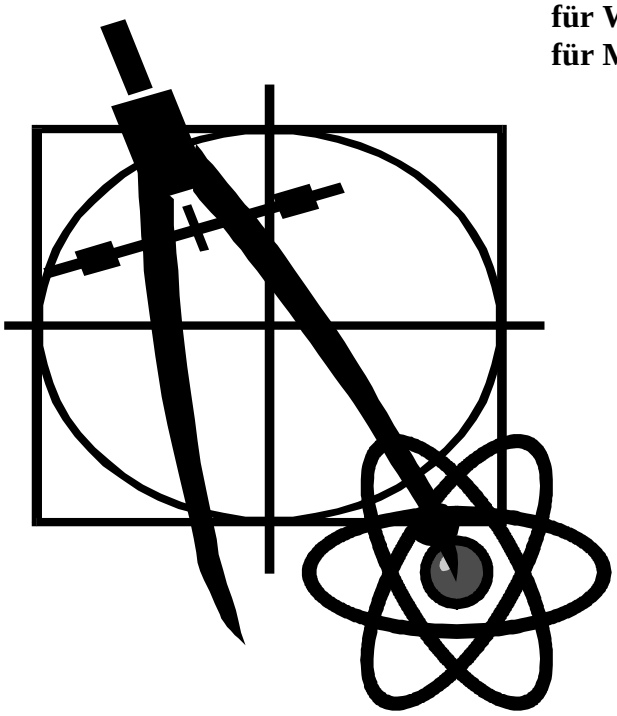
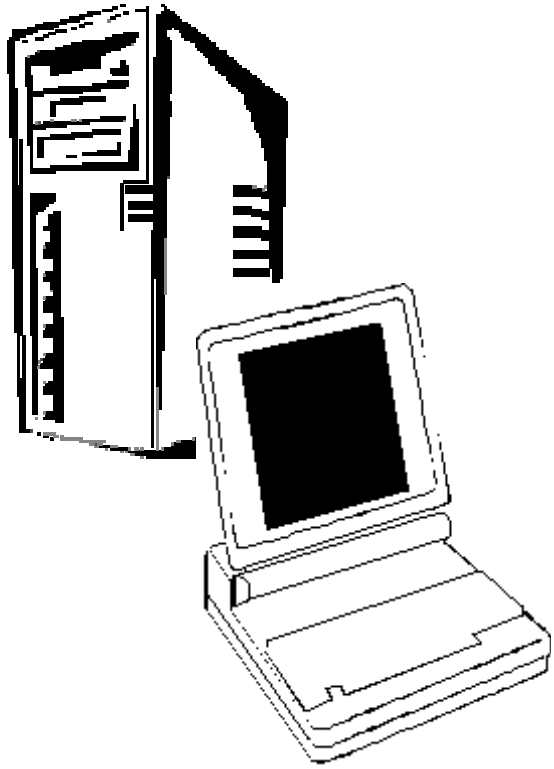
VIERTE DIMENSION

4 / 2003

Das FORTH-Magazin

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





In dieser Ausgabe:

19. Jahrgang - **Ausg. 4/2003** - 4,00 €

Organ der FORTH-Gesellschaft e.V.

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 Forthgesellschaft 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'!

Dipl.-Ing. Arndt Klingelnberg

Tel.: ++32 +87 -63 09 89 (Fax: -63 09 88)
Waldring 23, B-4730 Hauset, Belgien
akg@aachen.kbbs.org

Computergestützte Meßtechnik und Qualitätskontrolle, Fuzzy, Datalogger, Elektroakustik (HiFi), MusiCassette HighSpeedDuplicating, Tonband, (engl.) Dokumentationen und Bedienungsanleitungen.

Forth Engineering
Dr. Wolf Wejgaard

Tel.: +41 41 377 3774 - Fax: +41 41 377 4774
Neuhöflirain 10
CH-6045 Meggen http://holonforth.com

Wir konzentrieren uns auf Forschung und Weiterentwicklung des Forth-Prinzips und offerieren HolonForth, ein interaktives Forth Cross-Entwicklungssystem mit ungewöhnlichen Eigenschaften. HolonForth ist erhältlich für 80x86, 68HC11 und 68300 Zielprozessoren.

KIMA Echtzeitsysteme GmbH

Tel.: 02461/690-380
Fax: 02461/690-387 oder -100
Karl-Heinz-Beckurtz-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.: 08233-30 524 Fax: - 9971
Postfach 1173
D-86404 Mering

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

Inhalt

Diese Ausgabe der VD wird vermutlich ca. vier bis sechs Wochen nach dem Erscheinen der Druckausgabe im Internet auf der Web-Seite der Forthgesellschaft e.V. veröffentlicht.

http://www.forth-ev.de

Eine PDF-Version dieser Ausgabe wird ab dem Zeitpunkt der Veröffentlichung im Internet ebenfalls zur Verfügung stehen. Bitte wenden Sie sich hierzu über die oben angegebene Adresse an den Webmaster der Forthgesellschaft e.V. oder an die Redaktion der „Vierte Dimension“. *fep*

In der nächsten Ausgabe finden Sie voraussichtlich:

- 4 * 4; eine in Forth programmierte Lösung zum Viererproblem
- Target-Compiler: Fortsetzung
- USB-Tool Quellen

Editorial

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

Redaktion & Layout

Friederich Prinz
Hombergerstraße 335
47443 Moers
Tel.: (0 28 41) 5 83 98
E-Mail: ***VD@FORTH-EV.DE***

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluß

März, Juni, September, Dezember
jeweils in der dritten Woche

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00 € + Porto u. Verpackung

Liebe Leser,

diese Ausgabe unserer Zeitschrift erreicht die meisten von Ihnen gerade so rechtzeitig, daß Sie sich die Vierte Dimension selbst auf den Gabentisch legen können. Ich hoffe sehr, daß Sie das freut.
Daß diese Ausgabe überhaupt zustande gekommen ist, verdanken wir allesamt aber einigen wenigen, unermüdlichen Autoren. Diese haben, trotz aufreibender Geschäfte und vieler Arbeit, die Zeit aufgebracht, die es gekostet hat, einem dringenden Hilferuf meinerseits Gehör und in der Folge auch Taten zu schenken.
Sie, liebe Leser, sind es gewöhnt, an dieser Stelle Aufrufe um Artikel und beliebige Beiträge zu lesen. Und ich kann Sie alle auch dieses Mal nur eindringlich bitten, der VD genau das Material zu geben, das Sie hier abgedruckt sehen möchten. Bitte helfen Sie mir, damit ich auch im kommenden Jahr vier Ausgaben der VD für Sie zusammenstellen kann.

Das kommende Jahr 2004 ist ein ganz besonderes Jahr. Die Forthgesellschaft wird 20 Jahre alt. Das ist ein guter Grund für eine Feier. Unsere Tagung auf Fehmarn wird nicht zuletzt darum eine ganz besondere Tagung sein. Ich persönlich hoffe, möglichst viele von Ihnen auf dieser Tagung zu treffen. Informationen zum Tagungsort, zum Tagungstermin und mehr finden Sie auf der Rückseite dieser Ausgabe.

Ich wünsche Ihnen und uns allen ein gutes, friedvolles und erfolgreiches Neues Jahr, und zuvor natürlich ein schönes Weihnachtsfest.

Ihr

Friederich Prinz

Quelltext-Service

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können diese Texte auch direkt bei uns anfordern und sich zum Beispiel per E-Mail schicken lassen. Schreiben Sie dazu einfach eine E-Mail an die Redaktionsadresse. *fep*

Die Forthgesellschaft wird durch ihr Direktorium vertreten:

Prof. Dr. Fred Behringer
Dr. Ulrich Hoffmann
Dipl. Inf. Bernd Paysan

Kontakte: Direktorium@Forth-ev.de

Betreff: Leserbrief: HOLON lebt
Von: Ulrich Paul <upaul@paul.de>

Hi Fritz,

ich kenne das System von Wolf Wejgaard nicht, aber was ich dem Artikel entnehmen kann, ist eines: Er kennt die Regressionstests nicht. Das ist eine Suite von - hoffentlich - automatisierten Tests, die auf einen Teil der (oder auch der ganzen) SW angewendet wird. Witz der Sache ist einfach dieser: Die Suite hat bei einer stabilen Vorgängerversion keine Fehler festgestellt - nur dann wurde ja die Vorgängerversion ausgeliefert. Diese - identische! - Suite wird nach irgendwelchen Modifikationen wieder auf die neue SW losgelassen. Da die ja zumindest die alte Funktionalität erhalten soll, darf sie also bei diesem Test nicht versagen. So ein Regressionstest ist recht einfach zu bewerkstelligen, wird aber trotzdem oft unterlassen.

Natürlich muß dieser Regressionstest immer dann einer neuen Version angepaßt werden, wenn sich die Schnittstelle nach außen in einer inkompatiblen Weise geändert hat. So ein Fall macht u.U. sogar einen kompletten Retest - von Null weg und damit zeitaufwendig - notwendig.

Aber ich glaube, Wolfs Probleme kann man einfacher durch CVS in den Griff kriegen. Dazu braucht es kein spezielles Forth-System. CVS steht für "Concurrent Versioning System" und ist **das** System, mit dem man größere Projekte verwalten kann. Die ganze GNU-Gemeinde, nur als Beispiel, verwendet CVS um ihre tausende Projekte zu managen.

Trotzdem kann jeder User so etwas auch für sich privat einrichten. Damit hier keine Mißverständnisse aufkommen! Und es ist nicht einmal besonders kompliziert. Aber es erfordert eine gewisse Disziplin.

Ich will hier nicht einfach und blind das Wort für CVS reden, aber wer Probleme mit verschiedenen Versionen seiner SW hat, der sollte es ernsthaft in Betracht ziehen. Das gibt es kostenlos – weil unter GNU – für diverse Systeme.

Ich habe den Vorteil von CVS kennengelernt – aber, das gebe ich zu! – noch nicht – und die Betonung liegt auf "noch", bei mir eingeführt (ein "echter" Entwickler hat doch immer alles im Griff!). Vor ein paar Jahren war ich in einem Team von internen und externen Entwicklern. Sehr schnell – und wir haben nicht einmal in Forth programmiert – kamen Inkompatibilitäten zum Vorschein: A ändert etwas im Modul AM1, B reagiert darauf durch Änderungen am Modul BM1. Leider ist das nicht kompatibel mit Änderungen, die C am Modul CM1 gemacht hat. Vorher lief es, aber jetzt nicht mehr. Wer kann den ursprüglichen Zustand wieder rekonstruieren? Wir damals jedenfalls nicht, da jeder Dutzende von Änderungen an seinen eigenen Dateien vorgenommen hatte.

Deshalb hat einer von uns (der an sich chaotischen Gemeinde) die Sache auf sich genommen, und das ganze Projekt unter CVS zu stellen – und siehe da, wir konnten einen beliebigen Zustand in der Vergangenheit wieder herstellen und von dem aus die Änderungen in den einzelnen Dateien exakt nachvollziehen. Aber das hat uns verdammt geholfen, es ist aber nur ein Nebenprodukt von CVS an sich.
Das Vorgehen ist trivial: Irgendwann beschließt ein Manager, daß etwas unter CVS zu verwalten ist. Damit passiert folgendendes:

1.) Ein Mitarbeiter legt das CVS-Repository an. In dem werden alle Projekte verwaltet. Es kann durchaus auch mehrere Repositories geben, aber das ist nicht besonders sinnvoll, außer man muß wirklich eine strenge Grenze ziehen (z.B. wegen Erfüllung von militärischen Auflagen). Dieses Repository ist zu diesem Zeitpunkt noch leer.

2.) Er holt sich die Dateien der Entwickler (oder sie sind sowieso auf dem Fileserver verfügbar) und macht einen "check in" (na ja, so genau läuft es nicht, aber prinzipiell stimmt es) dieser Dateien in das CVS. Wenn die Entwickler ihre Dateien nicht auf dem Fileserver hinterlegt haben, dann bittet er sie, ihre Versionen zu löschen – ja, zu löschen, da danach nicht notwendigerweise die gleichen Verzeichnisstrukturen vorhanden sein werden wie zuvor.
Das Wichtige hier ist, daß der CVS-Manager hier die Möglichkeit hat, tatsächlich alle Dateien für das Projekt zu erreichen, damit sie einmal in das Projekt aufgenommen werden. Dateien, die später einmal zusätzlich aufgenommen werden müssen, kann der CVS-Manager jederzeit einfügen. Aber keine Datei, die nicht im CVS steht, wird zum endgültigen Produkt beitragen (OK, auch hier kann man betrügen, aber wozu?).

3.) Ein normaler Entwickler fängt eine Änderungssession damit an, daß er seinen Teil aus dem CVS "auscheckt". Damit passiert zweierlei: Es wird markiert, daß jetzt jemand an diesem Code arbeitet und in welchem Stadium er ihn übernommen hat.

4.) Irgendwann will ein Entwickler seinen Code – nach dem Testen, logischerweise, denn ansonsten würde das die Sache bloß aufblähen – wieder in das CVS einbringen. Dabei muß er einen Kommentar ausfüllen, der möglichst genau die Änderungen von ihm beschreibt.
Klar, kann jeder Entwickler für sich auch wieder ein CVS (obwohl da auch RCS ausreicht) einsetzen. Der Kernpunkt ist einfach nur der, daß man mit CVS sehr einfach eine Versionskontrolle hat und das auch noch, wenn von einer Version diverse andere Versionen abgeleitet werden (wurden) und aus diesen alten Versionen jetzt neue Versionen erstellt werden sollen.

Hey, dieses Mail wurde eh schon länger als geplant.

Ulrich Paul

Wolf Wejgaard wird es freuen, die GNU-Gemeinde wird es freuen, die Leser der VD wird es freuen. Und den Editor der VD freut das natürlich auch. Weiter so!

fep

Leserbriefe

Betreff: Randbemerkung zu Interpreter vs. Compiler
Von: Ulrich Paul <upaul@paul.de>

Hi Fritz,

diese Diskussion ist doch schon seit Ewigkeiten überflüssig!
...
Also: Kein Anwender weiß heute mehr, ob sein Code interpretiert oder kompiliert wird – warum soll er sich auch darum kümmern? Einzig diejenigen, die spezielle Vorgaben zu erfüllen haben, kümmern sich noch um solche Dinge. Und die Grenzen sind ja noch verschwommener, als es in dem Newsfeed dargestellt wird. Nehmen wir Perl oder Basic, beides "klassische" Interpretersprachen; und in beiden ist die Grenze in der Zwischenzeit so verwischt, daß man nicht mehr von "interpretiert" oder von "kompiliert" sprechen kann.

Bloß als Beispiel: Ich schreibe ein Perl-Skript und fummle so lange an ihm herum, bis es mir paßt. Dann füge ich einen einzigen Befehl ein, nämlich "dump" und ab da benutze ich nur noch das Dumpfile – gut, ich muß vorher "undump" oder "unexe" über es laufen lassen, aber das nur ein einziges Mal, um es in ein echtes Executable umzuwandeln. Das Produkt ist ein kompiliertes Programm, nicht ein Skript mit fest daranhängendem Interpreter.

Und ist Forth mit seinem "NEXT" nun kompiliert oder interpretiert – ich meine Versionen mit indirekt gefädeltem Code? Bei direkt gefädeltem Code taucht diese Frage nicht auf; alle Worte sind kompiliert und zwar im strengsten Sinn des Wortes: Alle Instruktionen sind CPU-Instruktionen, die keine Mehrdeutigkeit zulassen, was die Unterscheidung von Werten und Adressen betrifft.

Aber bei indirekt gefädeltem Code ist ein "Bruch" drin, nämlich beim "JMP AX" (so lautet es bei Intel Prozessoren). Das Register wird mit einem Wert geladen und normalerweise wird jetzt irgendeine Operation, sei sie arithmetisch oder wie auch immer, ausgeführt. Aber immer ist der Inhalt eine "Zahl" mit der man alles anstellen kann, was man mit Zahlen tun kann/soll/darf. Aber das "JMP AX" interpretiert nun diese Zahl als ein Adresse! Und auf Adressen gibt es keine Mathematik!

Vorsicht: Man kann bestimmte Operationen auf Adressen anwenden, aber die beschränken sich auf die Addition und die Subtraktion von OFFSETs – und das ist schon alles. Und diese OFFSETs müssen obendrein noch ganz speziellen Einschränkungen unterliegen. Was ja auch für jeden Menschen einsichtig ist, denn wie kann ich die Adresse des Objektes B aus der Adresse des Objektes A berechnen (in allgemeingültiger Form! Keine Spitzfindigkeiten wie die, daß man in FORTRAN sehr wohl irgendeine Arrayadresse von B berechnen kann, nur wenn man A kennt – das liegt nur an der Art wie alte – und auch noch manche neuere – Fortran-Compiler die Arrays im Speicher ablegen.)

