starting Forth Online Transkript
5. `http://thinking-forth.sourceforge.net/` Thinking Forth Online
6. I2C–Bus angewandt (Elektor–Verlag, ISBN 3928051717)
7. `http://en.wikipedia.org/wiki/Forth_programming_language`

Listings

```
1   \ 2006-07-31 SMBus.fs --- SMBus Basiscode
2   \ i2c Bus am R8C
3   \ P1.0 -- SCL
4   \ P1.1 -- SDA
5
6   \ Konstanten
7   0     Constant PinSCL
8   1     Constant PinSDA
9   port1 Constant PortI2C
10  $E3   Constant PddrI2C
11
12  \ SDA,SCL Pins auf 0 oder 1 setzen
13  : sda0 PinSDA PortI2C bclr ;
14  : sda1 PinSDA PortI2C bset ;
15  : scl0 PinSCL PortI2C bclr ;
16  : scl1 PinSCL PortI2C bset ;
17
18  \ 1 SCL Zyklus sind im Programm 4 ticks
19  : tick 1 us ;
20  : 2tick tick tick ;
21
22  \ sende 1 Bit
23  : bit>i2c ( bit -- )
24      IF sda1 ELSE sda0 ENDIF
25      tick scl1 2tick scl0 tick ;
26
```

```
27  \ erzeuge Bit Nr. i aus Byte x als 0 oder 1
28  : getBit ( x i -- b ) rshift $01 and ;
29
30  \ sende 1 Byte, (8 Bit) "most significant bit" zuerst
31  : >i2c ( x -- )
32      8 0 DO
33          dup 8 I 1+ - getBit
34          bit>i2c
35      LOOP
36      drop ;
37
38  \ sende START, STOP, REPEATED_START
39  : i2c_start ( -- ) tick sda0 2tick scl0 tick ;
40  : i2c_stop  ( -- ) tick scl1 2tick sda1 tick ;
41  : i2c_rstart sda1 tick scl1 tick sda0 tick scl0 tick ;
42
43  \ setze SDA zum Lesen / Senden
44  : sdaInput  ( -- ) PinSDA PddrI2C bclr ;
45  : sdaOutput ( -- ) PinSDA PddrI2C bset ;
46
47  \ lies SDA
48  : readSDA   ( -- f )
49      PinSDA PortI2C btst IF 1 ELSE 0 ENDIF ;
50
51  \ lies 1 Byte (8 Bit), "most significant bit" zuerst
52  : <i2c ( -- x )
53      sdaInput
54      0
55      8 0 DO
56          1 lshift
57          tick scl1 tick readSDA tick scl0 tick
58          +
59      LOOP
60      sdaOutput ;
61
62  \ lies ACK (0) oder NACK (1) vom Partner
63  : ack<i2c ( -- f )
64      sdaInput
65      tick scl1 tick readSDA tick scl0 tick
66      sdaOutput ;
67
68  \ sende N Bytes nach Adresse addr
69  \ das "most significant Byte" liegt oben auf dem Stapel
70  : NB>i2c ( x1 .. xN.msB N addr -- )
71      i2c_start               \ START
72      >i2c ack<i2c drop       \ ADRESSE senden, ACK lesen
73      0 DO                    \ Schleife, nimmt N vom Stapel
74          >i2c ack<i2c drop   \   DATA_BYTE verschicken
75      LOOP                    \ Ende Schleife
76      i2c_stop                \ STOP
77  ;
78
79  \ lies N Bytes von Adresse addr
80  \ das "least significant Byte" liegt oben auf dem Stapel
81  \ Adresse und ControlByte sind schon verschickt
82  : NB<i2c ( N addr -- xN.msB .. x1 )
83      i2c_rstart              \ REPEATED_START
84      1+ >i2c ack<i2c drop    \ ADRESSE | READ senden, ACK lesen
85      1- dup 0 > IF           \ Schleife ueber N-1 Byte
86          0 DO                \   Datenbyte = 0 (initialisieren)
87              <i2c 0 bit>i2c  \   DATA_BYTE lesen, ACK schicken
88          LOOP                \ Ende Schleife
89      ENDIF                   \
90      <i2c 1 bit>i2c          \ letzes Byte lesen, NACK schicken
91      i2c_stop                \ STOP
92  ;
```

```
\ 2006-06-28 EW adv_01.fs
\
\ i2c_bus connected to r8c
\ P1.0 -- SCL
\ P1.1 -- SDA
\ device PCF8574 (8bit IO) at address $20

rom

include SMBus.fs

$40   Constant i2c_addr_portexpander

\ send Byte x to portexpander
: >portexpander ( x -- )
    i2c_start                  \ START
    i2c_addr_portexpander >i2c \ Adresse schreiben
    ack<i2c drop               \ ACK lesen und verwerfen
    >i2c                       \ Datenbyte schreiben
    ack<i2c drop               \ ACK lesen und verwerfen
    i2c_stop                   \ STOP
;

\ watch the show!
Variable N
0 N !
: run
    0 >portexpander
    1000 ms
    BEGIN
        N @
        dup invert >portexpander
        1+ N !
        1000 ms
    key? UNTIL
;

ram
```

```
\ 2006-06-28 EW adv_02.fs
\
\ i2c Bus am R8C
\ P1.0 -- SCL
\ P1.1 -- SDA
\ am Bus:
\ PCF8574 (8bit IO), 7-bit Adresse: $20

rom

include SMBus.fs

$40   Constant i2c_addr_portexpander

\ vom portexpander lesen
: <portexpander ( -- x )
    $ff                    \ "ControlByte"
    1 i2c_addr_portexpander NB>i2c
    1 i2c_addr_portexpander NB<i2c
;

ram
```

```
\ 2006-06-28 EW adv_03.fs --- Thermometer
\
\ i2c Bus am R8C
\ P1.0 -- SCL
```

```
\ P1.1 -- SDA
\ am Bus:
\ PCF8574 (8bit IO), 7-bit Adresse: $20
\ LM75 (Thermometer, 9-bit signed, 7-bit Adresse: $4f

rom

include SMBus.fs

$9e   Constant i2c_addr_lm75

: get.T ( -- xh xl )
    $00                     \ ControlByte
    1 i2c_addr_lm75 NB>i2c  \ verschicken
    2 i2c_addr_lm75 NB<i2c  \ 2 Byte lesen
;

: decode.T ( xh xl -- T*10 )
    $80 and                 \ die ungueltigen Bits loeschen
    swap                    \ xh nach oben holen
    8 lshift                \ xh << 8
    +                       \ == (xh << 8) + ( xl & 0x80)
    5 128 */                \ == T*10
;

\ _always_ display sign '+' or '-'
: sign! 0 < IF 45 ELSE 43 ENDIF hold ;

\ format T for display (type or lcdtype)
: format.T ( T*10 -- )
    dup >R                  \ store copy for sign
    abs s>d                 \ remove sign, make double
    \ 1 digit, "." 2digits sign_always
    \ Numbers >= 1000 are truncated in the high digits!!!
    <# # 46 hold # # R> sign! #> \ == "%+5.1f"
;

\ write string to lcd position col (0..15) row (0..1)
\ with next lcdtype
: lcdpos ( row col -- )
    swap $40 * + $80 + lcdctrl! &1 ms ;

: thermometer
    lcdpage
    BEGIN
        0 5 lcdpos
        get.T decode.T format.T lcdtype
        250 ms
    key? UNTIL
;

ram
```

Gforth–R8C erzeugt Sudoku–Rätsel–Vorgaben

Fred Behringer

Einleitung und Ziel

In Sudoku (= Ziffer je einmal), dem neuen geistigen Volkssport für Anspruchsvolle, verlangt man, dass in einer 9x9–Matrix jede Ziffer von 1 bis 9 in jeder Zeile, in jeder Spalte und in jeder 3x3–Teilmatrix genau einmal auftritt. Eine gewisse Anzahl von Elementen (meist um die 27d herum) werden vorgegeben, der Rest muss erraten werden. Sudoku–Rätselsammlungen in Buch– oder Blockform (mit durchschnittlich 200 Rätseln und auf den letzten Seiten angegebenen Lösungen) kosten inzwischen nur noch 2–Euro–fünfzig.

Hexadoku (siehe VD–Heft 2/2006) fängt bei 0 an und hört mit F auf, Sudoku fängt bei 1 an und hört mit 9 auf. Es soll hier schon verraten werden, dass man sich manches vereinfacht, wenn man die gesamte Matrix zunächst einmal um 1 verringert, die reduzierte Matrix (die also auf Ziffern von 0 bis 8 statt von 1 bis 9 aufbaut) dann verarbeitet und erst bei den Bildschirm–Ausgaben wieder 1 hinzuzählt. Dass das funktioniert, ist klar, wenn man bedenkt, dass es ja auf die *Aufkleber* der Elemente nicht ankommt, wenn sie nur alle voneinander verschieden sind. Selbst auf eine bestimmte Reihenfolge der Ziffern kommt es nicht an. Man könnte auch per definitionem einfach eine neue Reihenfolge einführen, was mathematisch auf eine Umordnung hinausliefe.

Der R8C ist ein kleiner schwarzer Winzling mit etwas Drumherum. Das Drumherum soll uns hier nicht sonderlich interessieren. Harvard–Architektur oder von–Neumann–Architektur, Flash–Speicher oder RAM–Speicher, wer will das so genau wissen?

Auf der diesjährigen Forth–Tagung hat Bernd Paysan einiges über das Drumherum der Übertragung seines Gforth–Systems auf den R8C berichtet und Heinz Schnitter hat seinen Assembler für den R8C skizziert.

Wir wollen uns die Sache einfacher machen. Wir stellen uns einfach auf den Standpunkt, wir wissen gar nichts. Wir behandeln Gforth auf dem R8C einfach so wie viele andere Forth–Systeme auch, mit einem gewissen RAM–Bereich, in den wir (interaktiv per Tastatur) Forth–Wort um Forth–Wort eines Anwendungsprogramms packen können.