Wenn also schon eine einzige Maschineninstruktion einen "Interpreter" (Zahl in AX ist nicht eine "Zahl", sondern eine Adresse) darstellt, dann ist für mich die ellenlange Diskussion des Newsfeeds einfach langweilig.
Wollen wir in der Forthgemeinde uns diesen Schuh anziehen? Ich definitiv nicht! ...

Ulrich Paul

Betreff: Beitrag v. Andreas Klimas
Von: Ulrich Paul <upaul@paul.de>

Hi Friederich,

sorry, aber immer wenn ich die VD bekomme, dann erwacht etwas in mir, das mich dazu zwingt, Leserbriefe zu schreiben - nimm es als ein positives Zeichen (wie Martin Bitter) oder als ein negatives (was nicht von mir beabsichtigt ist).

Andreas Klimas hat sich da in meinen Augen vom Saulus zum Paulus gewandelt, was – biblisch gesehen ja positiv, in der Programmiererwelt aber – kritisch ist. Sein Vorwurf, daß man in Java Klassen definiert, neue davon ableitet, davon wieder neue ableitet, bis man die Übersicht verloren hat, trifft auf alle Sprachen zu, bei denen so etwas oder ähnliches möglich ist. Mein Java-Package sagt mir zwar nicht genau, wieviele Methoden da jetzt implementiert sind, aber mein F-PC sagt mir, daß es 4000+ Worte hat: Wo ist denn da Forth noch einfacher als Java???

Ich kann in beiden "das Rad von neuem erfinden", mich also auf sehr grundlegende Funktionen beschränken und Arbeiten von anderen damit ignorieren – aber das kann ich in jeder Sprache. Und wenn es schon darum geht, aus der Vielfalt von Funktionen auszuwählen, dann ist mir Java lieber, einfach weil der Compiler schon einen guten Teil von meinen Fehlern feststellt. Warum kann er das?

Weil er einfach – und wirklich nur einfach – die Prototypen miteinander vergleicht (gut, es sind die Signaturen, aber das ist bei Forth egal, weil zwischen Groß- und Kleinschreibung nicht unterschieden wird). Aber dazu muß er sie kennen! Also müßten sich die Forth-Progammierer die Stackkommentare strikt angewöhnen, allerdings erweitert mit einer Bezeichnung des Typs auf dem Stack. Syntaktisch ist das easy: Man schreibt einfach den Typ in eckigen Klammern vor den Parameter. Diese Klammern sind ja schon der Umschalter zwischen interpretieren und kompilieren und damit ist das trivial zu lösen. Also

```
: myadd ( [ int ] a -- [ int ] b [ int ] c ) ... ;
```

würde die neue Version von

```
: mayadd ( a -- b c)
```

sein. Ach, das schaut viel zu ähnlich zu den "anderen" aus und obendrein ist die "klassische" Methode viel einfacher zu lesen? Von mir aus, aber solange Forth mich bei so trivialen Sachen hängen läßt, werde ich es nicht für komplexere Aufgaben einsetzen.

Ja, für hardwarenahe Aufgaben ist es DAS Tool meiner Wahl, aber wenn es komplexer wird, dann greife ich lieber auf die Arbeiten anderer zurück – und die haben mit Forth nichts zu tun. Thats life!!!!!!!

Ulrich Paul

Betreff: swap-drachen
Von: Ulrich Paul <upaul@paul.de>

Hi Friederich,

mir hat der Swap-Drachen sein Leid geklagt. Als er vor so ca. 17 Jahren auf die Welt kam, dachte er sich noch nichts dabei, "Swap"-Drachen genannt zu werden. Inzwischen ist er etwas indigniert, daß man ihn immer noch mit seinem Babynamen anspricht. Er meint, daß wir ja auch nicht jeden <ausländischen Mitarbeiter> als "Hilfsarbeiter" ansehen, und es damit auch Zeit wäre seine Namen entsprechend anzupassen. Wir haben uns – nach einiger Kontroverse, in dessen Rahmen ich ihm auch vorgeschlagen habe einfach die Seiten zu wechseln – uns auf einen neuen Namen geeinigt: "Create Does". Wir haben bewußt alle Sonderzeichen aus dem Namen weggelassen, weil möglichst viele ihn verstehen sollen.

Der Drachen hat mir auch Tips gegeben, was ihm gefallen würde: "Create" auf der Brust und "Does" auf dem Schwanz. Meine Bedenken, daß das u.U. manche weibliche Mitglieder als eine Anspielung verstehen könnten, hat er mit dem Argument abgetan: "Schlimm, wer Schlimmes dabei denkt" (hat er von August dem Starken geklaut)

Ulrich Paul

Donnerwetter – der SWAP muß dem Thomas Beierlein ausgebüchst sein. Tatsächlich ist seine diesjährige Aufgabe, unseren ehemaligen Direktor in Mittweida zu inspirieren. Daß der Bursche (Swap, nicht Thomas – der aber auch; gelegentlich) ein ausgeprägtes Eigenleben entwickelt, habe ich das ganze Jahr 1995 über erfahren müssen. Davon kündet das Drachenbuch (***http://www.fprinz.dyndns.org/Forth/Drachenbuch***).
Einen anderen Namen will der kleine Kerl haben? Nun, das wird er mit dem Drachenrat diskutieren müssen. Wir sind aber allesamt bestechlich ;-) Die Namen CREATE und DOES für die Köpfe leuchten mir durchaus ein. DOES und DOES-NIX sind aber auch nicht schlecht ;-)

fep

– L E B E N S Z E I C H E N –

Lieber Friederich,

Deine jüngste Anfrage nach Beiträgen für die Vierte Dimension läßt mich denken, daß Du keinerlei Lebenszeichen vom Forth in der nächsten Ausgabe drucken wirst, wenn ich Dir erst nach dem SVFIG Treffen vom 25. Oktober schreiben werde. Darum schreibe ich Dir jetzt; wie immer ein wenig in Eile.

Wir haben Dr. Ting bei unserem Septembertreffen vermißt. Und mehr als ein Dutzend von uns haben sich für einen Tag getroffen, ohne daß wir einen ordentlichen Plan gehabt hätten. Das war vermutlich ein wenig enttäuschend für unseren Besucher – Bernd Paysan aus München.
Aber zu unserer Freude hat Bernd einen großen Teil des Tages mit authentischen Informationen zu GFORTH Updates und mit Neuigkeiten von der anderen Seite des großen Teiches ausgefüllt. Vielen Dank an Bernd!
Ich glaube, Bernd wird auch das Vintage Computer Festival am 11. oder 12. Oktober besuchen, wo die SVFIG eine Bude zwischen ungefähr 30 anderen Ausstellern stehen haben wird.
Das Festival wurde von dem Museum für Computergeschichte aus der Taufe gehoben und findet in einem neuen und eindrucksvollen Gebäude in Mountain View (Kalifornien) statt. Das Gebäude gehört Silicon Graphics. Ich habe über dieses Gebäude bereits im letzten Jahr ein paar Zeilen geschrieben. Damals hatte ich seinen Besuch mit einem Besuch im NASA-Gebäude in Sunnyvale verbunden.

Die aktuelle Ausstellung zeigt noch einige Exponate aus dem Vorjahr, einschließlich des Beitrages von Hans Franke (***www.gfhr.de***). Ich muß dazu gleich sagen, daß "vintage" nicht "antik" bedeutet. Das Museum zeigt aber Ausstellungsstücke, die älter sind als MITs Altair.
Um deutlich zu machen, wie sich das Museum selbst der Öffentlichkeit präsentiert, sollte ich wohl einfach auf die Webseite verweisen, die unter ***http://www.computerhistory.org/VirtualVisibleStorage/*** erreichbar ist.
Weiterhin habe ich eine interessante Nachricht aus der National Society of Professional Engineers bekommen. Diese Nachricht hänge ich meinem Brief an. Vielleicht, mit einiger Komprimierarbeit, enthält dieses Material auch Interessantes für die Leser der "Vierte Dimension".

=====

Techniker sagen wagemutige Innovationen für das Tal voraus.
SiliconValley.com (10/13/03); Langberg, Mike

Zukünftige technologische Entwicklungen, welche die Gesellschaft umformen und das Silicon Valley aus seiner wirtschaftlichen Flaute herausheben könnten, waren das zentrale Thema der 4. Silicon Valley Konferenz am 7. Oktober im Museum für Computergeschichte in Mountain View, Kalifornien.
Die angekündigten Innovationen kreisten um das allgegenwärtige Thema "wireless". Curt Carlson, der Präsident der Denkfabrik von SRI International geht davon aus, daß miteinander konkurrierende, erfolgreiche Entwicklungen in der Spracherkennung, in der sich vereinfachenden Arbeit mit Netzwerken und zu Batterien mit größerer Lebensdauer letztlich zu mobilen Geräten führen werden, die den Menschen so selbstverständlich sein werden, wie es heute ihre Brieftaschen sind.
Gary Purdy, Analytiker der Wireless-Szene, sagt dagegen voraus, daß zukünftige Geräte mit allen verfügbaren, drahtlos arbeitenden Netzwerken kompatibel sein werden. Jegliche geeignete Information wird damit automatisch empfangen werden können.

Die Spracherkennung wird sich als entscheidend herausstellen, weil die Menschen mit miniaturisierten Tastaturen auf den Handhelds nicht wirklich umgehen können.
Judy Estrin, Unternehmerin und Vorsitzende von Packet Design, nahm den Beginn eines "phänomenalen Zyklus" mit der Entwicklung von "embedded" Computern innerhalb von schätzungsweise fünf Jahren vorweg. Sie geht davon aus, daß Netzwerk-Hardware entwickelt werden wird, die den Datenverkehr und die gesamte zugehörige Ausrüstung überwacht, wenn embedded Computer und Sensoren mit dem Netz verknüpft werden. Estrin erklärte, daß Silicon Valley dieser Herausforderung nur gewachsen sein wird, wenn es gelingt, kleine Computer mit geringem Stromverbrauch zu entwickeln.
Jeff Hawkins von Handspring berichtete, daß die neurologische Forschung nahe daran ist zu verstehen, wie das Gehirn menschliche Sprache erzeugt und umsetzt. Hawkins fügte hinzu, daß eine Abbildung dieser Prozesse in Silkonchips eine "vollkommen unerwartete" Revolution in der Computertechnik bewirken wird.
Biologisches Engineering war auch das Diskussionsthema von Geoffrey Moore von Mohr Davidow Ventures, einer Gesellschaft, deren geschäftlicher Fokus auf der Entwicklung von Chips zur Überwachung medizinischer Prozesse liegt.

Im Original können Sie einen Bericht der Tagung lesen bei: (***http://www.siliconvalley.com/mld/siliconvalley/7001991.htm***)
=====
Das sind nun meine Ausführungen für heute. Ich hoffe, daß ich nach dem nächsten SVFIG-Treffen noch etwas mehr an Informationen haben werde. Glückauf

Henry

Gehaltvolles

zusammengestellt und übertragen
von Fred Behringer

FORTHWRITE der FIG UK, Großbritannien

Nr. 122, September 2003

1 Editorial
Graeme Dunbar <g.r.a.dunbar@rgu.ac.uk>

Graeme stellt sich als Interimsredakteur vor, und zwar für die Zeit, bis zu der Chris Jakeman, der der allgemeinen Rationalisierung zum Opfer gefallen ist, eine neue Beschäftigung gefunden hat. Graeme bedauert, dass das Heft diesmal nur 24 Seiten stark ist. Nach Meinung des Rezensenten trotzdem bewundernswert: Die Aufmachung ist dieselbe geblieben und es ist kein Bruch zu erkennen. Graeme begrüßt sechs neue Mitglieder, darunter Wolf Wejgaard (Holon-Forth) aus der Schweiz.

2 Forth News
Chris Jakeman

C-nach-Forth-Compiler; FLint (Lint für Forth); Forth für EPOC; Gforth 0.6.2 (mit Super-Instructions); VFX 3.5 für Windows ("schnellstes Forth des Planeten"); Quelltext zu eForth; #Forth-Portal; Win32Forth hat jetzt ein Bulletin-Board; xyplot 1.1.3 für Windows; Leaf-Editor (Programmiertool für Forth unter Windows).

4 What's All This Compiler Stuff, Anyhow?
Jeremy Fowell <jeremy.fowell@btinternet.com>

Wie kommt Forth-Code (PygmyHC11) in den Einplatinencomputer F11-UK? Schreiben auf dem PC in eine Datei und dann zusammen mit dem PygmyHC11-System in den Speicher des HC11 downloaden. Jeremy gibt genaue Anweisungen.

7 Connecting an LCD to the F11-UK Single Board Computer
Jeremy Fowell

Dem F11-UK wird ein LCDisplay (2 Zeilen zu 16 Zeichen zugeschaltet. Fast immer, sagt Jeremy, ist das ein HD44780U-Modul (Hitachi). Ein bisschen Forth und ein sehr schönes Schaltbild (eigentlich nur noch ein 74HC595 als Interface). Das Schaltbild ist so schön simpel, dass sich ab jetzt auch der Rezensent (elektronisch auf Bastlerniveau) zutraut, ausgeschlachtete LCDisplays irgendwo dranzuschalten. Mehr Informationen: ftp://ftp.ee.ualberta.ca/pub/cookbook/faq/lcd.doc .

10 F11-UK
Jeremy Fowell

Und hier wieder der Einplatinencomputer F11-UK. Multitasking und Assembler eingebaut, 30 Seiten Handbuch ... Mehr sagt Ihnen der Autor und Projektleiter.

11 Membership Matters
Doug Neale & Graeme Dunbar

Verwaltungschef und Redakteur machen sich Gedanken über die Mitgliederentwicklung. "Die Mitglieder kommen nicht nur aus dem Vereinigten Königreich, sondern von praktisch überall her. Einige davon beschäftigen sich mit Robotern." Duncan Louttit: "Ich arbeite mit Swallow-Systems, ein Hersteller von Lehrrobotern. Die gesamte Firmware ist in Forth geschrieben." Wo verstecken sich die Roboter-Addicts unter den Forth-Freunden?

12 FIG UK - AGM
Die "Jährliche Generalversammlung" am 25. Oktober bei Doug Neale zu Hause (im Großraum London, mit der U-Bahn ereich-

bar). Alle Mitglieder sind herzlich eingeladen. Besprochen wird auch (wurde nun wohl inzwischen) das Jubiläumstreffen 2004 zum 25-jährigen Bestehen der FIG UK.

13 Library Notes
Graeme Dunbar

Neuigkeiten und Bemerkungen vom FIGUK-Bibliothekar (und Redakteur in Personalunion). Leihfrist gewöhnlich bis zu drei Monaten. Für FIGUK-Mitglieder kostenlos. Erwartet wird die Rückerstattung des Portos. Ältere Forthwrite-Hefte ausleihen ist verwaltungstechnisch schwieriger als gezielt die interessierenden Artikel zu kopieren.

14 euroFORTH 2003

Die Konferenz findet (fand) diesmal in England statt, in Rosson-Wye, vom 17. bis 19. Oktober 2003. Unterbringungskosten (mit Vollpension) 300 Pfund.

15 Across the Big Teich
Henry Vinerts <Volvovid@aol.com>

Henrys Reports vom Juli und August 2003 über SVFIG-Aktivitäten in Originalfassung. Wir kennen sie in der Übersetzung von Thomas Beierlein. Diese Berichte werden in der Forthwrite stets mit den folgenden einleitenden Zeilen veröffentlicht: This material was prepared for Vierte Dimension by Henry Vinerts, and printed by kind permission of Forth Gesellschaft (German FIG).

18 What Languages Fix
Paul Graham

Ein Abdruck einer Tabelle von Paul Grahams Homepage, in der versucht wird, in nicht mehr als 10 Wörtern festzuhalten, was die einzelnen Sprachen auszeichnet oder durch was sie charakterisiert werden. Es werden 26 Sprachen aufgezählt. Der Forthwrite-Redakteur bemerkt, dass Forth sich nicht darunter befindet, und bittet die Leser um einen entsprechenden Ergänzungsversuch.
Beispiele:
C: Assembler ist nicht "hoch" genug.
C++: C ist nicht "hoch" genug.
Java: C++ ist ein "kludge" (?) und Microsoft erdrückt uns.
C#: Java wird von Sun beherrscht. ...

19 Vierte Dimension 1/2003
Joe Anderson

Joes Besprechung unserer VD. Auf Seite 20 steht ein "Dank der FIG-UK-Library an Rolf Schöne und die Forth-Gesellschaft für die Gratis-VD-Hefte 2/2003 und 3/2003".

21 Letters

Auch diesmal wieder keine Leserbriefe. Aber ein dringender Aufruf, welche zu schreiben.