Ausgangspunkt ist irgendeine Version von `gforth-r8c.mot` (siehe [1]), die (ein für alle Mal) per FDT in den Flash Speicher des R8C geladen wird — genau so, wie es bei den drei Beispiels–mot–Programmen auf der Glyn–CD auch geschieht. (Falls sich Fragen ergeben: Ich habe Gforth 0.6.2-20060409 verwendet.) Der interaktive Verkehr des eingeladenen Gforth–R8C–Systems mit dem PC als Terminal (über die Tastatur und den PC–Bildschirm) möge sich per Windows–HyperTerminal abspielen: Die Baudrate muss bei Gforth–R8C 38400 betragen! (Bei FDT pendelt sich die Baudrate auf 9600 ein.) Für den Rest der Parameter bei HyperTerminal verwende man 8N1, kein Protokoll. (Ich habe meine Experimente auf einem AMD–K6–2/500 unter Windows 98 und Windows ME und auf einem AMD–64/3200+ unter Windows XP durchgeführt.) Beim Arbeiten mit HyperTerminal könnte es sein, dass das Schriftbild (zunächst noch) viel zu klein erscheint. Man ändere das (beispielsweise) über *Ansicht – Schriftart – Courier New, Standard, 11 – Großbild* ab. Mit der größeren Schrift wird auch das Terminal–Fenster größer.

Schwierig ist die geringe Anzahl von internen RAM–Bytes (externes RAM gibt es beim R8C nicht). Etwas mehr als 700. Das war der Grund, nicht auf Hexadoku (16x16) der Zeitschrift Elektor aufzubauen, sondern sich auf das ursprüngliche Sudoku (9x9) zu besinnen. Das per Tastatur einzugebende Forth–Programm (siehe unten) benötigt nur etwas mehr als 640 Bytes. Es sind also noch genügend RAM–Bytes für schnelle Zusatzexperimente frei (siehe weiter unten bei den blinkenden LEDs). Warum Sudoku ausgerechnet auf dem R8C? Mehr dazu weiter unten.

Genau wie im VD–Heft 2/2006 geht es hier nicht um Sudoku–*Lösungsmöglichkeiten*, sondern um die Erzeugung von zulässigen Sudoku–*Vorgaben*. Dazu wird von einer *zulässigen* Sudoku–Matrix ausgegangen (die man als *Lösung* eines vorgegebenen Sudoku–Rätsels auffassen kann). Über einen einfachen Zufalls–Mechanismus (der hier nicht besprochen werden soll) werden daraus willkürlich etwa 27d Vorgabeelemente (Erfahrungswert aus der Sudoku–Literatur) ausgewählt.

Die unmittelbar einsichtige zulässige Anfangsmatrix (im Hexadoku–Rätsel aus dem VD–Heft 2/2006 hatte ich dafür die Bezeichnung *kanonische* Matrix verwendet) wird schon bei Tastatur–Eingabe des Sudoku–Programms erzeugt und in den RAM–Speicher gelegt und kann danach leider nicht mehr *per Knopfdruck* wiederhergestellt werden, ohne das Sudoku–Programm zu zerstören. Der RAM–Platz war für einen solchen Knopfdruck–Programmbaustein zu knapp. Allerdings hat man über `m` und `ij!` die (umständliche) Möglichkeit, sich durch Handeingabe eine *beliebige* Sudoku–Matrix, also auch eine *kanonische*, zu erzeugen. Platz für die Befehle für eine solche interaktive (über den Interpreter, nicht erst den Compiler laufende) Eingabe ist auf jeden Fall noch vorhanden. Platz für die Abspeicherung der Matrix selbst ist ja schon da. Ein Beispiel: Für das dritte Element (sein Wert ist 0) in der zweiten Zeile (bei der reduzierten Matrix beginnt die Zählung mit 0, nicht mit 1) gebe man für ein solches Vorhaben ein:

```
0 2 3 ij!
```

Die (reduzierte) kanonische Sudoku–Matrix sieht wie folgt aus:

```
0 1 2 3 4 5 6 7 8
3 4 5 6 7 8 0 1 2
6 7 8 0 1 2 3 4 5
1 2 3 4 5 6 7 8 0
4 5 6 7 8 0 1 2 3
7 8 0 1 2 3 4 5 6
2 3 4 5 6 7 8 0 1
5 6 7 8 0 1 2 3 4
8 0 1 2 3 4 5 6 7
```

Die Überprüfung der Zulässigkeit dieser Matrix (die Frage, ob die Sudoku–Regeln erfüllt sind) fällt nicht schwer: Man gehe einfach mit dem Auge durch, Zeile um Zeile, Spalte um Spalte, 3x3–Teilmatrix um 3x3–Teilmatrix. Mit

```
2 3 ij@ .
```

(der Punkt gehört dazu) kann man sich vergewissern, dass die Eingabe richtig angekommen ist.

Will man etwas Variation ins Programm bringen, so kann man statt der verwendeten *kanonischen* Matrix auch deren *Transponierte* (Zeilen mit Spalten vertauschen) einsetzen. Achtung: Im Feld m wird die um 1 reduzierte Sudoku–Matrix abgespeichert. Die Eintragungen gehen von 0 bis 8. Erst beim Bildschirm–Anzeigen über v wird alles wieder um 1 nach oben verschoben. Man beachte: Bei diesen nachträglichen Per–Hand–Eintragungen wird nicht ans Ende des Dictionarys *compiliert* (nicht `c,`). Das (mit `m` benannte) Speicherfeld ist ja schon vorhanden. Die (neuen) Werte werden per `c!` (byteweise) dorthin (nach `m`) *abgespeichert*. Also:

```
0 m 00 + c! 3 m 01 + c! 6 m 02 + c!
```

und so weiter und so fort, oder etwas einfacher:

```
0 0 0 ij!  3 0 1 ij!  6 0 2 ij!
```

Wichtig sind eigentlich nur die Zahleneingaben. Also nutze man die Möglichkeiten von Forth und baue sich ganz schnell eigens für diesen Zweck beispielsweise das Hilfswort

```
: y ij! ;
```

Dies lässt sich in direkter Tastatureingabe auch noch nachträglich erledigen. So viel Platz ist da, im R8C. Das spart Tipparbeit und das Tastatur–Umschalten für `!` .Was eingegeben und nach `m` gelegt werden soll, ist die gleich folgende *transponierte kanonische* Matrix. Sie entsteht aus der eingangs erwähnten *kanonischen* Matrix, indem man dort Zeilen und Spalten vertauscht. Der leichteren Überprüfbarkeit wegen habe ich im Folgenden die kanonische Matrix (als zweite) neben ihre Transponierte geschrieben.

```
0 3 6 1 4 7 2 5 8   0 1 2 3 4 5 6 7 8
1 4 7 2 5 8 3 6 0   3 4 5 6 7 8 0 1 2
2 5 8 3 6 0 4 7 1   6 7 8 0 1 2 3 4 5
3 6 0 4 7 1 5 8 2   1 2 3 4 5 6 7 8 0
4 7 1 5 8 2 6 0 3   4 5 6 7 8 0 1 2 3
5 8 2 6 0 3 7 1 4   7 8 0 1 2 3 4 5 6
6 0 3 7 1 4 8 2 5   2 3 4 5 6 7 8 0 1
7 1 4 8 2 5 0 3 6   5 6 7 8 0 1 2 3 4
8 2 5 0 3 6 1 4 7   8 0 1 2 3 4 5 6 7
```

Man prüft sofort nach, Zeile um Zeile, Spalte um Spalte, 3x3–Teilquadrat um 3x3–Teilquadrat, dass die Sudoku–Regeln erhalten bleiben.

Weitere Möglichkeiten, Variation ins Geschehen zu bringen: Klar, dass man aus einer beliebigen *zulässigen* Sudoku–Matrix wieder eine *zulässige* Sudoku–Matrix bekommt, wenn man alle Zeilen oder/und alle Spalten in umgekehrter Reihenfolge hinschreibt. Das alles wurde im VD–Heft 2/2006 schon beschrieben und begründet, muss aber hier *außen vor* bleiben, da das RAM des R8Cs eben doch nicht allzu viel hergibt. Insgesamt gilt:

Geänderte Vorgabe–Matrizen bei gleichbleibender Sudoku–Matrix werden durch `v` angezeigt. Geänderte Sudoku–Matrizen mit passender Vorgabe–Matrix werden durch `s` angezeigt.

Solche Tastatureingaben müssen natürlich immer durch [return] abgeschlossen werden, um Wirksamkeit zu erlangen.

Die Sudoku–Matrix sollte bei möglichst kleinem Programm genügend durchgerüttelt werden. Ich habe das mit den willkürlich aufzurufenden Bausteinen `x addn xchi xchj` erreicht.

Bei `x addn` kann `x` eine beliebige Zahl sein. Sie wird in der Form `9 mod` eingebaut. Man überzeugt sich leicht davon (Aufgabe für den Leser!), dass bei Addition modulo 9 von beliebigem x zu allen Matrix–Elementen die Sudoku–Regeln unverletzt bleiben. `xchj` vertauscht willkürlich zwei Spalten in einem Dreierlängsstreifen. (Dreierlängsstreifen, damit nicht nur die Spalten, sondern auch die geänderten Teilquadrate Regeltreue bewahren.) `xchi` macht Ähnliches, nur nicht ganz so willkürlich.

Man kann bei irgendeiner schon erreichten Sudoku–Matrix die angegebenen drei Befehle auch (einzeln oder in beliebiger Reihenfolge) per Hand eingeben.

Ich gebe zu, dass es bei diesen wirklich sehr wenigen Umformungsmöglichkeiten einfach ist, bei den geänderten *Sudoku*–Matrizen den ursprünglichen systematischen Aufbau (*kanonische Matrix*) wiederzuerahnen — zumal, wenn man das Prinzip kennt. Aus den vom Programm per `v` angebotenen veränderten *Vorgabe*–Matrizen dürfte der Wiedererkennungseffekt jedoch nicht mehr ganz so groß sein. Von *leichtem Erraten* kann bei den erzeugten Vorgabe–Matrizen sicher keine Rede sein – wenn einem nicht jemand das kanonische Aufbau–Prinzip vorher verraten hat.