22 Dutch Forth Users Group

Die Werbeanzeige der holländischen Forth-Freunde. Das nächste Mal wären dann wohl wir wieder dran.

Gehaltvolles

zusammengestellt und übertragen
von Fred Behringer

VIJGEBLAADJE der HCC Forthgebruikersgroep, Niederlande

Nr. 40, Oktober 2003

6809 werkgroep
Paul Wiegmans

Vor einigen Monaten hat Ron Minke der Forthgebruikersgroep ein paar ausgediente Meß- und Steuerungscomputer überlassen. Die Arbeitsgruppe, sie besteht aus 6 Mitgliedern, wurde gegründet, um diesen Geräten mittels Forth neues Leben einzuhauchen. Die Geräte sind mit einem Einplatinencomputer der Firma Brutech Electronics ausgerüstet Mittelpunkt ist der altbekannte 6809. Das verwendete Forth basiert auf Camel-Forth von Brad Rodriguez, ist voll ANS-konform und wird per F83 metacompiliert. Näheres auf http://home.hccnet.nl/p.c.wiegmans/6809werkgroep.

Morse decoder
Albert Nijhof

Das Vijgeblaadje 38 enthielt einen Artikel über einen Morsezeichengeber. Albert bringt hier das Gegenstück, einen Morsezeichendecodierer. Das Programm arbeitet zunächst noch mit nur simulierter Eingabe in Form von Zahlen. Es sollte auf den beschränkten Seitenplatz des Vijgeblaadjes passen. Es erkennt aber schon die Eingabegeschwindigkeit automatisch. Von der Hardwareseite her ist ein Mikrofon für die ATS-Platine vorgesehen. Albert hat sich mit den Hardware-Fachleuten der Forthgebruikersgroep besprochen: Zur Mikrofonansteuerung ist eine Interrupt-Routine nötig, die sinnvollerweise in Maschinensprache ausgeführt werden soll. Albert gibt hier schon die Vorgabe in High-Level-Forth und verlässt sich im Übrigen auf Willem Ouwerkerk und die nächste Ausgabe des Vijgeblaadjes. Näheres unter http://www.forth.hccnet.nl/nieuws .

Forth, USB und ein Webserver auf einer Smartcard

Bernd Paysan

Zusammenfassung

Dr. Walter Hinz von Giesecke und Devrient hat auf einem der Münchner Forth-Treffen eine Smartcard vorgestellt. Die Proof-of-the-Conzept-Implementierung in Forth zeigt, daß man eine Smartcard auch mit Standardprotokollen und Schnittstellen ansprechen kann. Das alles könnte man natürlich auch in Java implementieren, aber bis das fertig ist...

Einleitung

G&D ist eine Firma, die sich unter anderem mit fälschungssicherem Druck (z.B. von Geldscheinen und Ausweisen) beschäftigt, und natürlich auch mit Smartcards, denn schließlich stellen die ja Geld und Ausweise der Zukunft dar. Wie es sich für eine richtig paranoide Großfirma gehört, darf natürlich nicht jeder Mitarbeiter so einfach über seine Arbeit reden, sondern nur ausgewählte Leute. In diesem Fall war der Vortragende also Dr. Walter Hinz.

1. Smartcards am Computer

Jeder kennt sie, die Smartcards mit dem Kontaktfeld. Anwendungen gibt es schon eine Menge. Allerdings: Wer eine Smartcard einsetzen will, braucht einen Kartenleser und eine entsprechende Software, da Smartcards keine Standard-PC-Schnittstelle verwenden, und auch keine Standardsoftware das Protokoll versteht --- und für jede Kartenanwendung braucht man eine eigene Karte. Das soll sich ändern. Die vorgestellte Smartcard hat auf zwei weiteren Kontakten ein USB-Interface implementiert. Auf der Karte ist neben dem USB-Anschluß auch ein 8051 (die Dinger sind nicht totzukriegen), ausreichend ROM und EEPROM (zusammen 192k) und mickrige 4k RAM. Die Karte wird so von Siemens hergestellt, Kunden können sich eine ROM-Maske für ihre Software machen lassen.

Das USB-Interface verlangt nach einem Stecker, nicht nach einem flachen Kontaktfeld, aber eine ziemlich winzige Platine reicht aus, um alles nötige unterzubringen, nebst der obligatorischen LED, die den Kontakt anzeigt (auch einen Quarz für den genauen Takt und wohl einen Spannungsregler für die 3.3V Pegel auf den USB-Signalleitungen). Die Platine steckt in einem durchsichtigen Plastikgehäuse, das etwa die Größe eines USB-Sticks hat. Eine seitlich ausfahrende Klappe erlaubt es, die Karte hineinzustecken, und bietet gleichzeitig eine seitliche Führung.

2. USB und das Internet

Eigentlich ist USB als Schnittstelle nicht optimal. USB verbindet irgendwelche Peripherie wie Drucker oder Scanner mit dem Computer, und dafür braucht man dann einen Treiber. Und ein Programm, das dann mit den Treiberdaten etwas anfangen kann, wenn man nicht in das Schema F paßt. Das Treiberproblem wäre keins, wenn's sowas wie ein USB-Modem oder ein USB-Netzwerk gäbe. Das, was unter dem Namen "USB-Modem" fährt, ist aber eine Soundkarte --- der Computer muß sich selbst um das Modulieren und Demodulieren kümmern. Serielle Schnittstellenwandler für USB gibt's auch, aber bei näherem Hinsehen hat sich herausgestellt, daß deren Protokoll so komplex ist, daß man es lieber nicht emulieren will. Letztendlich hat G&D also einen eigenen USB-Treiber geschrieben, der eine Art serielle Schnittstelle zur Verfügung stellt. Über die wird dann mit SLIP ein Netzwerk gebaut. Der SLIP-Treiber wurde als C-Library eingekauft. Auf Betriebssystemseite genügen damit die eingebauten Netzwerktreiber für Dial-Up-Verbindungen.

3. Der Internet Explorer als Smartcard-Reader

Das Protokoll, das man jetzt verwendet, ist klar: HTTP. Einen Browser findet man wohl auf jedem Rechner, und Microsoft meint sogar, er gehöre direkt ins Betriebssystem. G&D hat sich dabei von meinem Web-Server in Forth inspirieren lassen, aber eine eigenständige Implementierung gemacht. Nicht nur wegen der GPL, sondern auch weil mein Server zu viele Anforderungen an das System stellt, wie zum Beispiel dynamische Strings. Die Smartcard hat ja nur 4k RAM, da geht's eng zu.

Immerhin hat die Smart-Card ein Dateisystem. Die Dateien liegen im EEPROM. Es gibt hier "normale" Dateien, die der HTTP-Server einfach verschickt, und CGI-bins. Damit hat man ein offenes System: Die Applikation der Smart-Card wird durch die CGI-bins und die statischen Seiten definiert.

Als Beispiel dient ein Ticket-System: Man kann Tickets buchen, bezahlen und später entwerten. Die ausgeführten Transaktionen werden gespeichert, und natürlich kann man sich dieses Log auch ansehen. Wenn man das mit einem typischen LAMP[1])-Webserver vorführen würde, wäre das nicht sonderlich spektakulär, aber statt LAMP auf einem Pentium läuft auf der Karte ja nur Forth und ein SLIP-IP-Stack auf einem 8051. Natürlich merkt man, daß der 8051 ziemlich lahm ist, aber mit Internet über ein Modem kann er schon mithalten.

Bernd Paysan

1) Linux+Apache+MySQL+PHP

Vier Gewinnt
Alpha-Beta Min-Max in Forth

Bernd Paysan

Zusammenfassung

Die Alpha-Beta-Min-Max-Strategie ist bei Computerspielen wie Schach der Weg zum Ziel. Schach ist aber ein komplexes Spiel, und eignet sich deshalb nicht so sehr, diese Strategie (die übrigens schon von Alan Turing entwickelt wurde) zu beschreiben. Ich verwende hier *Vier Gewinnt* als Beispiel.

Einleitung

Vier Gewinnt ist ein einfaches Strategie-Spiel. Das Spielfeld ist senkrecht, die Steine fallen also in den sieben Spalten à sechs Zeilen immer nach unten. Gewonnen hat, wer vier Spielsteine in eine Reihe setzt. Bild 1 zeigt eine solche Gewinn-Situation. Soviel ich weiß, ist Vier Gewinnt ein Nullsummenspiel, es kommt also nicht darauf an, wer den ersten Zug macht; bei idealer Strategie kann man jederzeit ein Remis erreichen. Auf den ersten Blick ist Vier Gewinnt ein übersichtliches Spiel, da man ja pro Zug nur maximal sieben Möglichkeiten hat. Trotzdem gibt es insgesamt

$$42! / 6!^7 \sim 1{,}4 * 10^{31}$$

oder 14 Quintillionen Spiele, wobei hier natürlich alle die ausscheiden, die schon vorher gewonnen werden.
Ein vollständiges Durchrechnen kommt also nicht in Frage. Die übliche Vorgehensweise ist eine Suche durch den Spielbaum bis zu einer gewissen Tiefe, und eine Bewertung der Endstellungen.

1. Das Spielfeld

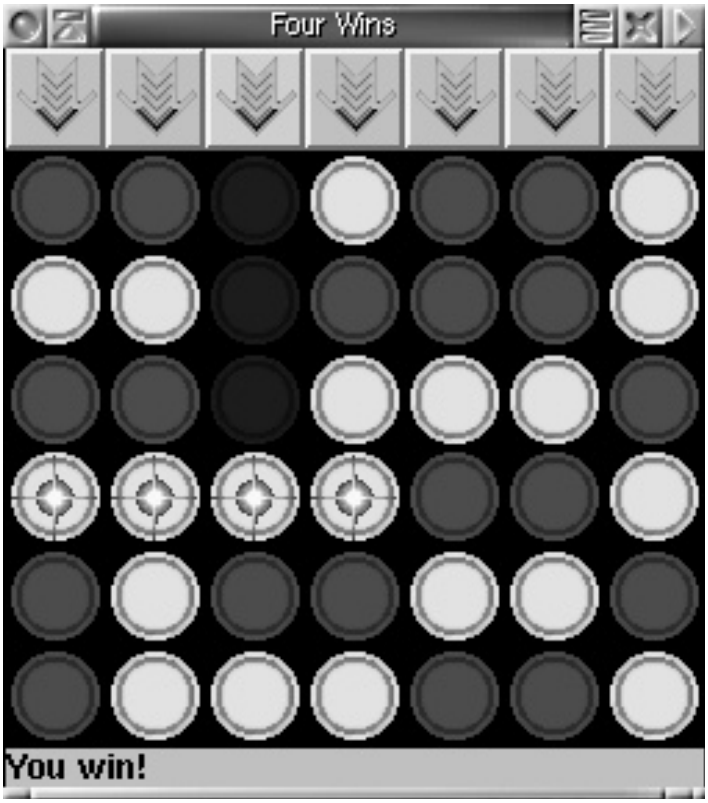
Zunächst brauchen wir ein paar Konstanten, die das Spielfeld und die Suchtiefe definieren:

```
6 Value #rows
7 Value #cols
4 Value #win
#rows 2 + Value *rows
#cols 2 + Value *cols
8 Value #depth
```

Das Board wird auf allen Seiten durch leere Spielsteine abgegrenzt. Das ist auch eine übliche Vorgehensweise, die es erlaubt, alle Berechnungen durchzuführen, ohne über Grenzen des Spielfelds nachdenken zu müssen. Die Steine in Fallrichtung sind aufeinanderfolgende Speicherstellen, damit kann man den Befehl skip verwenden, um den ersten belegten Platz zu finden.

```
Create board here *rows *cols * dup allot erase
  [IFUNDEF] cx@ : cx@ ( addr -- c )
                  c@ dup $80 and negate or ; [THEN]
: b[] ( x y -- board[x,y] )
  *rows * + [ board *rows 1+ + ] ALiteral + ;
```

Mit b[] kann man die Adresse einer Spielfeldposition berechnen. cx@ gibt's in bigFORTH schon, für andere Forthe wird es hier definiert: Es liest Zeichen mit Vorzeichen aus.
Abbildung 1: Vier Gewinnt



2. Bewertungsfunktion

Da der Computer umso besser spielt, je schneller er bewertet, wird der Wert jedes gesetzten Steins beim Setzen berechnet. Der Wert berechnet sich aus der Anzahl der Nachbarn gleicher Farbe in einer Reihe. Das Vorzeichen legt die Farbe fest. Wir gehen dabei von der Position des Steins aus, müssen also in acht Richtungen gehen, und dabei das Maximum der Werte berechnen. Am Schluß muß noch das Vorzeichen korrigiert werden.

```
: score? ( boardp -- score-addr )
  >r
  r@ >left r@ >right +
  r@ >up r@ >down + max
  r@ >lu r@ >rd + max
  r@ >ld r@ >ru + max 1+ cur-stone @ *
  r@ c! r> ;
```

Die Such-Wörter definieren wir mit Create..DOES>:

```
Variable cur-stone
: seeker DOES> @ ( addr index -- n )
  over #win 0 ?DO over + dup cx@ cur-stone @ * 0<=
  ?LEAVE LOOP
  swap >r - negate r> / 1- ;

: seek ( n -- ) Create dup , seeker Create negate , seeker ;

  1 seek >left >right
  *rows seek >up >down
  *rows 1- seek >lu >rd
  *rows 1+ seek >ld >ru
```

So, jetzt fehlt nicht mehr viel, um Steine zu setzen.

```
: stone ( side col -- score-addr ) over cur-stone !
  0 swap b[] #rows 0 skip drop 1- tuck c! score? ;

Variable gameover gameover on
: stone? ( n col -- ) stone cx@ abs #win >= gameover ! ;
```

Ein Stein auf eine Spalte gesetzt (1 für den Computer, - 1 für den Menschen) gibt die Adresse im Array zurück. Wollen wir nur wissen, ob der Zug das Spiel beendet, müssen wir den Wert des Steins mit der Gewinnzahl vergleichen: 4 gewinnt.
Zur Berechnung des besten Zugs brauchen wir noch ein paar Hilfsfunktionen zum Wechseln der Seite und zum Zurücknehmen eines Zugs. Und eine globale Variable für die Seite brauchen wir auch.

```
Variable side -1 side !
: <stone ( score-addr ) 0 swap c! ; [IFDEF] macro macro
                                    [THEN]
: /side side @ negate side ! ; [IFDEF] macro macro
                                [THEN]
```

<stone legt einen Stein zurück --- setzt also den Score zurück auf 0. Und /side wechselt die Seite. Am Ende der Baumsuche müssen wir das Spiel bewerten. Hier brauchen wir eine brauchbare Heuristik. Ich bilde die Quadratsummen über die Werte der einzelnen Steine, und multipliziere das Ergebnis für die bewertete Seite mit der Differenz zwischen beiden Seiten. Eine gute eigene Stellung ist nur viel wert, wenn die des Gegners entsprechend schlechter aussieht.

```
: leaf-score ( -- score )
  0 0 board *rows *cols * bounds ?DO
    I cx@ dup 0>= IF dup * + ELSE swap >r dup *
                + r> THEN
  LOOP side @ 0< IF swap THEN over swap - * 8* 7
  random + ;
```

Am Schluß wird noch ein Zufallswert dazugegeben, daß gleichwertige Spielstellungen nicht gleichwertig aussehen. Das sorgt für mehr Abwechslung beim Spielen.