Natürlich ist es nicht Aufgabe des R8Cs, Sudoku–Rätsel–Vorgaben zu liefern. Dazu nimmt man den PC. Gforth–R8C ist aber ein ausgewachsenes Forth–System (Bernd und Heinz sei Dank) und es kann sicher nicht schaden zu sehen, dass der R8C mit Gforths Hilfe auch anspruchsvollere (Nichtsteuer–)Aufgaben bewältigen kann. Andererseits braucht man sich aber auch von den Beispielsprogrammen auf der von Elektor mitgelieferten Glyn–CD nicht beeindrucken zu lassen. Man gebe (von Gforth–R8C aus interaktiv (!) ins per Reset–Taste wieder löschbare R8C–RAM) ein:

```
: blink
    100 0 do
       i led!  100 ms
       0 led!  100 ms
    loop ;
```

und schließe mit der Return–Taste ab. Aufruf per `blink` lässt die vier LEDs als Kombination im Rhythmus der Binärzahlen von 0000 über 0001 bis 1111 aufleuchten – mindestens genauso schön wie bei Glyn. Ohne Umwege über den PC, mit ganz normalen Forth–Mitteln. (Bernd hat dazu das schöne Forth–Wort `led!` mit den 16 LED–Kombinationen als Eingabeparameter eingeführt.)

Das ist aber dann auch ein weiterer Grund, weshalb ich auf Sudoku verfallen bin: Sudoku braucht keine Elektor–Experimentierplatine (für 69,– Euro), das R8C–Platinchen in Minimalbeschaltung reicht (die Minimalbeschaltung habe ich um den RS232 ergänzt und ich danke Rolf Schöne für Hinweise). Bei der LED–Ansteuerung müsste man sich auf der Minimalbeschaltung die LEDs selbst hinzubasteln — kein ersthaftes Problem, aber sicher nicht jedermanns Sache.

Es ist nicht ganz einfach, das Sudoku–Programm fehlerfrei ins R8C–RAM zu bringen. In jeder Zeile kann man zwar mit der Delete–Taste wieder zurückgehen, um Fehleingaben zu korrigieren, ist aber erst einmal die betreffende Eingabezeile über die Return–Taste abgeschlossen worden und hat das System das betreffende Wort akzeptiert, obwohl es nicht korrekt eingegeben wurde, dann bleibt einem nur die Reset–Taste. (Die ganze bisherige (Sudoku–)Arbeit ist dann futsch.) Das Wort `forget` ist nicht vorgesehen. `forget` ist aber auch kein ANS–Wort. Hat man es aber geschafft, das Sudoku–Programm ins R8C–RAM zu bringen, dann kann man das Ganze per `savesystem` fixieren. Die Reset–Taste greift ab dann (auch nach Ausschalten und Wiedereinschalten) diesen Teil nicht mehr an. Kleinere Schnellzusätze, wie das `blink` bei den LEDs, wird man dann immer noch per Reset–Taste wieder los. Will man aber wirklich das gesamte (eingegebene und per `savesystem` fixierte) Sudoku–Programm wieder loswerden, dabei aber das schon per FDT eingeflashte Gforth–R8C beibehalten, dann erreicht man das per `empty`.

[1] `http://www.forth-ev.de/wiki/doku.php/projects:r8c:r8c_forth`

Forth-Programm

```
1   \ Nach geladenem Gforth-R8C ueber die Tastatur in das
2   \ RAM des R8Cs eingeben. Danach per SAVESYSTEM
3   \ "festfrieren". Kann per EMPTY bis auf das eigentliche
4   \ Forth-System geloescht werden.
5
6   \ Ich verwende im gesamten Programm Grossschreibung.
7   \ Kleinschreibung reicht aber. Das System ist auf
8   \ egal voreingestellt.
9
10  HEX
11
12  \ Sudoku-Matrix, zeilenweise, pro Element ein Byte
13  CREATE M
14
15  0 C, 1 C, 2 C,  3 C, 4 C, 5 C,  6 C, 7 C, 8 C,
16  3 C, 4 C, 5 C,  6 C, 7 C, 8 C,  0 C, 1 C, 2 C,
17  6 C, 7 C, 8 C,  0 C, 1 C, 2 C,  3 C, 4 C, 5 C,
18
19  1 C, 2 C, 3 C,  4 C, 5 C, 6 C,  7 C, 8 C, 0 C,
20  4 C, 5 C, 6 C,  7 C, 8 C, 0 C,  1 C, 2 C, 3 C,
21  7 C, 8 C, 0 C,  1 C, 2 C, 3 C,  4 C, 5 C, 6 C,
22
23  2 C, 3 C, 4 C,  5 C, 6 C, 7 C,  8 C, 0 C, 1 C,
24  5 C, 6 C, 7 C,  8 C, 0 C, 1 C,  2 C, 3 C, 4 C,
25  8 C, 0 C, 1 C,  2 C, 3 C, 4 C,  5 C, 6 C, 7 C,
26
27  \ Hole Element in Zeile i und Spalte j von Matrix M
28  : IJ@ ( i j -- a[ij] ) SWAP 9 * + M + C@ ;
29
30  \ Speichere Element in Zeile i und Spalte j nach Matrix M
31  : IJ! ( a[ij] i j -- ) SWAP 9 * + M + C! ;
32
```

```
33  \ Addiere n modulo 9 zu saemtlichen Elementen von Matrix M
34  : ADDn ( n -- )
35      51 0 DO DUP M I + C@ + 9 MOD M I + C! LOOP DROP ;
36
37  \ INDEX
38  VARIABLE IX
39
40  \ Vertausche willkuerlich 2 Spalten einer Laengsdreierreihe in Matrix M
41  : XCHJ ( -- ) IX @ C@ 9 0
42     DO DUP 9 MOD DUP DUP 3 MOD 0> IF 1 - ELSE 1+ THEN
43       2DUP I SWAP IJ@ I ROT IJ@ ROT I SWAP IJ! I ROT IJ!
44     LOOP DROP ;
45
46  \ Vertausche Zeile 3 und 5 und Zeile 7 und 8 in Matrix M
47  : XCHI ( -- ) 9 0
48     DO 3 I IJ@ 5 I IJ@ 3 I IJ! 5 I IJ! 7 I IJ@ 8 I IJ@ 7 I IJ! 8 I IJ!
49     LOOP ;
50
51  \ ASCII --> Ziffernausgabe
52  : ZIFF ( n -- ) DUP 8 U> IF DROP SPACE SPACE ELSE 1+ . THEN ;
53
54  \ IX um 5 weitersetzen
55  : I5+! ( -- ) IX @ DUP 600 > IF 300 - THEN 5 + IX ! ;
56
57  \ Zufaellige Auslassungen
58  \ Etwa 27d Elemente bleiben als Vorgabe stehen. Breite Streuung!
59  : RAND ( n -- m ) IX @ C@ 3 MOD IF DROP 20 THEN I5+! ;
60
61  \ Bildschirmdarstellung von geaenderter Vorgabematrix
62  : V ( -- ) CR 9 0
63     DO 9 0 DO J I IJ@      ZIFF LOOP 5 SPACES
64        9 0 DO J I IJ@ RAND ZIFF LOOP CR
65     LOOP ;
66
67  \ Neue Sudoku-Matrix mit passender Vorgabematrix
68  : S ( -- ) IX @ C@  ADDN  XCHI  XCHJ  I5+! V ;
69
70  \ Will man zu ein und derselben (Loesungs-)Matrix M verschiedene
71  \ Vorgabematrizen erzeugen, dann rufe man (beliebig oft) V auf.
72  \ Bei jedem erneuten Aufruf von V wird die bis dato erreichte
73  \ Sudoku-Matrix M beibehalten, waehrend die Vorgabematrix neu er-
74  \ zeugt wird.
75
76  \ Selbstverstaendlich kann man eine erreichte Vorgabematrix
77  \ auch per Hand ausbessern: Man uebernehme (und notiere auf
78  \ Papier) nur, sagen wir, 27d Elemente oder ergaenze (bei zu
79  \ wenigen Vorgabeelementen) Elemente durch Einfuegen von der
80  \ Sudoku-Matrix M her.
81
82  \ Man kann eine erreichte Sudoku-Matrix M auch per Hand aendern:
83  \ Dazu stehen die drei Forth-Worte  x ADDN  XCHI  XCHJ  (in
84  \ beliebiger Anwendungsreihenfolge) zur Verfuegung. x kann
85  \ eine beliebige Zahl sein.
86
87  \ Geaenderte Vorgabe-Matrizen bei gleichbleibender Sudoku-
88  \ Matrix werden durch V angezeigt. Geaenderte Sudoku-Matrizen
89  \ mit passender Vorgabe-Matrix werden durch S angezeigt.
90
91  \ Solche Tastatureingaben muessen natuerlich immer durch
92  \ [return] abgeschlossen werden, um Wirksamkeit zu erlangen.
```

Wir konstruieren Zahlen: Galois–Felder für Forth–Programmierer am Beispiel der RS–Kodierung

Jens Storjohann

Zählen und Nummerieren

Ein wesentlicher Unterschied zwischen den Zahlen, auf die wir beim Zählen stoßen, und denjenigen, die wir nur zum Nummerieren brauchen, wird nach kurzem Nachdenken klar. Beim Zählen kommen wir zunächst auf die natürlichen Zahlen $\{1, 2, \cdots\}$, beim Umkehren der Addition, dem Subtrahieren, auf die Null und die negativen Zahlen, die zusammen die ganzen Zahlen $\{\cdots, -2, -1, 0, 1, 2, \cdots\}$ bilden.

Die Addition und die Multiplikation entsprechen Operationen, die mit unterscheidbaren Gegenständen durchgeführt werden, wie Zusammenfassen oder Zusammenfassen mehrerer Mengen mit einheitlicher Anzahl von Gegenständen.

Aber Nummern tragen keine nahe liegende Arithmetik in sich. Wenn die Addition zweier Hausnummern die Nummer eines anderen Hauses ergibt, ist dies eher Zufall und Willkür als Widerspiegelung eines Zusammenhanges. Bei einem Wechsel des Schemas, z. B. einer anderen Richtung der Nummerierung, würde man ein anderes Ergebnis erhalten.

Bei Kodierungssystemen ordnet man die zu übertragenden Zeichen in eine Reihenfolge, wie z.B. beim Alphabet ein, und überträgt die Nummern. Damit haben die zu übertragenden Nummern keine in sich liegende Arithmetik, die z. B. Regeln für ihre Addition festlegt. Aber wenn die Übertragung gestört ist, muss man sich Schemata überlegen, die durch Erzeugung von Zusatzinformationen eine sichere Datenübertragung gewährleisten. Dies würde man am liebsten durch arithmetische Operationen mit den zu übertragenden Nummern bewirken.