3. Die Baumsuche

Ein paar Konstanten brauchen wir noch, um eine eindeutige Bewertung für Sieg und Niederlage zu haben:

```
  $7FFFFFFF Constant <best>
  <best> negate Constant <worst>
  <best> 1- Constant <win>
  <win> negate Constant <lost>
  <best> 2/ 1+ Constant <half-best>
```

Das Prinzip der Suche ist eigentlich ganz einfach: Man probiert einfach alle Stellungen durch und nimmt dann die, bei der die Situation für den Gegner besonders schlecht aussieht. Das heißt: Der bestmögliche Zug des Gegners bringt ihm weniger Punkte als derjenige bei anderen Zügen. Damit haben wir die Rekursion schon definiert: Wir probieren rekursiv für jeden Zug aus, was der Gegner daraus machen könnte. Einige Teile des Suchbaums brauchen wir dabei nicht näher betrachten: Wenn z.B. der Gegner schon bei seinem ersten Zug eine bessere Bewertung kriegt als bei einem anderen Zug von uns, brauchen wir den Rest nicht durchsuchen: Es kann nur noch schlimmer kommen. Dasselbe gilt natürlich auch umgekehrt, denn der Gegner sucht ja seinerseits rekursiv nach dem für ihn besten Zug.
Im Detail, und Schritt für Schritt: Zunächst gucken wir uns die Abbruchbedingung der Rekursion an. Sind wir am Ende, bewerten wir nur die aktuelle Stellung.

```
: eval-min-max ( beta n -- score best )
   dup 0= IF 2drop leaf-score 0 EXIT THEN
```

Dann wechseln wir die Seiten, und initialisieren den besten Zug und dessen Score.

```
   /side -1 <worst> ( beta n best alpha )
```

Jetzt gehen wir Spalte für Spalte durch, und setzen einen Stein. Wenn der Stein gewinnt, sind wir fertig: Einen besseren Zug gibt's nicht.

```
#cols 0 ?DO
   0 I b[] cx@ 0= IF
      side @ I stone >r
      r@ cx@ abs #win >= IF
         r> <stone 2drop I <win> LEAVE THEN
```

Nun kommt der rekursive Aufruf: Unser Alpha ist das Beta für den Gegner --- nur negiert. Und die Tiefe verringert sich um eins. Der Zug des Gegners selbst interessiert uns gar nicht, nur sein Score.

```
         2over 1- swap >r over negate swap recurse drop
```

Für den Rückgabewert (den Score, den der Gegner erzielt) noch ein Trick: Wir ziehen eins ab, wenn's einen Gewinn gab. Das gibt einen Bonus für schnelle Gewinne, wir wollen den menschlichen Gegner ja nicht unnötig warten lassen. Außer-

Holländisch ist gar nicht so schwer. Es ähnelt sehr den norddeutschen Sprachgepflogenheiten. Und außerdem ist Forth sowieso international. Neugierig? Werden Sie Förderer der

HCC-Forth-gebruikersgroep.

Für 10 € pro Jahr schicken wir Ihnen 5 oder 6 Hefte unserer Vereinszeitschrift 'Het Vijgeblaadje' zu. Dort können Sie sich über die Aktivitäten unserer Mitglieder, über neue Hard- und Softwareprojekte, über Produkte zu günstigen Bezugspreisen, über Literatur aus unserer Forth-Bibliothek und vieles mehr aus erster Hand unterrichten. Auskünfte erteilt:

Willem Ouwerkerk
Boulevard Heuvelink 126
NL-6828 KW Arnhem
E-Mail: ***w.ouwerkerk@kader.hobby.nl***

Oder überweisen Sie einfach 10 € auf das Konto 525 35 72 der HCC-Forth-gebruikersgroep bei der Postbank Amsterdam. Noch einfacher ist es wahrscheinlich, sich deshalb direkt an unseren Vorsitzenden Willem Ouwerkerk zu wenden.

dem nehmen wir den Stein natürlich wieder runter; war ja nur ein Versuch.

```
dup <half-best> / - negate r> r> <stone
```

Jetzt gucken wir uns nur noch an, ob der Zug schlechter als das bisher schlechteste Beta ist (dann brauchen wir nicht weitermachen), oder besser als das beste Alpha (dann ist es unser bester Zug).

```
\ beta n best alpha score beta
\ if score better than beta, we are done
  2dup > IF drop nip nip I swap LEAVE
          THEN drop
\ if score better than alpha, new score is
\ best one
  2dup < IF nip nip I swap ELSE drop
          THEN
  THEN
  LOOP swap 2swap 2drop /side ;
```

Am Schluß müssen wir nur noch Beta und N vom Stack nehmen und die Seite wechseln. Das reicht aus, um den Computerzug zu definieren. Wir müssen nur die Seite festlegen, und dann suchen lassen.

```
: c ( -- score ) -1 side !
   <best> #depth eval-min-max 1 swap stone? ;
```

Und der menschliche Zug ist auch schnell definiert. Ggf. erst mal das Feld initialisieren, und dann noch ein paar Sicherheitsabfragen.

```
: 4init gameover off board *rows *cols * erase ;
: h ( n -- ) gameover @ IF 4init cr ." New game" THEN
    dup #cols 0 within abort" sorry, outside the field"
    0 over b[] cx@ abort" sorry, column already full"
```

Wenn der Zug gültig war, müssen wir natürlich schauen, ob der User gewonnen hat, und ihm ggf. gratulieren. Ansonsten ziehen wir, und gratulieren uns, wenn wir gewonnen haben.

```
-1 swap stone? gameover @ 0= IF
c <lost> #depth + <= IF ." I'm going to lose"
ELSE gameover @ IF ." I win" THEN THEN
ELSE ." you win" THEN
```

Wenn nichts mehr auf dem Spielfeld Platz hat, dann ist es halt unentschieden ausgegangen. Und das Ergebnis des Zugs zeigen wir natürlich an.

```
true #cols 0 ?DO 0 I b[] cx@ 0<> and LOOP
IF ." tie" gameover on THEN .board ;
```

Dank MINOS kann man dem auch recht einfach eine Benutzeroberfläche draufsetzen; die erkläre ich hier aber nicht.

Bernd Paysan

FIG UK

(Englische Forth-Gesellschaft)

Treten Sie unserer Forth-Gruppe bei.
Verschaffen Sie sich Zugang zu unserer umfangreichen Bibliothek.
Sichern Sie sich alle zwei Monate
ein Heft unserer Vereinszeitschrift.
(Auch ältere Hefte erhältlich)
Suchen Sie unsere Webseite auf:
www.users.zetnet.co.uk/aborigine/Forth.htm
Lassen Sie sich unser Neuzugangs-Gratis-Paket geben.
Der Mitgliedsbeitrag beträgt 12 engl. Pfund.
Hierfür bekommen Sie 6 Hefte unserer
Vereinszeitschrift Forthwrite.
Beschleunigte Zustellung (Air Mail)
ins Ausland kostet 20 Pfund.
Körperschaften zahlen 36 Pfund,
erhalten dafür aber viel Werbung.

Wenden Sie sich an:

Dr. Douglas Neale
58 Woodland Way
Morden Surrey
SM4 4DS
Tel.: (44) 181-542-2747
E-Mail: dneale@w58wmorden.demon.co.uk

Ein Target-Compiler, basierend auf einer Virtual-Machine

Ulrich Paul

Einleitung

Dieser Artikel, bzw. seine Fortsetzungen, beschreibt einen Compiler, der auf einem oder mehreren Rechnern läuft. Die Ausgabe ist ein Image, das direkt in ein ROM gebrannt werden oder unter einem entsprechenden Betriebssystem laufen kann, obwohl der Schwerpunkt auf dem ROM-Image liegt. Der Grundgedanke ist einfach: Es gibt eine so breite Palette von Prozessoren, daß es sich nicht rentiert, für jeden von ihnen ein eigenes Forth-System zu schreiben oder auch nur ein bestehendes zu portieren. Viele dieser Prozessoren werden dort eingesetzt, wo sie, klein und billig, herkömmliche Steuerungen in TTL-Technologie ersetzen. Daher haben manche nur ein Minimum an RAM und ROM - zu wenig, um ein "voll aufgeblasenes" Forth-System zu beherbergen. Andere mit mehr Ressourcen, profitieren davon, wenn sie nicht die volle "Last" eines Forth-Systems tragen müssen und statt dessen mehr für den Schnick-Schnack rund um die eigentliche Applikation frei haben. Das gundlegende Konzept wurde von dem Autor und Declan O'Mahony (Onsite Computer) auf einer "Klausurtagung" in Südtirol 1987 (im Herbst, beim Törggelen) festgelegt. Dieser Artikel erscheint mit vollem Einverständnis von Declan O'Mahony. Es sei aber darauf hingewiesen, daß das Konzept und seine Implementierung immer noch dem Copyright unterliegen, allerdings mit folgender Einschränkung: Es dürfen das Konzept und die hier und auf der Website zur Verfügung gestellten Implementierungen frei verwendet werden, solange die ursprünglichen Autoren genannt werden und jeder, dem etwas ausgeliefert wird, das zur Verfügung gestellte Material erthält, einschließlich dem Zugriff auf die entsprechenden Sourcen. Das ist in einfachen Worten die GNU-Public-License. Das ganze Projekt wurde unter FPC realisiert. Das heißt nicht, daß es nur unter FPC läuft, aber es wurde halt erst auf einem PC implementiert - und FPC läuft auch in einer DOS-Box unter WinXX oder unter dosemu unter Linux. Also gab es keinen Grund es jemals auf ein anderes System zu portieren. Teil des gesamten Konzeptes war es, daß es eben nicht so fest an ein gegebenes Forth-System gebunden ist. Wer die Beschreibung unter diesem Gesichtspunkt betrachtet, wird feststellen, daß eine Menge Sachen so und nicht anders gemacht wurden, um genau dieses Ziel zu erreichen: Es werden von dem Host-System (Erklärung weiter unten) nur minimale Funktionen erwartet, danach "lebt" der Target-Compiler in seiner eigenen Welt. Das ist auch der Grund dafür, daß man nicht unbedingt ein wirkliches Target haben muß, um bestimmte Dinge zu testen. Viele Funktionen können auf dem Host oder einem intermediären Rechner getestet werden, ohne daß ein lauffähiges Zielsystem zur Verfügung steht.

Begriffsdefinitionen

Host-System
Das System, das der compilierende Benutzer benutzt. Je nach Target hat er auch Zugriff auf verschiedene Funktionen des Targets. Aber:

Host-Rechner
Der Rechner an dem der compilierende Benutzer sitzt. Dieser Rechner kann aber durchaus auch vom Enduser benutzt werden. Damit ist gemeint, daß auf diesem Rechner durchaus z.B. eine Terminalemulation laufen kann, mit der das Target interaktiv bedient werden wird. In diesem Artikel spielt der Host-Rechner keine Rolle, und diese Definition wurde nur der Klarheit halber aufgenommen.

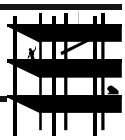
Target-System
Das System, für das der Code generiert wird. Je nach seinen Möglichkeiten läuft mehr oder weniger auf ihm ab. Es hängt von dem "Level" des Targets ab, was es kann und können muß.

Target-Levels

Level 0
Das Target hat keine Ahnung von einem Hostsystem. Es ist ausschließlich ein Programm-Image im ROM, das abläuft, und das keine Ahnung von seiner Entstehung hat. Dieses Image kann keine weitere Komunikation mit dem Hostsystem pflegen (z.B. zwecks Erweiterung seiner Fähigkeiten). Wohl aber kann es jede geforderte Kommunikation mit seiner Außenwelt unterhalten, z.B. Kontakt zu Sensoren, Aktuatoren oder einem User-Display, samt Keyboard.

Level 1
Das Target kann Werte an einer vom Host vorgebenen Position speichern und auch eine Routine aufrufen, die an einer vom Host angegebenen Adresse liegt. Bei Architekturen mit getrenntem Code- und Daten-Speicher ist der Zugriff vom Host auf beide Speicher in vollem Umfang möglich. Dazu muß das Target entsprechende primitive Routinen zur Verfügung stellen. Wichtig: Der Host braucht keinen direkten physikalischen Zugriff auf diese Speicherbereiche zu haben. Im Normalfall ist es sogar so, daß die entsprechende Kommunikation zum Erreichen dieser Ziele via einer physikalischen Schnittstelle geht, die so eine Funktionalität a priori gar nicht kennt, z.B. ein RS232-Port. Die Grenze zum nächsten Level ist etwas fließend, da dieser Level durchaus eine Interpretation von den empfangenen Daten machen muß - also auch so etwas wie Tokens (s.u.) interpretieren. Aber er muß seinen Speicher nicht selbst verwalten.

Level 2
Das Target kann aufgrund eines Tokens (einer binären Zahl) vom Host selbständig entscheiden, welche Funktion ausgeführt wird. Dazu konsultiert es eine Liste, die die Abbildung der Tokennummer auf die Startadresse enthält.



Diese Routinen können durchaus kompilierende Worte sein, nur daß ihr Aufruf eben über Tokens geht, also völlig unabhängig von der gewählten ASCII-Darstellung irgendwelcher Namen.

Level 3
Das Target erkennt einen ASCII-Strom von Anweisungen. Diese Anweisungen enthalten keine Kommentare oder sonstige Meta-Informationen. Allerdings muß es alle Kommandos und Parameter im Klartext verstehen. Andererseits hat es kein Verständnis von der Herkunft dieser Kommandos, bzw. Parameter. So etwas wie eine Datei oder gar ein Verzeichnis kennt es nicht.

Level 4
Das Target ist ein vollwertiges System, das vollen Zugriff auf den nativen Sourcecode hat. In diesem Artikel gilt das nicht als ein Target-System, da der Zugriff auf das Filesystem in diesem Konzept dem Host zugeordnet ist. Die Beschreibung dieses Levels wurde nur der Vollständigkeit halber aufgenommen. Aber das soll niemanden daran hindern, diesen Target-Compiler auch auf so ein Target "anzusetzen", obwohl da vielleicht ein Cross-Compiler besser geeignet ist.

Ist das so etwas wie Java-Byte-Code?

Ja und nein. Ja, weil derselbe Gedanke zugrunde liegt: Eine (bei Java), bzw. mehrere (in diesem Konzept), abstrakte Zwischenschichten zu definieren, um so eine Trennung vom Entwicklungsrechner und dem Zielrechner zu erreichen. Aber hier hören die Gemeinsamkeiten auch schon auf. Erstens ist dieses Konzept viel älter als Java (1987, s.o.), und zweitens baut es auf mehreren Schichten auf, die, je nach Anforderung, nach außen hin sichtbar sind. Java hat eine genau definierte JVM. Aber hier kann es, je nach Targetlevel, durchaus mehrere geben. Wichtig in diesem Zusammenhang ist auch eines: Das hier vorgestellte Konzept hat keinerlei Sicherheitschecks eingebaut - ein Target verläßt sich blind darauf, daß der Host gutartig und wissend ist. Das steht im Gegensatz zu der Sandbox bei Java. Aber die Zielsetzung beider Systeme ist grundlegend unterschiedlich: Java verfolgt das Konzept "Write Once, Run Everywhere" und unser Konzept ist "Write Once, Port It To Everything". Während ein Javasystem (potentiell zumindest) jederzeit Code vom Internet beziehen kann, geht das bei unserem System nicht, da ein entsprechend wissender Host am anderen Ende vorausgesetzt wird.

Eine kurze Einführung in die Kommunikation zwischen Host und Target

Gegeben sei in allen Fällen eine bytetransparente Verbindung zwischen Host und Target. Das kann ein serieller Port sein, aber auch jede andere Schnittstelle, über die man ein Byte (Mindestbreite, aber nicht unbedingt das obere Limit) ohne Modifikationen übertragen kann (z.B 9600Bd, 8bit, no parity). Wie die Übertragung wirklich geschieht, hängt von der Konstellation, meistens von den Möglichkeiten des Targets ab. Ich habe auch schon Targets über SPI an den Host angeschlossen. So, und nun schauen wir uns die Kommunikation zwischen dem Host und den Targets verschiedener Level an. Dabei gilt eines: Implementiert ein Target den Level N, dann ist es dem Host nicht erlaubt - und das Target stellt die Routinen dafür auch gar nicht erst zur Verfügung - Kommandos irgendeines Levels kleiner als N zu verwenden. Das sichert die Autonomie des Targets, und die ist wichtig! Es ist in diesem Konzept nämlich durchaus möglich, daß heute ein Host und morgen ein anderer die Betreuung des Targets übernimmt. Dabei kann es zu unerfreulichen Inkonsistenzen führen, wenn ein Host zuviel über ein Target weiß. Jeder Host darf nur soviel wissen, wie das Target es will. Aber das kommt erst später dran.

Target Level 1
(Den Level 0 können wir hier auslassen, da es keine Kommunikation mit dem Host gibt.) Das ist, wie wir oben gesehen haben, das dümmste aller Targets. Aber nichtdestotrotz braucht es etwas Intelligenz, die ihm allerdings in Form eines ROM-Images auf anderen Wegen zuteil geworden ist. Prinzipiell gilt das natürlich auch für alle anderen Target-Levels, aber hier ist es am erkennbarsten. Das Target hat also ein ROM eingesteckt bekommen (ok, es geht auch anders, aber das ist nur ein Bild) und lauert jetzt auf der seriellen Schnittstelle auf Kommandos und Parameter. Als Kommandos sind nur solche implemetiert, die das Speichern und Lesen von Werten im Arbeitsspeicher erledigen. Klar, dazu kommt noch die Übertragung von Parametern auf und von dem Stack des Targets und logischerweise ein Äquivalent von EXECUTE. Aber das ist schon alles. Wobei die Übertragung von Werten auf den Stack, wenn er in den Arbeitspeicher gemapt ist, keine eigene Routinen erfordert. Aber sie können der Einfachheit halber redundant vorhanden sein. Architekturen, die getrennte Speicher für getrennte Sektionen (z.B. page-maped) oder für Code und Daten (z.B. Harvard) vorsehen, müssen entsprechende Routinen für deren Zugriff zur Verfügung stellen. Routinen, die Parameter brauchen, erwarten diese auf dem Stack und dort werden auch die Ergebnisse abgelegt, außer eine Routine hat einen expliziten Seiteneffekt, der das Ergebnis woanders ablegt (z.B. ein STORE oder eine EMIT). All diese Kommandos werden binär an das Target gesendet, also keine ASCII-Strings! Jedes Kommando ist ein Byte lang (oder auch länger, s.o.) und die Parameter werden ebenfalls in diesem Format übertragen. Das dient der Minimierung des Speicherbedarfs des Targets. In der Praxis wird ein Target dieses Levels nur sehr selten wieder eine Kommunikation mit seinem Host haben, da es wahrscheinlich ein Embedded-System ist. Aber für Diagnosezwecke ist eine solche Kommunikation u.U. sehr wichtig. So, und jetzt kommt die "Schockier-die-Forthler-Wahrheit": Diese Kommunikation kann man nicht in UPN machen! Ganz gleich, wie man es dreht und wendet. Entweder gibt man haufenweise Möglichkeiten auf, oder man braucht ein Target, das Metainformationen mit dem Host austauschen kann. Aber erklärt einmal einem Kunden, daß sein Barometer-in-der-Armbanduhr auf einmal Kilobytes an Speicher braucht oder daß der Leistungsmesser für Sportler, die abends ihre Trainingsdaten am PC analysieren wollen, nach ein paar Stunden seine Batterien ausgelutscht hat. Nee, da zählt jedes Byte

und jedes Microampere. Also: KISS (Keep It Simple, Stupid). Beruhigt Euch, die Targets höherer Level werden schon mehr "forthig", weil es da nämlich wirklich Sinn macht. Aber hier, auf diesem Level, eben nicht. Punkt und Basta! Also erfolgt die Kommunikation in genau der umgekehrten Weise - oh, Gott, ja und verdammt, genau wie in solchen Sprachen wie Lisp, also die Operation zuerst und dann die Parameter. Auf dem Level 0 von einem Target ist das allerdings etwas eingeschränkt: Jedes Kommando hat genau einen Parameter. So blödsinnig es für Euch Kameraden auch klingt, aber auf diesem Level schaut ein STORE (ich schreibe es aus, denn das Ausrufezeichen ist manchmal nicht so recht erkennbar) so aus: PUSH_TO_STACK n PUSH_TO_STACK adr PUSH_TO_STACK exec-adr-of-store EXECUTE. Klingt umständlich, braucht aber nur minimale Ressourcen im Target, um das zu implentieren. Aber, wie schon gesagt, solch eine Kommunikation ist nur im Falle einer Diagnose sinnvoll - aber genau dafür auch vorgesehen. (Alle Worte in Großbuchstaben sind Stellvertreter eines Tokens und keine Texte, die dem Target übermittelt werden, s.o.)

Target Level 2

Hier kann das Target schon mehr. Es kann die komplette Verwaltung seines Speichers und seiner Token selbst vornehmen. Also gibt es keine Routinen mehr, die dem Host erlauben, beliebige Speichersegmente zu überschreiben oder zu lesen. Aber trotz allem läuft die Kommunikation zwischen Host und Target in rein binärer Weise ab. Eine typische Kommunikation beim Kompilieren eines neuen Wortes läuft so ab: Host: Hey Target, gib mir eine neues Token, weil wir ein neues Wort lernen werden. (Das Target allokiert ein neues Token und trägt die Startadresse des neuen Wortes in seiner Tokenliste unter der Kennung des neuen Tokens ein.) Target: Hier hast Du es. Host: (macht das für jedes Teilwort der zu kompilierenden Routine) COMPILE TOKENx. Am Ende schickt er das Token END_COMPILE (entsprechend dem Semikolon). Wenn ein Wort auf dem Target ausgeführt werden soll, dann schickt der Host einfach das dazugehörige Token. Und damit sind wir bei einer wichtigen Eigenschaft des Targets: Es kennt keine Unterscheidung zwischen IMMEDIATE-Worten und anderen. Jedes Wort ist IMMEDIATE. Wozu der Quatsch? Einfach deswegen, um auf dem Target Ressourcen zu sparen. Erinnern wir uns, wann IMMEDIATE-Worte gebraucht werden. Nur beim Kompilieren! Und im Leben eines normalen Targets ist das wohl praktisch nie der Fall - von seiner Generierung einmal abgesehen, aber da hat es ja seinen Host. Also ist selbst ein Bit pro Wort (den dazugehörigen Code bitte nicht vergessen) einfach zuviel. Die Verwaltung der Token liegt voll in der Verantwortung des Targets. Zwar schreibt die Kommunikation mit dem Host ein bestimmtes Format vor (derzeit 16 Bit), aber das heißt noch lange nicht, daß das Target tatsächlich soviel dafür allokiert. Es kann genausogut nur 10 oder 12 Bit verwenden und einfach führende Nullen bei der Kommunikation einfügen - der Host erfährt es nie, weil er es nie zu wissen braucht.

Target Level 3

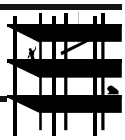
So, jetzt sind wir auf einem Niveau gelandet, wo die meisten wohl das Target als ein eigenständiges System betrachten. Es versteht nämlich nunmehr tatsächlich Kommandos in ASCII. Es kann mit einem Zeichenstrom umgehen, aber eines fehlt ihm immer noch: Es kann keine Kommentare (auch keine Leerzeichen oder so!) überspringen. Folglich braucht es eine Abfolge von ASCII-Texten, die den Namen der Worte entsprechen. Aber es kann jetzt selber zwischen dem Compilemode und dem Interpretermode unterscheiden. Aber dazu muß es doch IMMEDIATE-Worte von den anderen unterscheiden können! Tut es auch, aber auf ein Weise, die nicht sofort vorteilhaft erscheint: Es hat nämlich zwei vollkommen ge-trennte Namenslisten. In der zuerst Durchsuchten stehen die Namen der IMMEDIATE-Worte, und in der Zweiten alle anderen. “Welche Platzverschwendung!” werden manche rufen. Im Prinzip ja, aber wenn es zum Schluß nur darum geht, daß ein anderes System in ASCII mit dem Target kommuniziert und von diesem System aus auch eine Fehlersuche gemacht werden soll, dann kann man nämlich nach dem Kompilieren der Applikation die erste Namensliste komplett löschen und hat immer noch die gewünschte Funktionalität, da Worte der Applikation wohl nicht kompilieren wollen. Aber dazu weiter unten etwas mehr.

Target Level 4

Nun gut, dazu gibt es nicht viel zu sagen, denn so ein Target ist eigentlich ein voll aufgeblasenes Forth-System, mit File-I/O und allem Schnickschnack. Aber eines sollte doch hier angemerkt werden: Auch so ein Target hat eine innere Struktur, als ob es schichtmäßig aus primitiveren aufgebaut wäre. Es hat die Verwaltung seiner Tokenlisten immer noch in eigenen Routinen eingebaut und auch die Namenslisten sind wie in einem einfacheren Target implementiert. Lediglich der "Overhead" der vollen textuellen Eingabe, zusammen mit dem Verständnis von Verzeichnisstrukturen und so, ist hier einfach "aufgepfropft". Warum diese Verschwendung? Einfach deswegen, weil ein solches Target mehr als genug Ressourcen hat, um auch das zu verkraften. Warum nicht einfacher? Weil man dann den Code ändern muß. Und hier sollte eines klargestellt werden:

Warum diese Aufteilung und diese Abgrenzungen?

Um es zu wiederholen: Das oberste Ziel dieser ganzen Architektur ist einfach eines: Man schreibt Code und kann ihn dann einfach den Gegebenheiten, sprich Ressourcen, anpassen. Es soll erreicht werden, daß jede Stufe der Kompilation oder der Ausführung wahlfrei auf dem Host oder dem Target erfolgen kann. Oder anders formuliert: Mit wenigen Änderungen in Konfigurationsdateien, dazu zähle ich jetzt auch INCLUDE-Dateien, muß ein Target beliebigen Levels erzeugt werden können. Ach ja, das Laufzeitverhalten. Klar, wenn ein Target vom Level 3 oder 4 ein Kommando in ASCII bekommt, dann sucht es erst in einer Namensliste das dazugehörige Token und dann in der Tokenliste die eigentliche Startadresse - eigentlich verdammt umständlich. Aber um das zu umgehen, müßte man neue Routinen implementieren und - was viel wichtiger und umständlicher ist - dem Compiler beibringen, welche jetzt tatsächlich verwendet werden sollen. Aber macht das eigentlich einen Sinn?



Wir sollten eines nicht vergessen: Alles, was bisher gesagt wurde, bezieht sich auf die Kommunikation zwischen Host und Target - nicht zwischen Target und Applikation (sprich, seiner Umwelt im normalen Betrieb). Ich selber habe gerade ein Target zu bauen, das (aus Speichergründen) eines vom Level 0 sein muß - also ohne Ahnung von einem Host, aber trotzdem muß es einfache Kommandos (hier 4 Einzeichenkommandos) verstehen. Das ist kein Widerspruch, denn was für eine Applikation kompiliert wird, hat nichts mit dem Vorgang der Kompilation - oder allgemein gesagt - dem Aufbau des Targets zu tun. Das hier vorgestellte Konzept bezieht sich ausschließlich auf den Aufbau von Targets verschiedener Levels und explizit nicht auf die Fähigkeiten der implementierten Applikation. Das heißt, daß u.U. manche Sachen doppelt (na ja, nicht ganz doppelt, aber weitgehend) implementiert werden. Vor ein paar Jahren mußte ich ein Boot-ROM für ein Board schreiben. Klar, das wurde in Forth erledigt und mit dem hier vorgestellten Konzept war das auch ein Klacks. Vorgegeben waren: POST (Power On Self Test) und ein Prompt, von dem aus man einige (Was "einige" alles sein kann, habe ich dann später erfahren - weiterlesen) Tests aufgerufen werden können. Aber dann kamen neue Anforderungen hinzu: Die MAC-Adresse der Ethernetchips sollte über die serielle Schnittstelle programmierbar sein und auf einmal sollten auch die optional vorhandenen DSPs auf Vorhandensein und korrekte Funktion überprüft werden können. Und dann kam die "Schikane". Das ATE-System hat manche Sonderzeichen für sich beansprucht. Damit fiel z.B. das AT-Zeichen aus meinem "Wortschatz" weg. Die Einigung war dann aber dennoch gütlich: Die Applikation verwendet die eine serielle Schnittstelle, und ich die andere. Aber trotzdem war ein Namenskonflikt zu vermeiden, denn das ATE konnte u.U. (es ist meines Wissen nicht passiert, aber der Kunde wollte es so) ein AT-Zeichen schicken, ohne daß irgendjemand es veranlaßt hat (Auf deutsch: Da war ein Bug im ATE-System oder in dessen Benutzern, aber so klar formuliert man das ja nicht mehr). Ergo lag die Aufteilung nahe, die eine Schnittstelle mit dem Target (das, was mit dem Host kommuniziert) und die andere mit der Applikation (das, was mit den Usern kommuniziert) zu verbinden. Beide hatten getrennte Namenslisten und damit gab es keinen Konflikt mehr. Um die Sache kurz zu machen: Nachdem ich meine Aufgabe soweit erfüllt hatte, kam die Anforderung, daß ich 16kB für den eigentlichen Loader der Applikation in meinem 64kB-Segment zur Verfügung zu stellen habe. Und zwar an einer festen Adresse. Ach so - sie meinten auch noch, daß es auch etwas mehr werden könnte. Und es wurde mehr! Gut, man kann schieben und die Lücke verkleinern, aber irgendwann stößt das an eine Grenze. So, und jetzt will ich das hier vorgestellte Konzept nicht übermäßig loben, aber welches Konzept erlaubt es sonst, einfach durch Reduktion des Targets auf einen niedrigeren Level, massig Speicherplatz zu sparen? Schmeiß die ellen-langen Namenslisten raus (soll die doch der Host verwalten) und wenn es oberknapp wird, dann hau auch noch die Tokenlisten raus (ebenfalls Job vom Host). Zugegeben, irgendwann hat auch das eine Grenze: Wenn das Target ASCII-Strings als Kommandos von der Applikation verstehen muß, dann muß man abwägen, wieviel welche Maßnahme bringt.

Aber dazu mehr in einer Fortsetzung.

Zusammenfassung und Klarstellung

Um eines klarzustellen: Das alles hier Geschilderte hat nur mit der Interaktion eines Hostes mit einem Target zu tun. Je mehr das Target selber kann, desto weniger muß der Host können und umgekehrt. Es geht darum, die Grenze zwischen dem Host (der ja wohl meistens auf einem PC mit üppigen Ressourcen läuft) und dem Target möglichst variabel zu machen. Es sollte klar herausgestellt worden sein, daß die hier beschriebene Kommunikation zwischen dem Host und dem Target nicht eine eventuelle Kommunikation zwischen dem Target und der Applikation (hier als die Schnittstelle zum Enduser definiert) ausschließt. Letztere hat einfach über eine andere physikalische Schnittstelle zu erfolgen - Änderungen an dem sind in der Planungsphase: Vorschläge willkommen! Und jetzt die Antwort auf die Frage: "Warum portiere ich nicht einfach ein Forthsystem auf das Target?" Antwort: Weil das zuviel Zeit in Anspruch nimmt. Klar, mit diesem hier kann ich FPC (oder welches System auch immer) nicht portieren. Nein, ich muß es neu schreiben (oder von der Homepage des Autors runterladen). Aber ab dann kann ich dieses System relativ einfach auf verschiedene Targets abbilden. Um es überhaupt ans Laufen zu bringen, braucht es nur, daß etwa 6-12 Routinen in binärer Form vorliegen. Binär heißt hier, daß deren Code in einer Form vorliegt, daß (leider eine derzeitige Einschränkung der derzeitigen Implementation) jedes Byte als ASCII-Hexword vorliegt. Ich mache das derzeit einfach so, daß ich ein Wort in Assembler (vom Hersteller mitgeliefert) schreibe und dann den ASCII-Output der Opcodes mit meinem Editor einfach in das Sourcefile übernehme.

Für Mitdenker

Habt Ihr Euch gefragt, wie denn der Host mit dem Target überhaupt die erste Kommunikation auf Tokenebene aufbauen kann, sprich, wie er sich sicher sein kann, daß selbst das primitivste Target - u.U. noch voll alleinig existent in seinem eigenen Memory - mit ihm kommunizieren kann? Fein, daß Ihr mitgedacht habt. Details kommen zwar erst später, aber soviel sei jetzt schon verraten: Es gibt eine Liste von Standard-Token. Die bekommt jedes Target a priori verpaßt. Und vergessen wir eines nicht: Jedes Target macht bei seiner Kompilierung die Stufen 2-N durch. Noch Fragen: Brief an den Editor und ich schreibe weiter - denkt daran, das Ganze steht unter GPL, aber ich stelle es nur ins Netz, wenn genügend Resonanz da ist. Sonst rentiert sich das einfach nicht. Wenn der Editor und die Leserschaft es wüschen, dann schreibe ich noch ein paar Artikel, die das Innenleben dieses Targetcompilers offenlegen. Aber ich brauche Hilfe: Wer kann Freihandzeichnungen in eine Form umwandeln, die veröffentlicht werden können?

Ulrich Paul

Echelons Neuron

Ein proprietärer in C programmierter Stackprozessor
für Anwendung im Feldbus

Rafael Deliano

Die kalifornische Firma Echelon wurde 1988 mit einer Menge Venture Capital gegründet, um den universell verwendbaren Feldbus LON ("Local Area Network") zu entwickeln. 1991 startete die Vermarktung mit einigem Propagandaaufwand. Die Anwender stellten schnell fest, daß so ziemlich alles an LON proprietär und auch etwas teuer ist.

Vorteile

Andererseits sind die Komponenten von Echelon vorentwickelt und zueinander kompatibel. Angeboten werden Chips, d.h. verschiedene Varianten des Neuron-Prozessors und Software wie das Neuron-C, um diesen zu programmieren. Und natürlich die Protokolle für die Kommunikation. Aber auch Module wie Powerline-Modems, die der Anwender fertig auf seine Leiterplatten setzen kann. Obwohl Draht sicherlich das gängigste Medium ist, ist ein Vorzug von LON die Verfügbarkeit auch anderer Übertragungswege. Daraus resultieren unterschiedliche Anforderungen an Modulationsarten, Fehlererkennung und Fehlerkorrektur. Der Neuron-Prozessor hat diese Flexibilität, da er auch die unteren Protokollschichten in Software ausführt.

Der Anwender kommt damit schnell zu einer Lösung und muß sich um die technischen Probleme der Vernetzung wenig kümmern. Verbreitung hat LON hauptsächlich in den USA gefunden. Speziell in der Gebäudeausrüstung und der Automatisierungstechnik. Für Gebäude hat LON auch in Europa Chancen.