Aber in der Computertechnik sind die natürlichen Zahlen $\{1, 2, 3, ...\}$ und die mit ihnen verbundene Arithmetik nur unvollkommen zu behandeln. Denn nur ein Teil dieser Elemente aus dieser unendlichen Zahlenmenge kann auf einem reservierten Speicherplatz gegebener Größe gespeichert werden. Damit haben schon die einfachsten Operationen Addition und Multiplikation immer die Gefahr in sich, einen Speicherplatzüberlauf zu erzeugen. Außerdem zwingt uns alleine die Umkehrung der Addition, die Subtraktion, die Zahlenreihe beidseitig unendlich, d.h. auch mit negativen Zahlen anzusetzen. Damit treten bei der Multiplikation Fallunterscheidungen wie *MINUS*PLUS = MINUS* auf. Am unerfreulichsten aber ist, dass die Division manchmal ein ganzzahliges Ergebnis hat, *manchmal nicht aufgeht*, d. h. einen Rest lässt.

Weil, wie wir oben festgestellt haben, das Rechnen mit Nummern ohnehin keine in sich liegende Bedeutung hat, sind wir frei, uns eine Arithmetik zu wählen, ausschließlich unter dem Gesichtspunkt, dass sie für unsere Bedürfnisse möglichst zweckmäßig ist. All dies kann nur die Konsequenz haben, dass wir uns Zahlen verschaffen, die diese Nachteile einer für unsere besonderen Zwecke ungeeigneten Arithmetik nicht haben. *Also konstruieren wir neue Zahlen mit einer praktischen Arithmetik.*

Die Restklassen–Arithmetik

Die folgende sehr einfache Konstruktion liefert schon Zahlen mit einer geeigneten Arithmetik.

Bevor wir sie beschreiben, lohnt der Hinweis, dass im Programmierer–Jargon zwar von *Hex–Zahlen* oder *Binär–Zahlen* gesprochen wird, dies aber die bekannten natürlichen oder ganzen Zahlen sind, die nur abweichend von der uns seit der Kinderzeit bekannten Dezimal–Schreibweise dargestellt sind.

Im Folgenden dagegen werden andere *neue Zahlen* beschrieben, für die wir aber zum Teil die alten uns bekannten Ziffernsymbole verwenden.

Wir beginnen mit den natürlichen Zahlen. Wenn wir uns eine Zahl wie z. B. die 3 fest vorgeben und vereinbaren, dass alle Zahlen, die bezüglich der 3 den gleichen Rest lassen, als gleich anzusehen sind, haben wir nur 3 Zahlen zu unterscheiden: 0, 1, 2. Denn z. B. die 4 ist mit der 1 gleichzusetzen und braucht deshalb nicht als eigenes Zeichen dargestellt zu werden.

Als Multiplikationsregel und Additionsregel vereinbaren wir, dass die Operationen wie gewohnt durchzuführen sind, aber vom Ergebnis nur der Rest bezüglich der 3 zu betrachten ist. Die Klasse von Zahlen, die den gleichen Rest bei Division durch eine gegebene Zahl lassen, nennt man in der Algebra Restklasse. Wir können die Zahlen 0, 1, 2 als Restklassenvertreter ansehen, und die folgende Tabelle 1 für Addition und Multiplikation als Beschreibung der Arithmetik mit Restklassen.

Tabelle 1: Addition und Multiplikation modulo 3

+	0	1	2	·	0	1	2
0	0	1	2	0	0	0	0
1	1	2	0	1	0	1	2
2	2	0	1	2	0	2	1

Wir erkennen auch, dass jede Division außer der durch 0 möglich ist. Dies sieht man daran, dass in jeder Zeile und jeder Spalte der Multiplikationstabelle, wenn man die Multiplikation mit der Null außer Acht lässt, jede Zahl genau einmal vorkommt. Unser nächster *Versuch*

arbeitet mit den Restklassen, die bei Division durch 4 entstehen.

Tabelle 2: Addition und Multiplikation modulo 4

+	0	1	2	3	·	0	1	2	3
0	0	1	2	3	0	0	0	0	0
1	1	2	3	0	1	0	1	2	3
2	2	3	0	1	2	0	2	0	2
3	3	0	1	2	3	0	3	2	1

Hier haben wir *eine Arithmetik, die nicht zweckmäßig ist,* erhalten. Denn wenn ein Produkt aus zwei Zahlen, die beide von 0 verschieden sind, 0 ergibt, wie oben 2*2=0, heißt das, dass beispielsweise durch 2 nicht sinnvoll zu dividieren ist.

Man erkennt, dass bei einer Restklassen–Arithmetik nach einer *Primzahl* dieses Problem nicht auftritt. Damit wissen wir, dass wir eine ganze Klasse von neuen Zahlen erfunden haben, die endliche Körper oder Galois–Felder genannt werden und mit den Symbolen $GF(p)$, $F(p)$ oder F_p bezeichnet werden. Hierbei steht p für eine beliebige Primzahl.

Wir stellen die Eigenschaften dieser Zahlen kurz zusammen. Es gilt

Kommutativgesetz der Addition $a + b = b + a$

Kommutativgesetz der Multiplikation $a \cdot b = b \cdot a$

Assoziativgesetz der Addition $(a+b)+c = a+(b+c)$

Assoziativgesetz der Multiplikation
$a \cdot (b \cdot c) = (a \cdot b) \cdot c$

Distributivgesetz $a \cdot (b + c) = a \cdot b + a \cdot c$

Existenz der Null $a + 0 = a$

Existenz der Eins $1 \cdot a = a$

Existenz eines Negativen zu jedem Element
$-a$ mit $a + (-a) = 0$

Existenz eines Inversen zu jedem Element
(außer der Null) $aa^{-1} = 1$

Alle diese allgemeinen Gesetze kennen wir aus der Bruchrechnung schon seit der Schulzeit. Wir haben sogar darüber hinausgehend ziemlich komplizierte Algorithmen für Operationen mit Brüchen gelernt, mit denen beispielsweise zwei Brüche addiert werden und als gekürzter Bruch dargestellt werden können. Bei der Restklassenarithmetik geht jede Division auf, deshalb benötigt man keine Brüche. Dies gilt auch für die später eingeführten endlichen Körper mit p^n Elementen $F(p^n)$. Wir brauchen also hier keine solch komplizierten Algorithmen zu betrachten und können Rechenergebnisse entweder in einer Tabelle nachsehen oder sie berechnen, indem wir das ursprüngliche einfache Bildungsgesetz heranziehen. Auch beim Implementieren dieser Rechnung auf einem Computer haben wir diese beiden Möglichkeiten. Beides ist einfacher als das, was wir in der Schule mit 12 Jahren gelernt haben.

Ich habe mir die Aufgabe gestellt, *ohne Tabellen und ohne Probierverfahren* die elementare Arithmetik der Galois–Felder $F(p)$ und einer weiter unten erklärten Erweiterung dieser Zahlenbereiche zu programmieren, natürlich in Forth.

An den Programmtexten für die Arithmetik von $F(p)$ erkennt man, dass Addition, Multiplikation und Subtraktion begrifflich und von der Programmierung her einfach durchzuführen sind. Nur die Division ist aufwändig. Die Idee hinter der Division ist ähnlich dem Verfahren, das man für Dezimalzahlen in der Schule lernt. Wir nutzen aus, dass die Multiplikation in $F(p)$ durchgeführt werden kann, indem die bekannte Multiplikation der ganzen Zahlen ergänzt wird durch eine Restbildung modulo p.

Damit können wir auch die Division durchführen, indem wir die bekannte Division ganzer Zahlen verwenden und dem Dividenden oder seinem abgearbeiteten Rest genügend oft ein p dazu addieren.

Ein Beispiel erläutert das. Wir bilden $2/5$ in der Restklassenarithmetik modulo 7. Im Folgenden steht der Doppelpunkt für die gewöhnliche Division ganzer Zahlen.

$$2 : 5 = 0 \text{ Rest } 2.$$

Also bilden wir

$$(2 + 7) : 5 = 9 : 5 = 1 \text{ Rest } 4.$$

$$(4 + 7) : 5 = 11 : 5 = 2 \text{ Rest } 1.$$

$$(1 + 7) : 5 = 8 : 5 = 1 \text{ Rest } 3.$$

$$(3 + 7) : 5 = 10 : 5 = 2 \text{ Rest } 0.$$

Also haben wir als Quotienten

$$2/5 = 0 + 1 + 2 + 1 + 2 = 6 \text{ modulo } 7.$$

Probe ergibt

$$6 \cdot 5 = 30 = 2 \text{ modulo } 7.$$

Erweiterung der Restklassenarithmetik — die endlichen Körper $F(p^n)$

Wir haben nun auf eine sehr einfache Weise für jede Primzahl p endliche Körper mit p Elementen konstruiert. Aber der Versuch, auf die gleiche Weise, nämlich durch *Rechnen modulo 4*, beispielsweise einen Körper $F(4)$ zu erzeugen, ist gescheitert.

Es ist aber mit einer Verallgemeinerung des gleichen Gedankens möglich. Wir betrachten Polynome, d. h. formale Ausdrücke der Form

$$P(x) = c_0 + c_1x^1 + c_2x^2 + \cdots + c_{n-1}x^{n-1}$$

.

Wenn wir für die Koeffizienten Werte aus dem endlichen Körper $F(p)$ wählen, haben wir für jeden der n Koeffizienten p Möglichkeiten. Also gibt es p^n solcher Polynome. Die Größe x hat hier die Bedeutung eines Platzhalters. Wir bezeichnen den höchsten auftretenden Exponenten von x als *Grad* eines Polynoms.