Big Brother

Proprietär bedeutet z.B., daß technische Informationen wie die Innereien des Prozessors nicht veröffentlicht werden. Und wenn ein Anwender entsprechende Unterlagen trotzdem auf seine Webseite stellt, er von Echelon-Anwälten verfolgt wird. Proprietär bedeutet auch, daß es zwar nominell Newsgroups im www zu LON gibt, diese aber nicht benutzt werden. Der Meinungsaustausch findet im firmeneigenen Diskussionsforum statt, wo man nur nach Eingabe eines Passwortes reinkommt. Es darf angenommen werden, daß dort nur "gesäuberte" Nachrichten auftauchen.

ANSI/EIA

Echelon entschied sich 1996, das Protokoll offenzulegen. Lizenznehmer können damit auf andere Prozessoren übergehen, soweit die Implementierung komplett und kompatibel zum Neuron-Chip ist. Allerdings ist die erforderliche Rechenleistung ziemlich hoch, so daß der Kostenvorteil gering ist. Echelon hat zumindest von einer externen Firma eine C-Referenzversion für den MPC68360 entwickeln lassen. Es ist nicht bekannt, ob heute tatsächlich in nennenswertem Umfang andere CPUs als Neuron verwendet werden. Zweck der Übung dürfte eher gewesen sein, Voraussetzungen zu schaffen, daß LON zu einem ANSI/EIA Standard werden konnte. Ein auf offenen Standards basierendes System ist Grundanforderung mancher Kunden.

Neuron

Die ursprünglichen Prozessoren (Tabelle 1) wurden von Motorola mit *second source Toshiba* gefertigt und verkauft. Da sie typisch Firmware von Echolon im ROM haben, waren auch geringe Lizenzgebühren vom Halbleiterhersteller an Echelon zu zahlen. Es wurden zwar einige Millionen Stück verkauft, aber der Prozessor blieb mit $ 3-4 eher teuer und es wurden nicht viel mehr als die beiden ursprünglichen Varianten 3120 und 3150 entwickelt.
Motorola kündigte Anfang 1999 an, die Produktion der ICs 2001 einzustellen.

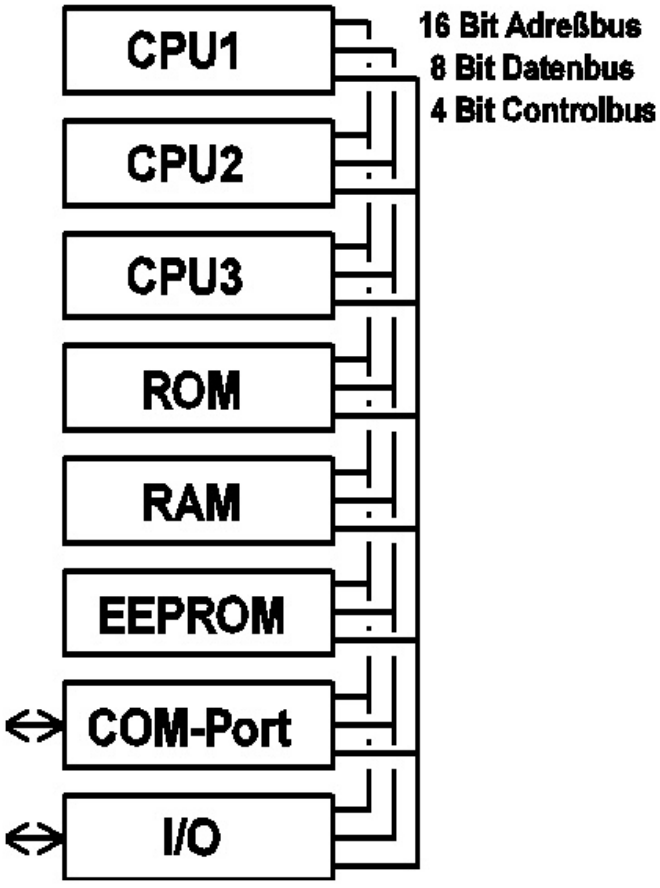
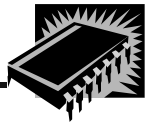


Bild 1: Funktionsblöcke des Neuron-Controllers

Toshiba erklärte 2000 jedoch, für weitere 10 Jahre LON zu unterstützen.



Die ursprünglichen Chips sind aber inzwischen technisch überaltet. Eine Umsetzung auf kleinere Strukturgrößen und damit höhere Geschwindigkeit war dringend erforderlich. Auch sollte FLASH das bisherige EPROM ersetzen.
Andererseits mußten neue Chips zu den bisherigen weitgehend kompatibel bleiben. Es gelang Echelon Mitte 1999 Cypress für diese Erneuerung zu gewinnen, die die Zukunft von LON sicherstellt. Diese ICs sind inzwischen verfügbar (Tabelle 2).

Tabelle 1: ungefähre Kompatibilität

Motorola	Toshiba	Cypress
MC143150	TMPN3150	CY7C53150
MC143120E2	TMPN3120FE3	CY7C53120E2
	TMPN3120FE5	CY7C53120E4

Tabelle 2: Cypress

	ROM	RAM	EEPROM	Takt (MHz)
CY7C53120E2	10 k	2 k	2 k	10
CY7C53120E4	12 k	2 k	4 k	40
CY7C53150	0 k	2 k	0,5 k	20

Tabelle 3: Toshiba

	ROM	RAM	EEPROM	Takt (MHz)
TMPN3120A20M	16 k	1 k	1 k	20
TMPN3120A20U	16 k	1 k	1 k	20
TMPN3120B1AM	10 k	1 k	0,5 k	10
TMPN3120E1M	10 k	1 k	1 k	10
TMPN3120FE3M	16 k	2 k	2 k	20
TMPN3120EE3U	16 k	2 k	2 k	20
TMPN3120FE5M	16 k	4 k	3 k	20
TMPN3150B1AF	0 k	2 k	0,5 k	10

Tabelle 4: CPU-Register

Bits	
8	FLAGS
16	IP Instruction Pointer
8	TOS Top Of Stack
16	BP Base Page
8	DSP Data Stack Pointer
8	RSP Return Stack Pointer

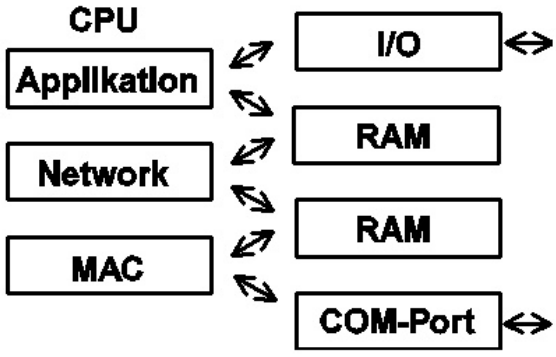


Bild 2: Kommunikation zwischen Prozessen

Tabelle 5: Befehlssatz Program Control

cyc	byte	Name	
1	1	NOP	;No Operation
1	1	SBR	;Short Branch
2	2	BR BRC BRNC	;Branch ; Branch on CARRY ;on /CARRY
3	1	SBRZ SBRNZ	;Short Branch on TOS=0 ;on TOS<>0 (C1 ---)
4	3	BRF	;Branch Far abs
4	4	BRZ BRNZ	;Branch on TOS=0 ;on TOS<>0 (C1 ---)
4	1	RET	;Return from Subroutine
4/6	3	BRNEQ	;Branch if TOS not equal (C1 ---)
5	2	DBRNZ	;Dec (RSP) branch if <>0
5	2	CALLR	;Call relative
6	2	CALL	;Call (nur untere 8kByte)
7	3	CALLF	;Call Far abs

Tabelle 6: Befehlssatz ALU

cyc	byte	Name	
2	1	INC DEC NOT	;+1 -1 Invertierung
2	1	ROLC RORC	;Rotate TOS durch Carry
2	1	SHL SHR	;Logic shift
2	1	SHLA SHRA	;Aritmetic Shift
4	1	ADD ADC AND OR XOR	;mit NEXT
3	2	ADD AND OR XOR	;#literal mit Konstante
7	1	ADD AND OR XOR _R	;mit NEXT dann Rücksprung
4	1	SUB SBC	;NEXT - TOS = TOS
4	1	SUB	;TOS - NEXT = TOS
3	1	ALLOC #literal	;+1... +8 zu Stackpointer
6	1	DEALLOC #literal _R	;-1 .. -8 zu Stackpointer dann Rücksprung
4	1	XCH	;"Echange" = SWAP
6	1	INC [PTR]	;Inkrement PTR-Zeiger 0 ...+3

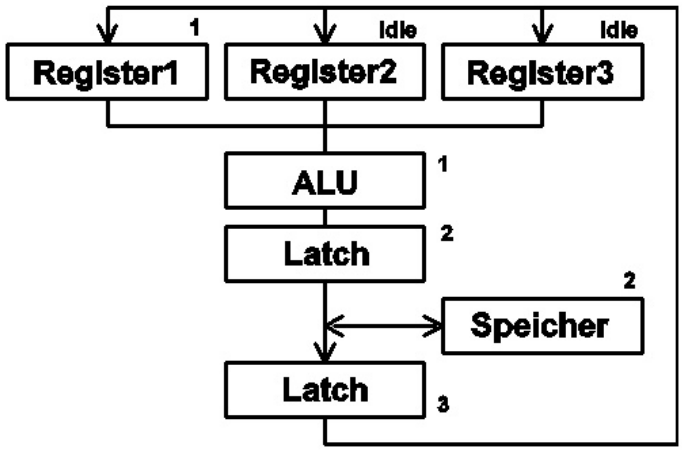


Bild 3: Speicherverteilung in der Base Page

Tabelle 7: Befehlssatz Arithmetik

cyc	byte	Name	
3	1	PUSH TOS	DUP
3	1	DROP TOS	DROP
6	1	DROP _R TOS	DROP und Rücksprung
4	1	PUSH NEXT DSP RSP FLAGS	CPU-Register auf Stack
4	1	POP DSP RSP FLAGS	Stack auf CPU-Register
2	1	DROP NEXT	= SWAP DROP
5	1	DROP _R NEXT	und Rücksprung
4	1	PUSH POP !D	Register Byte @ !
4	1	PUSH POP !TOS	Adresse = BP + TS ; Speicher @ !
4	1	PUSH [RSP]	Von Return Stack auf Stack; RSP unverändert
2	1	DROP [RSP]	Inc RSP
4	1	PUSHS #literal	Short Literal 0 ...7 auf Stack
4	2	PUSH #literal	Literal = Byte auf Stack
5	1	PUSHPOP	Von Return Stack auf Stack
5	1	POPPUSH	Von Stack auf Return Stack
5	3	LDBP addr BP	mit Literal 16 Bit laden
5	2	PUSH POP [DSP] [-D]	EA = BP + DSP - Konstante 1...8
6	3	PUSHD #literal	Double Literal 16 Bit
6	1	PUSHD [PTR]	@ via PTR Zeiger
6	1	POPD [PTR]	! via PTR Zeiger
6	1	PUSH POP [PTR] [TOS]	EA = (PTR) + TOS @ !
7	1	PUSH POP [PTR] [D]	EA = (PTR) + Konstante 0 ... 255 @ !
7	3	PUSH POP	abs @ 1 mit 16 Bit Adresse
7+4n	1	IN OUT	Übertragung von n Bytes zu/von I/O

Architektur

Die Neuron-Controller haben intern 3 CPUs, die über einen gemeinsamen Bus auf Speicher und I/O zugreifen (Bild 1). Der Application-Prozessor wird vom Anwender programmiert. MAC- ("Media Access Control") und Network-Prozessor werden von der von Echelon fertig gelieferten Protokoll-Firmware betrieben. MAC bearbeitet die hardwarenahen Teile der Datenübertragung, der Network-Prozessor macht das Protokoll. Die CPUs kommunzieren über RAM miteinander.
(Bild 2).

CPUs

Alle 3 CPUs sind identisch aufgebaut. Sie haben eigene Register (Tabelle 4). Da es sich um Stackprozessoren handelt, gibt es keinen Akku, sondern das oberste Byte des Stacks (TOS) befindet sich in der CPU, Rest des Stacks im RAM (Bild 4). Als NEXT wird hier der zweite Wert unterhalb des TOS auf dem Stack bezeichnet. Dabei wird über die Base Page BP eine im 64k Speicher frei verschiebbare Zeropage adressiert. Diese enthält beide Stacks, Registervariablen und Zeiger. Die Einheitlichkeit der CPUs ergibt sich daraus, daß ihre Bestandteile gemultiplext betrieben werden (Bild 3). Die Nummern im Bild geben an, welche Teile in diesem Augenblick gerade für welche CPU arbeiten. Es wird kontinuierlich zwischen allen 3 CPUs umgeschaltet.

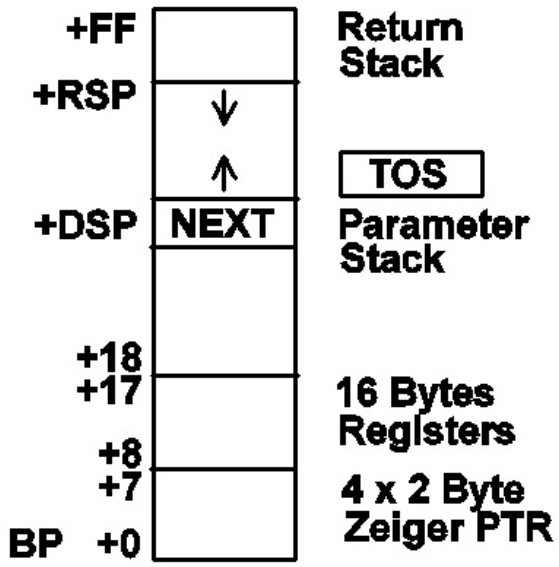


Bild 4: gemultiplexte CPU

Praktisch hat man so einen starren Zeitscheiben-Multitasker für 3 Tasks. Jedoch keinen Overhead durch Umspeichern von Registern. Und eine nicht wahrnehmbare Umschaltzeit, da Opcodes aller 3 CPUs praktisch gleichzeitig verarbeitet werden. Kehrseite ist, wie bei anderen Multiprozessor-Systemen aber, daß alle 3 CPUs selten voll ausgelastet sind. Wenn keine Nachrichten im Netzwerk laufen, ruhen 2 von 3 CPUs. Auf einem konventionellen, entsprechend größeren Prozessor mit Multitasker hätte dann die Applikation volle Rechenleistung.

Für LON dürfte allerdings entscheidend gewesen sein, daß man durch die kurze Taskwechselzeit viel von der Bitstopselei der untersten Ebene flexibel in Software im MAC machen kann und die I/O-Logik primitiv gehalten werden kann.
Außerdem hat durch die klare Trennung in einzelne CPUs eine fehlerhaft funktionierende Applikation weniger Möglichkeiten, das Netzwerk zu beeinträchtigen.

Befehlssatz

Aufgeteilt nach Funktionen in Tabellen 5, 6, 7. Irgendwie spürt man den NOVIX. Vgl. den DBRNZ-Opcode oder Mehrfachfunktionen im Opcode wie bei _R . Diese Option leitet bei manchen Befehlen den Return ein und spart damit einen folgenden RET-Opcode. Und einiges an Speicher, wenn man FORTH-artig alles in viele keine Unterprogramme zerlegt.

Bei den Sprungbefehlen in Tabelle 5 hat ein normaler Branch Sprungweite -128 ... +127 , Short Branch aber nur 0 ... +15.

Der Befehl CALL erreicht nur die unteren 8 kByte des 64k Speichers. Es gibt anscheinend kein Overflow-Flag; schlecht für Arithmetik mit Vorzeichen.

Neuron C

Es gibt keinen Assembler für den Prozessor. Der C-Compiler erzeugt allerdings als Option ein Assemblerlisting, um dem C-Programmierer die Optimierung der Source zu ermöglichen. Neuron C ist stark modifiziertes C. Es gibt nützliche Erweiterungen.
Ein Multitasker ermöglicht die Aufspaltung in Tasks und deren Prioritätszuordnung. Es gibt Befehle für den Zugriff auf Sekunden- und Millisekunden-Timer und die I/O. Und natürlich für die Kommunikation mit dem Netzwerk.
Andererseits ist es ein sehr abgespecktes C: int ist 8 Bit und long 16 Bit. Damit dürfte kein konventionell geschriebenes C-Programm portierbar sein.
Der Compiler gilt bezüglich Codedichte als effizient. Geschwindigkeit ist ein anderes Thema. Mit 10 MHz Takt liegt die Ausführung eines Assemblerbefehls bei 0,6 ... 4,2 usec. Die Ausführungszeiten von HLL sind entsprechend gemächlich. Jemand hat sich mal für den Hausgebrauch ein FORTH in C programmiert: paßte in 4k. Aber die Befehlssequenz 1 1 + . dauerte 1 sec. Trotzdem zeigt der Chip, daß man C auf einen Stackprozessor compilieren kann.
Bei der Beschaffung der Unterlagen war Klaus Kohl behilflich, der den Stackprozessor auch ursprünglich auf einem Treffen der Münchner FORTH-Gruppe einmal kurz vorgestellt hat.

Rafael Deliano

[1] Cypress "Neuron Chip Technical Reference Manual v1.0"

Fletcher Prüfsumme

Rafael Deliano

In Software ohne Tabellen schneller berechenbar und fast so wirksam wie CRC.

Erste breitere Anwendung erfolgte in OSI-Protokollen. Da Pakete aus Bytes bestehen, wurde hier als Wortbreite der Daten 8 Bit verwendet. Wie auch in [1] als gängigste Variante angenommen.
Bild 1 zeigt schematisch den Ablauf. Es gibt zwei Akkumulatoren LB und HB, deren Länge dem Datenwort also einem Byte entspricht. Nach Ende der Berechnung werden die Akkumulatoren zur Prüfsumme, hier also mit 16 Bit Länge zusammengesetzt. Typisch wird diese little-endian dem Paket angehängt.

Es sind nur Additionen erforderlich, aber Berechnung in 1er-Komplement ist besser als in 2er-Komplement und deshalb üblich [1]. In Bild 1 ist diese etwas andere Addition durch "+" gekennzeichnet.

1er-Komplement

Eine recht ungebräuchliche Variante der Zahlendarstellung (Bild 2), die Arithmetik üblicher CPUs verarbeitet 2er-Komplement direkt. Für 1er-Komplement ist hier die nachträgliche Addition des Carrybits nötig.

Beispiel:

```
  FE  \ -1
+ 04  \ + +4
————
0102  \ 02 ; carry

  02
+ 01  \ add carry
————
  03  \ = +3
```

Bei 8 Bit CPUs ist damit nur ein zusätzlicher Opcode fällig:

```
FE #. LDA, \ -1
04 #. ADD, \ +4
00 #. ADC, \ +carry
```

Da man in FORTH mit 16 Bit Wortbreite rechnet, kann man sich kompliziertere Programme ausdenken, bei denen das Carry-bit von bis zu 255 Bytes akkumuliert wird, bevor man es in einem Korrekturschritt zum unteren Byte summiert. Auf einem Controller dürfte jedoch der simple Algorithmus in Assembler eine deutlich effizientere Lösung sein.

Null

Wenn man bei CRC mit Startwert Null alle Bytes eines Pakets und die Prüfsumme addiert, ist das korrekte Ergebnis Null. Es hat sich eingebürgert, dieses Verhalten auch anderen Prüfsummen aufzuzwingen. Dazu muß man die Prüfsumme passend doktern, bevor man sie dem Paket anhängt. Als Grund dafür wird angegeben, daß Mehraufwand im Sender sinnvoll ist, wenn er zu geringerem Aufwand im Empfänger führt. Denn ein Paket wird nur einmal erstellt, aber beim Weg durch ein Datennetz wird die Prüfsumme oft überprüft.
Wenn alle Bytes des Pakets Null sind, ist auch die Prüfsumme Null. Auch dagegen gibt es Einwände, sodaß für Fletcher ein Patch üblich ist; 00 auf FF zu ändern.
Beide Modifikationsschritte sind in Bild 3 dargestellt. Nur im Sender benötigt; Empfänger "addiert" beide Bytes konventionell, um das Ergebnis Null zu erhalten.

Ein „einfacheres CRC“

Im Listing FLET1 sind die Teile des Algorithmus nochmal in 6502-Assembler und nanoFORTH für 8 Bit Fletcher aufgeführt.

Fletcher-16

Für Internetanwendungen, z.B. RFC-1146 wird alternativ zu 8 Bit die 16 Bit Fletcher mit 32 Bit Prüfsumme favorisiert. Weil auf grossen CPUs effizienter [2]. Die Akkumulatoren sind nun 16 Bit breit, am Rechenschema ändert sich nichts. Da aber nun 16 Bit Worte eingelesen werden, muß man in Paketen ungerader Länge das letzte Byte um 00h ergänzen. Big/little-endian-Probleme sind natürlich auch noch möglich.
Anders als bei ISO wird die Prüfsumme nicht so modifiziert, daß die Summe Null ergibt.

Adler-32

In RFC-1950 gibt es von 16 Bit Fletcher eine aufwendige Variante, die bei etwas geringerer Fehlersicherheit immer noch effizienter als CRC32 in Software sein soll.
Bei Fletcher wird der Moduloüberlauf der beiden Akkumulatoren ignoriert. Man kann den Inhalt der Akkumulatoren auch als Rest einer Division durch 65536 interpretieren.
Bei Adler ist der Rest einer Division durch die Primzahl 65521 vorgesehen; in Bild 4 durch den Befehl "mod" ,der nur den Rest als Ergebnis hat, dargestellt.
Durch passende Auslegung des Programms ist es wohl möglich, die Division nur nach 5552 Bytes ausführen zu müssen. Trotzdem scheint das Verfahren für kleine Controller nicht attraktiv.

Rafael Deliano

[1] Fletcher "An Arithmetic Checksum for Serial Transmission" IEEE Trans. On Com. Jan. 1982
[2] Sklower "Improving the Efficiency of the OSI Checksum Calculation"
ACM Computer Communication Review Okt. 1989


```
<| \ FLET1

1 ZVARIABLE HB
1 ZVARIABLE LB

: FLET! \ ( --- )
  0 HB C! 0 LB C! ;

: FLET@ \ ( --- UN1 )
  HB C@ 8<SHIFT LB C@ OR ;

:CODE FLET+ \ ( UC1 --- ) assembler
  LB LDA,
     CLC,
  BOT- ,X ADC, \ load from Stack
  00 #. ADC,
  LB     STA,
         CLC,
  HB     ADC,
  00 #. ADC,
  HB     STA,
  INX,  INX, \ remove from Stack
  RTS,
CODE;

: FLET+ \ ( UC1 --- ) simple
  LB C@ + DUP FF00 AND IF 00FF AND 1+ THEN
  DUP LB C!
  HB C@ + DUP FF00 AND IF 00FF AND 1+ THEN
  HB C! ;

: FLET+ \ ( UC1 --- ) other version
  LB C@ + DUP FE U> IF FF - THEN DUP LB C!
  HB C@ + DUP FE U> IF FF - THEN HB C! ;

: FLETCH \ ( LastAddr 1.Addr --- UN1 )
  FLET!
  DO I C@ FLET+ LOOP
  FLET@ ;

: FLET@' \ ( --- UN1 )
  FF
  LB C@ HB C@ + DUP FF00 AND IF 00FF AND 1+
  THEN \ 1er-Komplement-Addition
  - FF AND
           DUP 00 = IF DROP FF THEN \ LB
    HB C@ DUP 00 = IF DROP FF THEN \ HB
  8<SHIFT OR ;

\ NH. ( UN1 --- ) print as hex

: Test \ ( --- )
  CR FLET!
  00 FLET+ 00 FLET+ 00 FLET+ FLET@' NH. ." FFFF"
  CR FLET!
  AB FLET+ CD FLET+ EF FLET+ 01 FLET+ FLET@'
  NH. ." F89C"
  CR FLET!
  14 FLET+ 56 FLET+ F8 FLET+ 9A FLET+ 00 FLET+
  01 FLET+ FLET@' NH. ." DC24" ;
|>
```

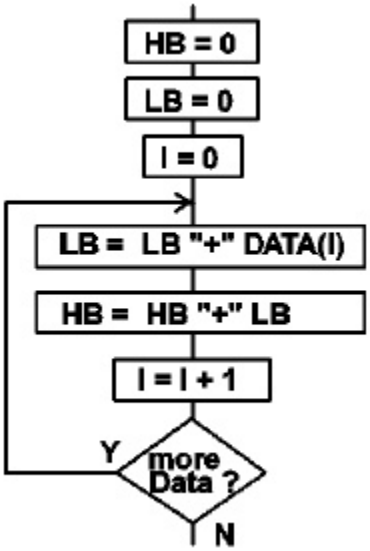


Bild 1:
Berechnung der Prüfsumme

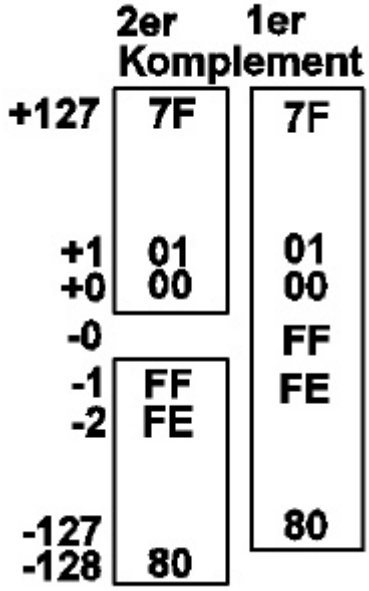


Bild 2:
1er/2er Komplement 8 Bit

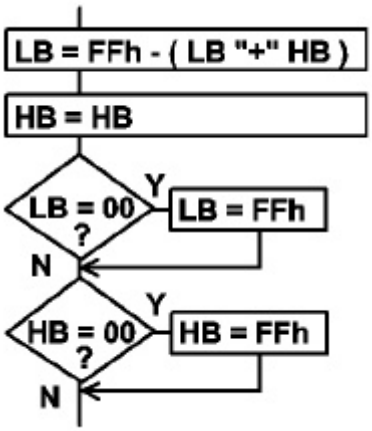


Bild 3:
Modifikation des Endwertes

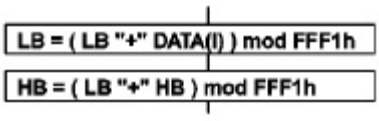


Bild 4:
Adler Variante

Catch & Throw

Filippo Sala

1. Allgemeines

Dieser Bericht befaßt sich mit den ANSForth Worten CATCH und THROW. Das Verstehen ihrer Funktionsweise und ihrer Handhabung ist nicht einfach und erfordert einige Zeit. Hierzu möchte ich meine Erfahrungen den Lesern der VD mitteilen.

2. Was machen CATCH und THROW?

CATCH und THROW ermöglichen eine transparente und äußerst flexible Fehlerbehandlung. Sie stellen unter anderem eine beträchtliche Erweiterung der bekannten Worte ABORT und ABORT" dar.

Mit CATCH werden diejenigen Worte aufgerufen, die fehleranfällig sind. CATCH erwartet auf dem Stack die CFA (ANSForth: xt) des Wortes und startet es mit EXECUTE. Ist während der Execution kein Fehler aufgetreten, so liefert CATCH selbst eine Null, sonst die von THROW gelieferte Zahl. Diese Zahl stellt die Fehlernummer dar.

THROW wird nach Worten aufgerufen, die eine Fehlernummer auf den Stack liefern. Liegt auf dem Stack eine Null, so wird die Ausführung fortgesetzt und THROW hat praktisch keine Wirkung. Liegt auf dem Stack eine Zahl, so wird die Ausführung sofort abgebrochen und nach dem CATCH fortgesetzt, wobei die Zahl auf dem Stack weitergereicht wird.

Der Textinterpreter in QUIT, (üblicherweise INTERPRET genannt) ist besonders kritisch. Darum wird er in ANSForth mit CATCH aufgerufen.

```
[`] interpret ( xt ) CATCH ( e# ) ?dup
  IF
    .fehlermeldung \ Fehlertext ausgeben
  THEN
```

Die Tabelle 9.2 der AnsForth-Spezifikation enthält die vorgeschriebenen Fehlermeldungen mit den Nummern -1 bis -58. Eigene Fehlermeldungen können ab Nummer -256 definiert werden. Nach ANSForth erfolgt also die Fehlermeldung nach der Rückkehr nach QUIT. Früher war es mit ABORT" umgekehrt. Die Fehlermeldungen sind in QUIT sozusagen zentralisiert worden.

3. Ausgabe der Fehlermeldung

Wegen der Zentralisierung der Fehlermeldungen ist die unmittelbare Ausgabe (zum Beispiel die Meldung von ABORT" ....."") nicht mehr möglich. Der Counted-String muß zuerst in eine Variable gespeichert und später in QUIT ausgegeben werden.

```
VARIABLE (cstring) \ Countedstring-Adresse
: .cstring ( -- ) (cstring) @ count type ; \ Ausgabe
```

Das Wort .fehlermeldung besteht aus einer Folge von OF-ENDOF Anweisungen. Hier ein Auszug:

```
: fehlermeldung ( n -- )
 CASE
    -1   OF                                ENDOF \ ABORT
    -2   OF .cstring                       ENDOF \ ABORT"
    -13  OF .cstring ." unbekannt"         ENDOF \ ?
    -14  OF .cstring ." nur compilierbar" ENDOF
    -15  OF .cstring ." ist geschützt      ENDOF \ Fence
    -22  OF ." Strukturfehler"           ENDOF \ IF,THEN..
ENDCASE
;
```

Die Fehlernummern -1 und -2 sind für ABORT bzw. ABORT" reserviert. Die neuen Definitionen lauten:

```
: ABORT ( -- ) -1 THROW ;
    \ abbrechen und weiter nach dem CATCH
```

ABORT" compiliert üblicherweise das Runtime-Wort (abort").

```
: (abort") ( flag -- ) ( R: cstring )
  IF
    R> @ (cstring) !   \ Adresse von cstring speichern
    -2 THROW           \ abbrechen zurück nach dem CATCH
  ELSE
    R> COUNT + aligned >R \ cstring überspringen
  THEN
;
```

Nachfolgend einige Worte als Beispiele für die neuen Definitionen:

```
: unbekannt ( cstring -- ) (cstring) !     -13 THROW ;
: ?comp ( -- ) STATE @ 0= IF              -14 THROW THEN ;
: ?geschuetzt ( adr -- ) fence @ > IF      -15 THROW THEN ;
: ?pairs ( n1 n2 -- ) - IF                 -22 THROW THEN ;
```

4. Wozu das Ganze?

Es scheint, als seien CATCH und THROW nur eine unnötige Kosmetik. Dem ist natürlich nicht so. Man kann nämlich mit CATCH sehr leicht und **ohne Tricks:**

a. gezielt besonders kritische Worte aufrufen, und danach eine komplizierte Fehlerbehandlung ausführen, bevor man mit THROW nach QUIT zurückkehrt.

b. einen mehrfachen Aufruf von CATCH ausführen (z.B. mit INCLUDED Forthquellen laden, wobei INCLUDED wiederholt aufgerufen wird). Bei Fehlern sorgt THROW dafür, daß alle CATCH nacheinander abgearbeitet werden. Alle geretteten Daten der Quelle (Input Source Spezifikation) werden nacheinander restauriert und die geöffneten Dateien geschlossen, bevor man bei QUIT landet;

c. gar nicht zu QUIT zurückkehren, sondern den Fehler beheben oder ignorieren und mit CATCH das Wort erneut aufrufen.

Sicher war das früher auch alles möglich, aber etwas umständlicher. Die Fehlerbehandlung wurde zum Beispiel mit der Variable (error) gehandhabt: Die Worte ABORT und ABORT" führten (error) @ aus:

- Inhalt von (error) sichern
- Adresse der Fehlerroutine in (error) schreiben
  — (diese muß am Schluss den Inhalt von (error) restaurieren)
- Wort aufrufen und hoffen, dass (error) nicht von anderen Worten geändert wird

Es war sehr umständlich!
Mit CATCH und THROW wird alles flüssiger und transparenter.
Die Fehlermeldungen sind alle zentral zusammengefaßt und nach Bedarf leicht zu ändern.

5. Wie werden CATCH und THROW implementiert?

Bevor CATCH ein Wort mit EXECUTE ausführt, müssen wichtige Zeiger (Exception Frames) gerettet werden. Der Datenstackzeiger (und eventuell andere wichtige Zeiger oder Daten) werden auf den Returnstack gerettet. Der Returnstackzeiger selbst wird in eine Variable gespeichert.

```
VARIABLE (catch_rp) \ Returnstackzeiger von CATCH
```

Nachfolgend eine mögliche Definition von CATCH und THROW, wie sie im Anhang der ANSforth-Spezifikationen beschrieben wird:

```
: CATCH ( xt -- 0 | e# )
  SP@ >R             ( xt  ) \ Datenstackzeiger retten
  (catch_rp) @ >R    ( xt  ) \ letzter catch_rp retten
  rp@ (catch_rp) !   ( xt  ) \ Returnstackzeiger speichern
  ( xt ) EXECUTE     (     ) \ ( R: adr1 )
  R> (catch_rp) !    (     ) \ letzter catch_rp restaurieren
  R> DROP            (     ) \ Datenstackzeiger droppen
  0                  ( 0   ) \ ( R: adr2 ) normale Ausführung
;
```

Ist kein Fehler aufgetreten, so wird die Ausführung nach dem EXECUTE (adr1) fortgesetzt. Die Zeiger werden restauriert und CATCH liefert eine Null.