Es fehlt nur noch eine geeignete Arithmetik für solche Polynome. Die Addition liegt nahe:

$$\begin{aligned} P(x) &= c_0 + c_1x^1 + c_2x^2 + \cdots + c_{n-1}x^{n-1}. \\ Q(x) &= d_0 + d_1x^1 + d_2x^2 + \cdots + d_{n-1}x^{n-1}. \\ P(x) + Q(x) &= (c_0 + d_0) + (c_1 + d_1)x^1 + (c_2 + d_2)x^2 \\ &\quad + \cdots + (c_{n-1} + d_{n-1})x^{n-1}. \end{aligned}$$

Aber es führt aber die übliche Multiplikation von zwei Polynomen mit dem Grad k, so wie man es in der Schule lernt, immer zu einem Polynom mit Grad $2k$.

Ein Beispiel illustriert das:

$$\begin{aligned} P(x)Q(x) &= (c_0 + c_1x^1)(d_0 + d_1x^1) \\ &= c_0d_0 + (c_0d_1 + c_1d_0)x + c_1d_1x^2. \end{aligned}$$

Damit haben wir wieder das Problem, dass bei naiv durchgeführter Arithmetik der Speicher überläuft, weil bei der Multiplikation Polynome beliebigen Grades entstehen können.

Man kann vermuten, dass die Lösung dieses Problems auch wieder durch eine Restbildung, aber nicht modulo einer Zahl, sondern einem Polynom erfolgen kann. Es kann gezeigt werden, dass dies genau dann zu einem Körper führt, wenn das Polynom, das man zur Restbildung heranzieht, analog zu einer Primzahl, nicht zerlegbar in ein Produkt von anderen Polynomen ist. Die gebräuchliche Bezeichnung für diese Eigenschaft eines Polynoms lautet *irreduzibel*.

Die Konstruktion von $F(2^2)$ als Beispiel

Wir betrachten alle Polynome mit Koeffizienten aus $F(2)$. Das Polynom $P(x) = 1 + 1x + 1x^2$ ziehen wir heran, um modulo diesem Polynom Reste zu bilden. Wir können zeigen, dass dieses Polynom irreduzibel, also für unsere Aufgabe geeignet ist, weil es nicht als Produkt von Polynomen mit Koeffizienten in $F(2)$ zerlegbar ist.

Denn es könnte nur ein Produkt zweier linearer Polynome von der Art $(x + a)(x + b)$ sein. Jede Annahme von Werten für a und b, wobei nur 0 und 1 möglich sind, führt zu einem anderen Produkt als dem oben gegebenen Polynom.

Ein Beispiel zeigt wie die Multiplikation zweier Polynome P_1 und P_2 durchzuführen ist:

$$P_1(x) = 1 + 1x$$
$$P_2(x) = x$$
$$P_1(x) \cdot P_2(x) = (1 + 1x) \cdot (1x) = 1x + 1x^2$$

Nun ist

$$\begin{aligned} (1x + 1x^2) : P(x) &= (1x + 1x^2) : (1 + 1x + 1x^2) \\ &= 1 \text{ Rest } 1x + 1x^2 - (1 + 1x + 1x^2) \end{aligned}$$

Für den Rest erhalten wir

$$1x + 1x^2 - (1 + 1x + 1x^2) = -1 = 1$$

Damit erhalten wir als Multiplikationsergebnis in $F(2^2)$

$$(1 + 1x) \cdot (x) = 1 \text{ modulo } (1 + 1x + 1x^2).$$

Dabei haben wir benutzt, dass in F(2) jede Zahl gleich ihrer negativen ist, was symbolisch durch $+ = -$ beschrieben wird. Mit einigen ähnlichen Rechnungen gewinnen wir eine Additions- und Multiplikationstabelle für $F(2^2)$. Darin folgen wir einer Konvention und benutzen das Symbol a statt x.

Tabelle 3: Addition und Multiplikation für $F(2^2)$

+	0+0a	1+0a	0+1a	1+1a
0+0a	0+0a	1+0a	0+1a	1+1a
1+0a	1+0a	0+0a	1+1a	0+1a
0+1a	0+1a	1+1a	0+0a	1+0a
1+1a	1+1a	0+1a	1+0a	0+0a

·	0+0a	1+0a	0+1a	1+1a
0+0a	0+0a	0+0a	0+0a	0+0a
1+0a	0+0a	1+0a	0+1a	1+1a
0+1a	0+0a	0+1a	1+1a	1+0a
1+1a	0+0a	1+1a	1+0a	0+1a

Diese Tabelle lässt sich unmittelbar einleuchtend in eine Arithmetik mit Nummern übersetzen

Tabelle 4: Addition und Multiplikation für $F(2^2)$ durchnummeriert

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Didaktische Bemerkung

Mathematiker sind vertraut mit einem abstrakten Begriff von Zahlen, so dass die Einführung neuer Zahlen für sie Routine ist. Für Anwender, die sich daran gewöhnt haben, dass sie nie über die in ihrem Taschenrechner implementierte Begriffswelt hinausgehen müssen, bedarf es einer geistigen Lockerung, solche Gebilde als Zahlen anzuerkennen. Das eigentliche Problem liegt darin, dass wir uns ungern damit abfinden, dass es Zahlen gibt, von denen wir noch nie etwas gehört haben, obwohl wir in der Schule gut aufgepasst haben und den Alltag meistern. Wir müssen also eher *entlernen* als lernen. Beim durchschnittlich trinkfesten erwachsenen Mann konnte ich empirisch nachweisen, dass die Einnahme von ca. zwei Gläsern Wein das Vertrautwerden mit diesen Zahlen signifikant erleichtert. Der Wechsel in der Symbolik von x zu a hilft psychologisch dabei, weil x gewohnheitsmäßig mit unbekannten Größen assoziiert wird.

Grundkörper und Erweiterungskörper

Die beschriebenen Gebilde werden als Galois–Felder (Galois fields) oder endliche Körper bezeichnet. Die $F(p)$ werden auch als Grundkörper der Erweiterungskörper $F(p^n)$ bezeichnet. Man kann beweisen, dass alle endlichen Körper wie hier beschrieben konstruiert werden

können. Auch kann man zeigen, dass es im abstrakten Sinne genau einen endlichen Körper für ein Parameterpaar (p, n) gibt.

Implementierung in Forth

Eigener Stapel für die Zahlen aus $F(p^n)$

Für die eigentlich interessierenden Daten, nämlich Zahlen aus $F(p^n)$, wird, außer bei sehr kleinem n, ein eigener Datenstapel, ähnlich wie für Gleitkommazahlen üblich, eingerichtet. In Hilfsrechnungen treten Zahlen aus $F(p)$ auf. Diese lassen sich sinnvoll auf dem Parameterstapel bearbeiten.

Auch für $n = 2$ kann man mit etwas Stapelmanipulationen und Benutzung des Rücksprungstapels die Arithmetik ohne eigenen Stapel durchführen. Damit sind die abgedruckten Programme im Prinzip wiedereintrittsfest, weil sie auf keine eigenen Variablen schreibend zugreifen.

Festlegen der Arithmetik

Wir haben zwei freie Parameter, die es uns erlauben, einen endlichen Körper auszuwählen. Wenn man seine Wahl von p und n trifft, muss ein in $F(p)$ irreduzibles Polynom vom Grade n (aus Tabellen) bekannt sein. Es ist aber auch möglich, geratene Polynome auf Irreduzibilität zu prüfen oder alle geeigneten systematisch aufzustellen und zu prüfen.

Anwendung auf Kodierung

Mit den vorangegangenen Erklärungen ist das Prinzip der fehlerkorrigierenden Kodierung mit Hilfe endlicher Körper leicht zu schildern: Aus den zu übertragenden Zahlen m_i werden unabhängige einzelne Blöcke $m_0, m_1, \cdots, m_k$ herausgegriffen und als Koeffizienten einer Folge von Polynomen $M_l(x)$ vom Grade k aufgefasst. Die Werte dieser Polynome für vereinbarte Argumente x_k werden als Prüfinformationen mit übertragen.

Weil die Arithmetik dieser Zahlenbereiche im abstrakten Sinne wie Bruchrechnung oder Rechnen mit reellen Zahlen abläuft, können bekannte mathematische Formeln herangezogen werden, um beispielsweise fehlerhaft übertragene Daten zu rekonstruieren.

Ein Beispiel zeigt das: Wir verwenden Blöcke aus zwei Zahlen m_0, m_1. Das Polynom $M(x) = m_0 + m_1 x$ werten wir für x_1 und x_2 aus und erhalten M_1 und M_2. Daraus rekonstruieren wir m_0 und m_1 durch Interpolation nach Lagrange.

$$M(x) = M_1 \frac{x - x_2}{x_1 - x_2} + M_2 \frac{x - x_1}{x_2 - x_1}$$

$$m_o = M_1 \frac{(-x_2)}{x_1 - x_2} + M_2 \frac{(-x_1)}{x_2 - x_1}$$

$$m_1 = M_1 \frac{1}{x_1 - x_2} + M_2 \frac{1}{x_2 - x_1}$$

Man kann nachvollziehen, dass mit dieser Kodierung, die genau so viel Bandbreite beansprucht wie eine doppelte Übertragung der ursprünglichen Nachricht, alle einzelnen Fehler in einem Block korrigiert und zwei Fehler entdeckt werden können. Eine doppelte Übertragung kann nur einzelne Fehler entdecken, aber nicht korrigieren. Zwei Fehler in einem Block können unentdeckt bleiben.

Für praktische Anwendungen verwendet man Dekodierungs–Algorithmen, die effektiver, aber auch schwieriger zu verstehen sind. Die grundlegende Idee wurde im Jahre 1960 von Reed und Solomon veröffentlicht, blieb aber lange Jahre mangels geeigneter Hardware und schneller Algorithmen zum Dekodieren eine akademische Lösung, von der man glaubte, dass sie nie eingesetzt werden würde. Heute ist Reed–Solomon–Kodierung ein Standard–Verfahren, z. B. für CDs.

Für das effektive Rechnen mit Zahlen aus Galois–Feldern verwendet man auf Tabellen basierende schnelle Algorithmen, wie z. B. auf dem Zech–Logarithmus beruhende, die hier nicht behandelt werden sollen.

Imaginäre Einheit und galoisionäre Einheit

Für Anwender in Technik und Naturwissenschaften werden die komplexen Zahlen oft so eingeführt:

- es gibt eine imaginäre Einheit i, die der Gleichung $i^2 = -1$ gehorcht.
- wer damit nicht rechnen kann, fällt durch die Prüfung.

Ähnlich hätte ich auch für den Fall $F(2^2)$ schreiben können:

- $2x = 0$ oder $x = -x$ für alle x
- es gibt eine galoisionäre Einheit a, die der Gleichung $a^2 = -a - 1 = a + 1$ oder $a^2 + a + 1 = 0$ gehorcht.
- wer damit nicht rechnen kann, muss für seine Programmieraufgaben Lösungen mit geschobenen oder sonst wie manipulierten Bits erdenken.

Forth als universelle Algebra

Der Verfasser hat vor etwa 25 Jahren Vorlesungen über Algebra bei Prof. Ernst Witt gehört. Er gehörte zu den bedeutendsten Algebraikern des zwanzigsten Jahrhunderts und war auch einer der ersten Mathematiker, der Computer zu Forschungszwecken in der höheren Algebra einsetzte. Er machte damals die Bemerkung, dass Forth eine universelle Algebra sei. Das heißt, dass Forth insbesondere durch seine Konvention der Nutzung des Stapels für die Übergabe der Eingangsgrößen und der Ergebnisse geeignet ist, beliebige mathematische Operationen mit beliebigen Zahlen von Argumenten und Ergebnissen auf eine einheitliche Weise durchzuführen.

Bei einem Zusammenspiel einer Computersprache mit einem *realen Computer* sind natürlich Einschränkungen

solch allgemeiner Aussagen, insbesondere durch den endlichen Speicherplatz, zu bemerken. Aber bei Forth ist dies im Wesentlichen nur der insgesamt für den Stapel reservierte Platz. Die vorliegenden Programme demonstrieren diese Eignung von Forth, Algebren mit geringem Aufwand zu realisieren, nur mit für Algebraiker sehr durchsichtigen Gebilden.

Biografische Notiz

Der Mathematiker Evariste Galois lebte als Zeitgenosse Napoleons von 1811 bis 1832. Er starb in jungen Jahren an den Folgen eines Duells. Die hier beschriebenen endlichen Körper sind vom ihm entdeckt. Die Theorie über die Bedingungen für die Auflösbarkeit algebraischer Gleichungen durch Wurzelausdrücke, an deren Anfang er steht, gehört zu den beeindruckendsten Leistungen der Mathematik.

Quellen und Literaturempfehlungen

Eine für angehende Mathematiker geeignete Einführung findet sich in den bekannten Büchern von van der Waerden[1], die in vielen Auflagen und Übersetzungen erschienen sind, oder im Netz [2]. Die Originalarbeit [3] über die Kodierung mit Hilfe endlicher Körper ist nun schon fast ein halbes Jahrhundert alt. Aber trotzdem lässt sich bis heute ein Strom von darauf aufbauenden Arbeiten, die sich mit der Implementierung und vielen weiteren praktisch relevanten Problemen befassen, beobachten. Als Beispiel sei eine Diplomarbeit [7] genannt, für deren Zusendung ich mich bei Herrn Garz bedanke.

Forth und Reed–Solomon sind schon mehrfach in Verbindung gekommen. Der Artikel in der VD von Rafael Deliano [6] war für mich der Anstoß, einen Artikel zu entwerfen, in dem möglichst allgemein verständlich Galois–Felder erklärt werden.

Ich danke Herrn Behringer für den Hinweis auf G. Dixons Artikel [4] und [5], der mir bei der Abfassung der vorliegenden Notiz entgangen war.

Literatur

[1] B.L. van der Waerden, Algebra I, Achte Auflage der Modernen Algebra, Springer, 1971

[2] Wikipedia deutsch unter dem Stichwort *Endlicher Körper*

[3] Reed, I. und G. Solomon, Polynomial codes over certain finite fields, Journal of the Society for Industrial and Applied Mathematics [SIAM J.], 8 (1960), 300–304.

[4] Dixon, G., Reed–Solomon–Fehlerkorrektur, *Vierte Dimension* Das Forth Magazin, Ausgabe 4/1999, 13–16 (Übersetzt von Fred Behringer, das Original ist in Forth Dimensions Band XX, Nr. 4 erschienen)

[5] Dixon, G., Reed–Solomon–Fehlerkorrektur Teil 2, *Vierte Dimension* Das Forth Magazin, Ausgabe 4/1999, 34–36 (Übersetzt von Fred Behringer, das Original ist in Forth Dimensions Band XX, Nr. 5,6 erschienen)

[6] Deliano, R., Arithmetik im Galoisfeld, Forth Magazin *Vierte Dimension*, Nr. 3/4/ (2005), 24–26

[7] Bernhard Garz, Decodierungsalgorithmen für Reed–Solomon–Codes, 1995 Diplomarbeit Universität Karlsruhe — Fakultät für Informatik

```
1   ( Arithmetik in Galois Feldern, Grundkoerper F_p )
2   : Restklassen ;
3   variable F_pp
4   2 F_pp ! \ Vorbesetzung  mit 2
5   : F_p+ ( a b -- a+b ) + F_pp @ mod ;
6   : F_p- ( a b -- a-b ) - F_pp @ mod ;
7   : F_p* ( a b -- a*b ) * F_pp @ mod ;
8   : F_p/ ( a b -- a/b )
9      swap 0 swap ( b 0 a) ( oder b q r )
10     begin
11       rot dup >r       \ q r b
12       /mod             \ q r delta-q
13       rot +            \ r q
14       r> swap rot dup  \ b q r r
15       F_pp @ + swap    \ b q r' r
16       0=
17     until
18     drop swap drop ;
19
20  \ Einrichtung des gesonderten F_p^n Stapels entfaellt hier
21  \ ************ weil nur fuer F_p^2 ******************
22
23  2             constant F_p^nn
24  F_p^nn cell * constant F_p^2#bytes \ Speicherplatz fuer eine Zahl aus F_p^2
25
26  \ ------------------------------------------------------------
```

```
27  \ Arithmetik fuer Galois-Felder F_p^n auf dem Parameterstapel
28  \ nur Spezialfall n=2 und Polynom a^2+a+1=0
29  \ dieses Polynom ist irreduzibel fuer z. B. p= 2, 5
30  : F_p^2+  ( a0 a1 b0 b1 -- a0+b0 a1+b1 )
31     swap >r f_p+ swap r> f_p+ swap ;
32  : F_p^2-  ( a0 a1 b0 b1 -- a0-bo a1-b1 )
33     swap >r f_p- swap r> f_p- swap ;
34   : F_p^2*   ( a0 a1 b0 b1 -- c0 c1 )
35                              \ c0 = a0*b0 - a1*b1
36                              \ c1 = a1*b0 + a0*b1 - a1*b1
37   dup >r swap dup >r  \ a0 a1 b1 b0                    |R        b0 b1
38   rot dup >r f_p*     \ a0 b1 a1*b0                    |R     a1 b0 b1
39   rot dup >r          \ b1 a1*b0 a0                    |R  a0 a1 b0 b1
40   rot f_p* f_p+       \ a1*b0+a0*b1                    |R  a0 a1 b0 b1
41   r> r> r> rot        \ a1*b0+a0*b1 a1 b0 a0           |R           b1
42   f_p* swap r> f_p*   \ a1*b0+a0*b1 a0*b0 a1*b1        |R
43   dup >r f_p-         \ a1*b0+a0*b1 a0*b0-a1*b1        |R        a1 b1
44   swap r> f_p- ;      \ a0*b0-a1*b1 a1*b0+a0*b1-a1*b1 |R
45
46  : konj ( a0 a1 -- a0+a1 a1 )
47   dup rot f_p+ swap ; \ nur falls Polynom  a^2+a+1=0  korrekt
48
49  : F_p^2/   ( a0 a1 b0 b1 - c0 c1 )
50  \ aehnlich wie durch Erweiterung mit dem konjugiert komplexen Nenner
51  \ machbar, Als Uebungsaufgabe empfohlen.
52                              ;
53  \ ----------------------------------------------------------------
54  \ Variablen und Konstanten
55  : F_p^2variable
56    create [ F_p^2#bytes ] literal allot ;
57  : F_p^2!
58    dup [ F_p^2#bytes 1- ] literal + swap
59    do
60       i ! 4
61    +loop ;
62
63  : F_p^2@ dup 1- swap [ F_p^2#bytes 2 / ] literal +
64    do
65      i @ -4
66    +loop ;
67   f_p^2variable test
```

Das Forth–e.V.–Wiki

Bernd Paysan

Die Website der Forth–Gesellschaft stellt Hilfsmittel zur Kommunikation zwischen Vereinsmitgliedern und anderen Forth-Begeisterten zur Verfügung. Eines dieser Hilfsmittel ist das Wiki. Dieser Artikel soll die Bedienung und den Zweck des Wikis beschreiben, und auch den Unterschied zum Blog (der Struktur der Hauptseite) aufzeigen.

Einleitung

Die Einstiegsseite der Forth–Gesellschaft ist mit Geeklog gemacht. Geeklog ist ein Blog, also ein primär für Meldungen und Nachrichten gedachtes Medium. Neue Nachrichten landen oben auf der Seite, ältere Nachrichten verschwinden schnell aus dem Blickfeld und irgendwann auch von der Seite.

Ein Blog ist also eine stapelartige Struktur, gemacht für temporäre Einträge, nicht für dauerhafte Strukturen. Zwar enthält Geeklog auch "static pages," also Seiten, die nicht so schnell verschwinden. Diese übersichtlich miteinander zu verlinken ist aber nicht einfach — das System ist eben nicht dafür gedacht.

Deshalb habe ich Ende Februar ein Wiki (DokuWiki) installiert. Artikel in einem Wiki stehen unter einen „Stichwort," das Wiki stellt also eine Art Wörterbuch dar. Wikis sind damit für themenorientierte Dokumentation geeignet. Verschiedene Themen–Komplexe kann man trennen, indem man "Namespaces," also Vokabulare verwendet.

Wie beim Geeklog gilt auch für's Wiki: Kollaboratives Arbeiten ist möglich und *erwünscht*.

Wir hatten ja schon einmal ein Wiki auf der Forth–eV–Seite, und damit eher gemischte Erfahrungen gemacht. Es sind damals nicht gerade viele Beiträge zusammengekommen, die von Vandalen aber gleich wieder gelöscht wurden. Deshalb nutzen wir die Zugangskontrolle des neuen Wikis und erlauben nur Beiträge angemeldeter Benutzer. Derzeit kann sich jeder selbst anmelden, sollte das zu einem Problem werden, können wir auch das manual machen — Wikispammer und andere Vandalen sollten somit keine Chance haben.

Damit mehr Inhalte kommen, muss der Zweck und die Benutzung des Wikis auch allgemein bekannt sein, und genau diesem Zweck dient dieser Artikel.

Themen des Wikis

Im Moment haben wir folgende Themen im Wiki:

- Forth–Wörter und ihre Definitionen
  - Forth Target-, Cross- oder Meta-Compiler? Die verschiedenen Möglichkeiten ein Forthsystem zu erzeugen hat Ullrich Hoffman aufgelistet.
  - Threaded Code Die verschiedenen Möglichkeiten eine verkettete Liste von Vektoren (Adressen) in einem Forth System anzulegen.
  - Inner Interpreter und der Instruction Pointer in der virtuellen Forthmaschine.
  - Forth Kernel Words for embedded Systems
  - ANS American National Standard for Information Systems, Programming Languages: Forth
  - Interessante Wörter aus Newsgruppen und andere Fundstücke
- Projekte und ihr Stand
  - R8C — Das R8C–Gforth
  - Roboter — Robotik–Projekte
    - Forth in Balance
    - Forth spielt Solitär
    - Lego Forth
  - PIC — PIC–basierte Forthe
- Literatur–Liste
- Quiz — vom Ruby Quiz inspiriert
  - LCD Numbers — ein Quiz–Element aus dem Ruby Quiz für Forth adaptiert
- Beispielhafte Forthprogramme
  - Tetris for Terminals
  - Links auf andere Kollektionen
- Rätselhaftes: Forth–Programme, die man nicht auf Anhieb versteht
  - r> >r nicht in derselben Colon–Definition?
- Dies und Das
  - SMD–Chips einlöten

Bedienung des Wikis

Die Bedienung des Wikis ist eigentlich ganz einfach. Oben und unten an jeder Seite sind Bedienknöpfe. Wenn man das Wiki nicht nur lesen, sondern auch bearbeiten will, muss man sich zunächst

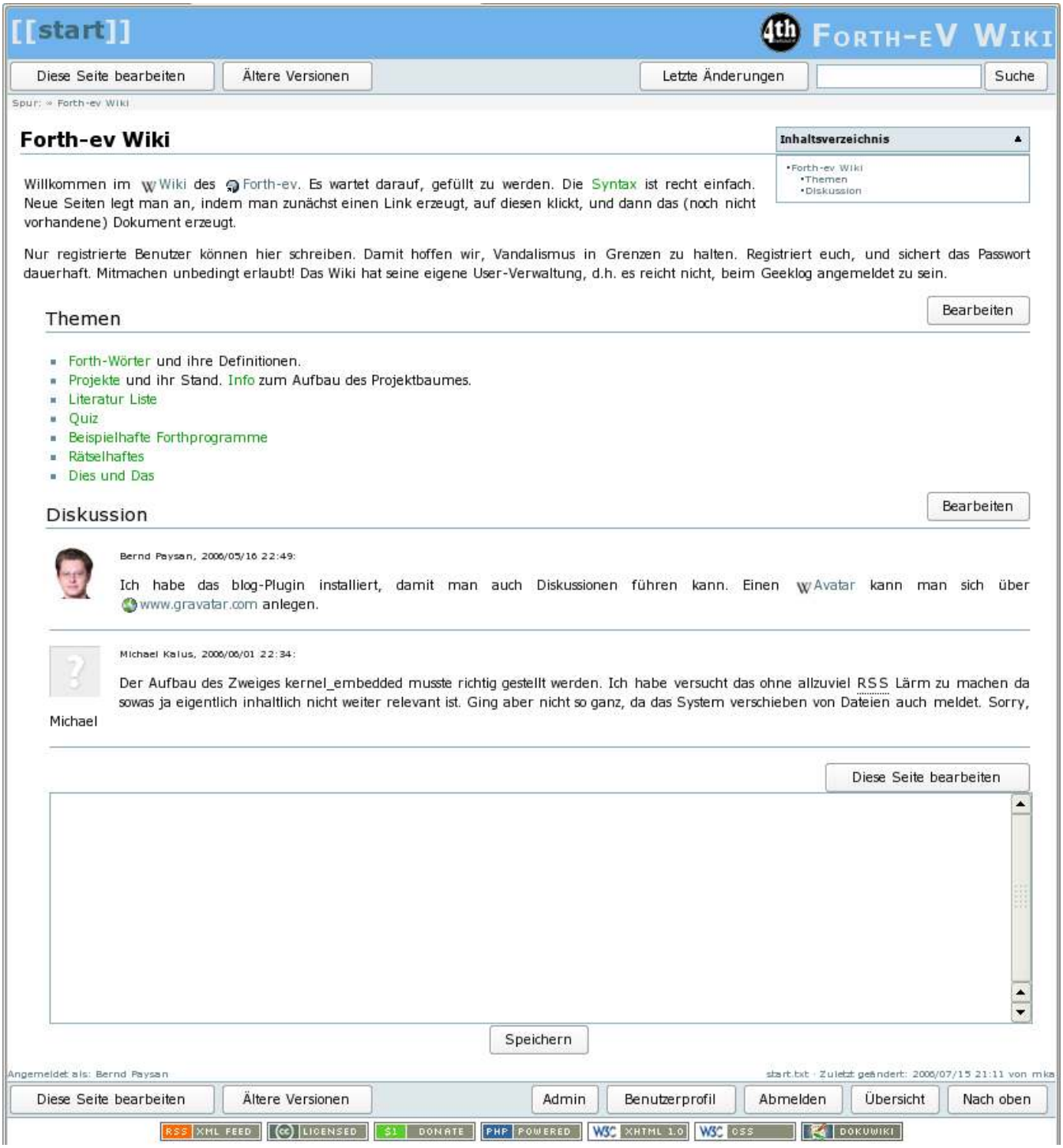


Abbildung 1: Die Einstiegsseite des Wikis

Anmelden

Hat man noch keinen Benutzernamen und Passwort, registriert man sich einfach. Dazu wird ein Benutzername, ein voller Name und eine E–Mail–Adresse verlangt. An diese E–Mail–Adresse bekommt man sein Passwort geschickt — auch in dem Fall, dass man das Passwort vergessen hat.

Hat man das Passwort, meldet man sich am besten dauerhaft an — dann speichert der Browser einen Cookie. Im Benutzerprofil kann man das vom System vergebene Passwort auch noch ändern.

Navigation

Neben den bekannten Hyperlinks gibt's im Wiki eine Übersicht, die alle Seiten in Namespaces gruppiert anzeigt. Im Eingabefeld rechts oben kann man auch ganz gezielt suchen. Die „Spur"–Zeile oben zeigt einem den Pfad, den man durch's Wiki genommen hat, und führt auf vorher besuchte Seiten zurück. Ein Klick auf den Seitennamen zeigt, welche andere Seiten auf diesen verweisen. Über das 4th–Logo kommt man zurück zur Einstiegseite der Forth–Gesellschaft, und mit dem Text–Link daneben („Forth–eV Wiki") zur Startseite des Wikis.

Änderungen verfolgen

Das Wiki ändert sich ständig — um den Überblick nicht zu verlieren, was sich als letztes geändert hat, unterstützt es RSS–Feeds (allgemein und auch Namespace–spezifische). Der Button „Letzte Änderungen" zeigt ebenfalls, was sich gerade im Wiki tut. Ältere Versionen lassen sich anzeigen und mit der aktuellen Version vergleichen.

Diese Seite bearbeiten

Diesen Knopf findet man sowohl oben und unten links auf jeder Seite (wenn man nicht angemeldet ist, kann man hier nur den Quelltext der Seite sehen). Das Wiki verwendet eine einfache Markup–Sprache, deren wichtigste Elemente ich hier erkläre. Der Editor zeigt auch Icons für diese Elemente an, sodass man für's Formatieren eigentlich nur klicken muss.

Struktur

Abschnitte und Unterabschnitte markiert man links und rechts mit =–Zeichen. Der oberste Abschnitt mit 6, die untergeordnete mit 5, 4, 3 und so weiter. Absätze werden durch Leerzeilen getrennt.

Textattribute

Fett Mit zwei ** links und rechts vom ****fetten Text****

Kursiv Mit zwei // links und rechts vom *//kursiven Text//*

Unterstrichen Mit zwei __ links und rechts vom __unterstrichenen Text__

Schreibmaschine Mit zwei '' links und rechts vom ''Schreibmaschinen--Text''

Natürlich kann man die Attribute auch beliebig kombinieren.

Links

Links schließt man in zwei eckige Klammern ein ([[link]] oder [[link|Beschreibung]]). Interne Links sind ohne http:// und weisen auf Wiki–Seiten, externe Links mit http:// verweisen auf entsprechende externe Seiten. Das Wiki erkennt http:// aber auch direkt, ohne eckige Klammern. Namespaces des Wikis werden im Linknamen durch Doppelpunkte abgetrennt, etwa wiki:syntax oder projects:r8c. Natürlich kann man auch Bilder mit in die Link–Beschreibung einbinden.

Bilder und andere Dateien

Bilder werden mit zwei geschweifte Klammern eingeschlossen, etwa {{bild.png}}. Das Image–Icon im Texteditor öffnet einen Dialog, mit dem Bilder ins Wiki hochgeladen werden können. Neben Bildern können auch Filme, PDFs und ähnliche Dokumente hochgeladen werden.

Listen

DokuWiki unterstützt sowohl geordnete als auch ungeordnete Listen. Alle Listen–Einträge fangen mit 2 oder mehr Spaces ein (gerade Anzahl!); gefolgt von * für ungeordnete und – für geordnete Listen. Die Tiefe der Einrückung hängt von der Anzahl der Spaces durch zwei ab.

Umgewandelte Strings

DokuWiki wandelt einige Strings automatisch um. Aus ASCII–Smileys werden Icons, aus ASCII–Art–Pfeilen werden die entsprechenden Unicode–Zeichen.

Code

Code schreibt man zwischen <code> und </code> Tags. DokuWiki unterstützt Syntax Highlighting, etwa <code forth> für Forth–Quelltext.

Weitere Features

DokuWiki unterstützt auch Tabellen, ungeparste Blöcke und eine Menge Plugins. Deshalb verweise ich hier einfach auf die Online–Dokumentation unseres Wikis: http://www.forth-ev.de/wiki/doku.php/wiki:syntax

Zweck

Das Wiki soll mehrere Zwecke erfüllen:

- Zunächst wollen wir den Fortschritt von Projekten online dokumentieren. Sichtbarkeit ist alles, nur über öffentlich zugängliches Material können wir Aufmerksamkeit bekommen.
- Dann wollen wir natürlich Hintergrundwissen über Forth sammeln und zugänglich machen.
- Das Wiki ermöglicht auch einen schnellen Austausch zwischen Mitgliedern, ohne auf den Vierteljahresrhythmus der VD achten zu müssen. Ein Rückfluss ist aber jederzeit möglich.
- Zuletzt soll das Wiki eine zentrale Anlaufstelle für die Forth–Community bilden. Es ist also für alle da, die Informationen über Forthiges finden oder ins Netz stellen wollen.

Das Wiki wächst, also nutzt es. Je mehr Leute mitmachen, desto mehr Nutzen kann jeder Einzelne daraus ziehen.

Buchbesprechung: Designing Embedded Hardware

Carsten Strotmann

Als Hardware–"Greenhorn" hat es mich gefreut, auf dem Linux–Tag ein Buch zu finden, welches in die Grundlagen der Hardware–Entwicklung einführt. Und noch erfreuter war ich, ein Kapitel über Forth in diesem Buch zu finden.

"Designing Embedded Hardware" von John Catsoulis ist im Mai 2005 bei O'Reilly in englischer Sprache in der zweiten Auflage erschienen. Das Buch richtet sich vornehmlich an Leser, die bisher wenig Erfahrung mit Hardware–Entwicklung gesammelt haben. Der Autor erklärt die Grundlagen der Hardware–Entwicklung, ohne allzu oberflächlich zu bleiben. Kapitel 1 beschreibt die grundlegenden Konzepte von Computer–Hardware–Architekturen (von Neumann, Harvard), Bussysteme, Interrupts, DMA, RAM und ROM. Das anschließende Kapitel 2 gibt einen allgemeinen Überblick über die Programmierung in Assembler, mit einigen Beispielen in 68HC11 und PIC–Assembler.

Das dritte Kapitel beschäftigt sich mit Forth (Überschrift Forth/OpenFirmware). In der ersten Auflage des Buches war dieses Kapitel noch nicht enthalten, es ist in der neuen, zweiten Auflage hinzugekommen. Der Autor beleuchtet kurz die Geschichte von Forth und führt dann in die Forth–Konzepte ein (UPN, Stack). Das Erstellen neuer Wörter wird erklärt, sowie das Erstellen einfacher Datenstrukturen (jedoch kein DOES>). Der letzte Abschnitt des Kapitels beinhaltet Forth–Programmier–Richtlinien und die interaktive Hardware–Fehlersuche mit Forth.

Das folgende Kapitel 4, Überschrift "Electronics 101", bietet eine Einführung in Elektronik–Grundlagen und erklärt Dinge wie Widerstände, Dioden und Kondensatoren. Kapitel 5 liefert die Informationen für die Stromversorgung der zu erstellenden Hardware, während Kapitel 6 auf den Aufbau der Hardware eingeht, inkl. Löten.

Die folgenden Kapitel 7 bis 12 beschäftigen sich mit der Kommunikation der Hardware mit der Außenwelt und Peripherie, über SPI, I2C, serielle Schnittstelle, Infrarot, USB und Netzwerke (CAN und Ethernet).

Kapitel 13 enthält Informationen zur Verarbeitung von analogen Signalen wie Analog–Digital–Konverter, Licht– und Temperatursensoren.

Die verbliebenen Kapitel 14 bis 19 stellen verschiedene Prozessor–Architekturen für kleine Computer–Systeme vor (PIC Mikrocontroller, AVR, 68HC11, MaxQ, Motorola–68000–Serie und DSP–basierte Prozessoren.

Als Hardware–Laie hat mir dieses Buch sehr gut gefallen (nicht nur wegen des Forth–Kapitels). Es bietet eine gute Einführung in die Welt der Elektronik und der Embedded Hardware, speziell für Leser ohne Vorkenntnisse auf diesem Gebiet. Für Elektronik–Einsteiger empfehlenswert. Und schön zu wissen, dass diese Einsteiger gleich mit Forth vertraut gemacht werden.

Designing Embedded Hardware, Second Edition
by John Catsoulis
O'Reilly Media, Inc
1005 Gravenstein Highway North, Sebastopol, CA 95472
ISBN: 0-596-00755-8

Forth–Gruppen regional

Moers
Friederich Prinz
Tel.: (0 28 41) – 5 83 98 (p) (Q)
(Bitte den Anrufbeantworter nutzen!)
(Besucher: Bitte anmelden!)
Treffen: 2. und 4. Samstag im Monat 14:00 Uhr,
MALZ, Donaustraße 1
47441 Moers

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

München
Bernd Paysan
Tel.: (0 89) – 79 85 57
bernd.paysan@gmx.de
Treffen: jeden 4. Mittwoch im Monat ab 19:00. Treffpunkt ändert sich im Moment gerade, bitte auf der Web–Site nachsehen oder nachfragen!

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

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

Gruppengründungen, Kontakte

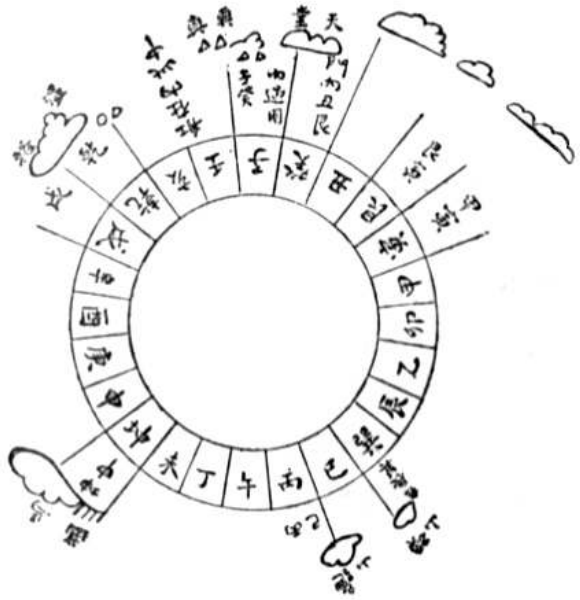
Hier könnte Ihre Adresse oder Ihre Rufnummer stehen — wenn Sie eine Forthgruppe gründen wollen.

µP–Controller Verleih

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

Spezielle Fachgebiete

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



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.

Ankündigung EuroForth 2006

Anton Ertl

Die 22. EuroForth wird vom 15. – 17. Semptember 2006 im Sidney Sussex College in Cambridge (England) stattfinden. Am Tag davor (also 14. 9. Nachmittag und 15. 9. Vormittag) findet wie letztes Jahr das Forth 200x Standardisierungstreffen statt. Die Konferenzsprache ist Englisch.

Auf der Konferenz werden wie immer Vorträge gehalten und in Workshops interessante Themen nach Wahl der Teilnehmer diskutiert.

Neben dem eigentlichen Konferenzprogramm kann man wie immer ein starkes Begleitprogramm erwarten, z. B. eine Stadtbesichtigung, ein Konferenz–Bankett, und Staken auf dem Fluss Cam. Die EuroForth in Oxford (1997) war die meistbesuchte aller Zeiten.

Call for Papers

Artikel über alle Aspekte von Forth (Anwendung, Implementation, Erweiterung, etc.) und verwandte Themen sind erwünscht.

Artikel können entweder als begutachtete (refereed) Artikel eingereicht werden (academic stream), oder als nicht begutachtete Artikel (industrial stream). Man kann natürlich auch einen Vortrag halten, ohne einen Artikel einzureichen, oder auch einfach so auf die Konferenz fahren.

Artikel sollen per Email (an Anton Ertl, `anton@mips.complang.tuwien.ac.at`) eingereicht werden, und zwar in Form von Postscript– oder PDF–Dateien. Das Format für die Endversion ist A4 mit 25mm Rand auf allen Seiten, unnummeriert.

Das Programm–Kommittee, das die Begutachtung der begutachteten Artikel vornimmt, besteht derzeit aus:

- M. Anton Ertl, TU Wien (Vorsitz)
- Phil Koopman, Carnegie Mellon University
- Jaanus Pöial, Estonian Information Technology College, Tallinn
- Bradford Rodriguez, T–Recursive Technology
- Reuben Thomas
- Sergey N. Baranov, Motorola ZAO, Russia
- Ulrich Hoffmann, Heidelberger Druckmaschinen AG

Daten

28. 6.	Einreichfrist für zu begutachtende Artikel (academic stream)
26. 8.	Benachrichtigung, ob der begutachtete Artikel angenommen wurde
4. 9.	Einreichfrist für die Endversion bzw. für nicht begutachtete Artikel
14. 9. – 15. 9.	Forth 200x Standardisierungstreffen
15. 9. – 17. 9.	EuroForth

Die Einreichfristen sind streng; zu spät eingereichte Artikel können nicht mehr bearbeitet werden.

Adressen

Aktuelle Informationen zur Konferenz finden sich auf ihrer Homepage (`http://dec.bournemouth.ac.uk/forth/euro/ef06.html`). Den Call for Papers und Informationen über das Einreichen von Artikeln findet man auf `http://www.complang.tuwien.ac.at/anton/euroforth2006/`. Informationen über Forth 200x und das Standardisierungstreffen findet man auf `http://www.forth200x.org/forth200x.html`.

Die Konferenz wird von Janet Nelson (`jen@micross.co.uk`) organisiert. Artikel sollten bei Anton Ertl (`anton@mips.complang.tuwien.ac.at`) eingereicht werden. Es gibt eine Mailing–Liste für die Konferenz (`http://groups.yahoo.com/group/euroforth/`), für die man sich über `euroforth-subscribe@yahoogroups.com` anmelden kann.