Ist ein Fehler aufgetreten, so restauriert THROW die Zeiger und legt die Fehlernummer auf den Datenstack.

```
: THROW ( 0 | e# -- | e# )
  ?DUP                 \ Wenn 0 ist THROW wirkungslos
  IF
    (catch_rp) @ RP!  ( e# )
                       \ Catch-Returnstackzeiger laden
    R> (catch_rp) !   ( e# )
                       \ letzter catch_rp restaurieren
    R>                ( e# sp )    \ Datenstackzeiger holen
    SWAP >R           ( sp )       \ Fehlernummer retten
    SP!               ( xt )  \ Datentstackzgr. restaurieren
    DROP R>           ( e# ) \ Fehlernummer auf dem Stack
  THEN                       \ ( R: adr2 )
;
```

Die Restaurierung des Returnstackszeigers bewirkt, daß die Ausführung automatisch nach dem CATCH (adr2) und nicht nach dem EXECUTE (adr1) fortgesetzt wird.

Sehr trickreich sag i !!

In einem nächsten Beitrag werde ich über die praktische Implementierung von CATCH und THROW in einem bestehenden System berichten.

Filippo Sala

- 20 Jahre Forthgesellschaft -

2004 ist ein Jubiläumsjahr!

USB-Entwicklung mit FORTH

Carsten Strotmann

In diesem Artikel möchte ich von unserem Projekt, die Erstellung eines USB-Adapter für einen 8-Bit Rechner, berichten. Die Treibersoftware für diesen USB-Adapter ist in FORTH erstellt.

Obwohl in diesem Projekt ein recht betagter 8-Bit Homecomputer (Atari XL/XE) verwendet wird, lassen sich die grundlegenden Informationen für jede Art von USB-Projekten mit Mircoprozessoren nutzen.

Wie alles anfing

Im Oktober 2002, auf der jährlichen Hauptversammlung des ABBUC Atari 8-Bit Computerclubs (Atari Bit Byter User Club, http://www.abbuc.de), sprach mich der holländische Club-Kollege Guus Assmann an. Er hatte einen USB-Anschluss für den Modulschacht des Atari 8-Bit Rechners entwickelt, konnte diesen aber Mangels Software nicht testen.
Ich war in dieser Zeit auf der Suche nach einem sinnvollen Projekt, um mein im Jahre 2000 begonnenes FORTH-"Hobby" weiter auszubauen. Beides passte zusammen, und so war im Oktober letzten Jahres schnell der Entschluss gefallen, für die USB-Hardware Treiber in FORTH zu erstellen.

Der Rechner

Der USB-Adapter wird an einem ATARI 8-Bit Computer betrieben (ATARI XL/XE, Bj 1979-1989). Der Atari besitzt eine 6502 CPU, welche mit 1.7 Mhz getaktet ist, sowie 48 bis 1 MB Hauptspeicher (64 KB direkt adressierbar).

Die USB-Hardware

Der USB-Adapter besteht aus einer einfachen, zweiseitigen Platine für den ATARI-Modulport. Hauptbestandteil der Platine sind zwei USB-Node-Controller der Firma National Semiconductor, NS9603. Diese USB-Controller erlauben die Kommunikation mit einem USB-Host-Controller (z.B. in einem PC), jedoch keine Kommunikation mit anderen USB-Endgeräten (Drucker, USB-Speicher etc.). Es wird das USB 1.1 Protokoll benutzt, da dieses für die Zwecke des Projektes voll ausreicht. Zu einem späteren Zeitpunkt ist geplant, den Adapter mit einem USB-Host-Kontroller auszustatten, um eine Kommunikation mit anderen USB-Endgeräten zu ermöglichen.

Die Software

Für die FORTH Entwicklung wurde ein vom original FIG-6502-FORTH abstammendes FORTH für den ATARI 8-Bit Rechner benutzt. Die Entwicklung fand ausschliesslich auf den Atari-Rechnern statt (keine Emulatoren auf PCs).

Bisher wurde ein interaktives Entwicklungs-Werkzeug namens USBTOOL entwickelt, mit dessen Hilfe die USB-Kommunikation und die USB-Kontroller ausgetestet werden. Der FORTH-Programmaufbau des USB-Tools ist noch nicht optimiert. Der Programmcode wurde direkt von einem C-Programm aus dem USB-SDK von National Semiconductor nach FORTH übertragen. Eine Überarbeitung des Programms, um FORTH-Vorzuege besser auszunutzen, ist als einer der nächsten Projektschritte geplant.

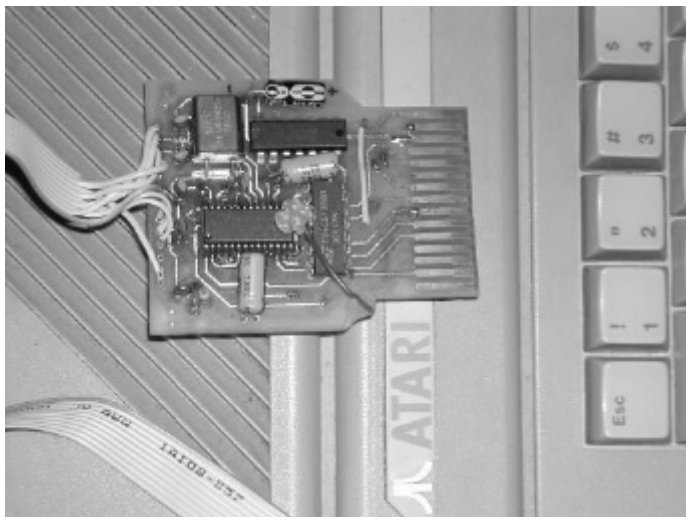
Das FORTH-Programm kommuniziert mit den USB-Controllern über je zwei Speicherstellen, die von den USB-Kontrollern in den Speicherbereich des 6502 Prozessors eingeblendet werden. Über das paralelle Interface der Kontroller wird über das Address-Register das interne USB-Kontroller-Register adressiert und dieses dann über das Daten-Register gelesen oder beschrieben. (Siehe Kap. 3.0 des NS9603-Kontroller Datenblattes).

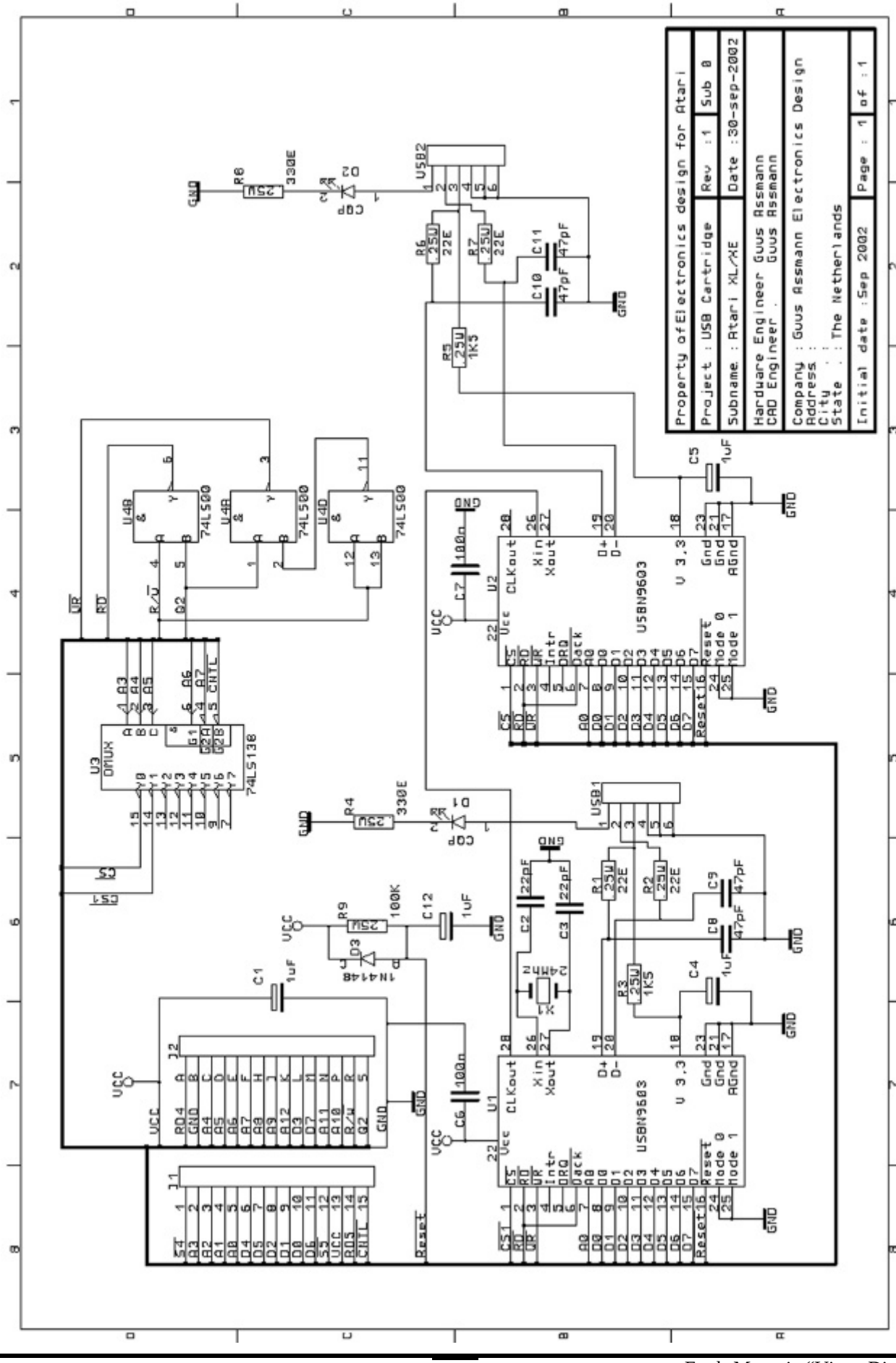
Der USB-Kontroller übernimmt eine Vielzahl der Low-Level-USB-Funktionen, so daß sich die FORTH-Software um die direkte Verarbeitung der USB-Meldungen kümmert.

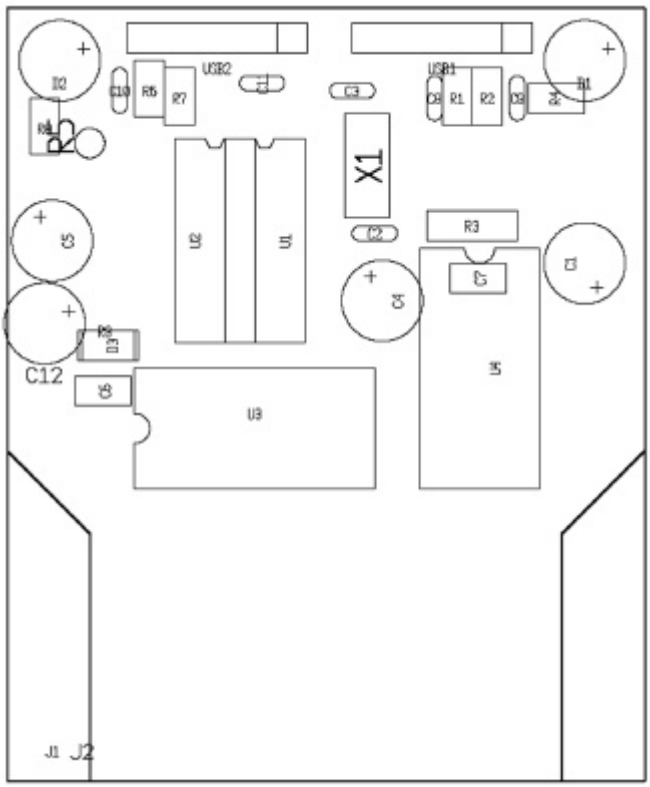
Ziel der vorliegenden Software ist es den Atari als HID-(USB Human Interface Device)-Joystick an einem PC/Mac anzuschliessen. Dieses Ziel ist noch nicht abgeschlossen, derzeit erkennt der PC einen USB-Joystick, kann aber noch keine X/Y-Koordinaten lesen. Die Software wird, wenn die Zeit es erlaubt (Hobby), regelmäßig weiterentwickelt.
Aktuelle Versionen können immer auf der Projekt-Homepage im Internet gefunden werden.

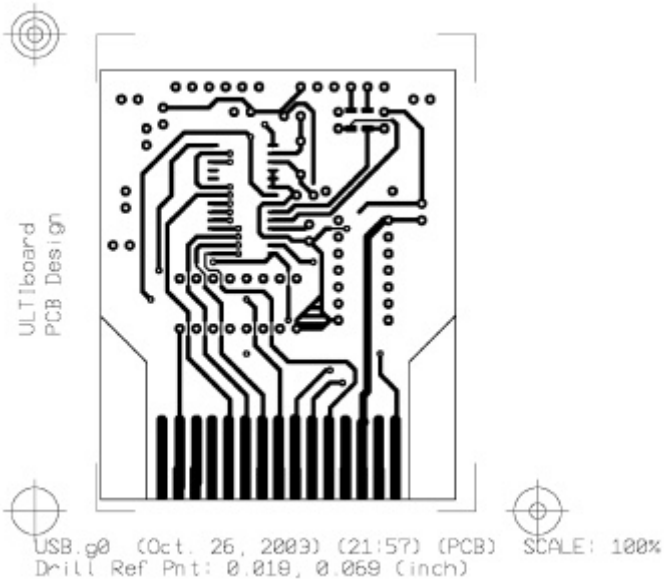
Auf dieser Projekthomepage (http://www.strotmann.de/twiki/bin/view/APG/ProjUSBCart) befinden sich weiterführende Informationen zu dem Projekt, inkl. der jeweils aktuellen Schaltpläne und Datenblätter.

Carsten Strotmann
carsten@strotmann.de

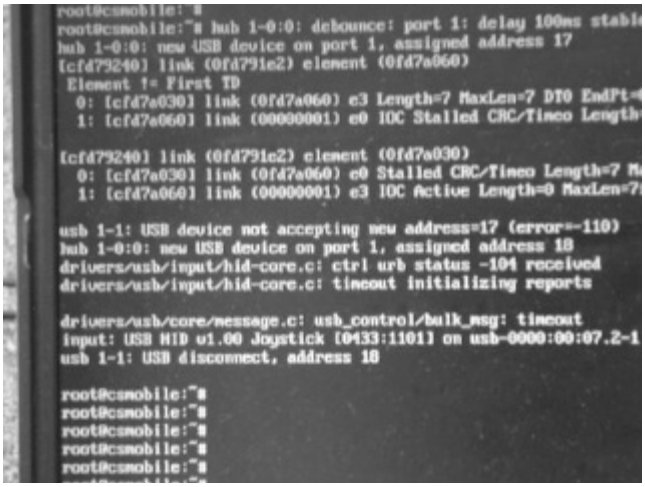



Property of Electronics design for Atari
Project : USB Cartridge
Rev : 1
Sub 0
Subname : Atari XL/XE
Date : 30-sep-2002
Hardware Engineer Guus Assmann
CAD Engineer . Guus Assmann
Company : Guus Assmann Electronics Design
Address :
City :
State : : The Netherlands
Initial date : Sep 2002
Page : 1
of : 1
U5BN9603
74LS00
74LS138
74LS245
1N4148
GND
VCC
Reset



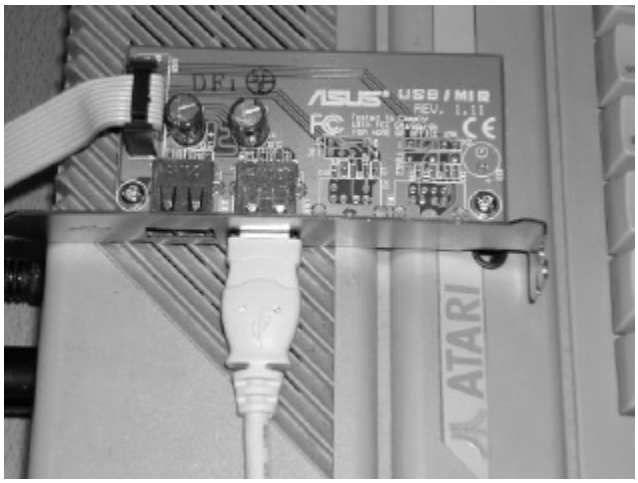


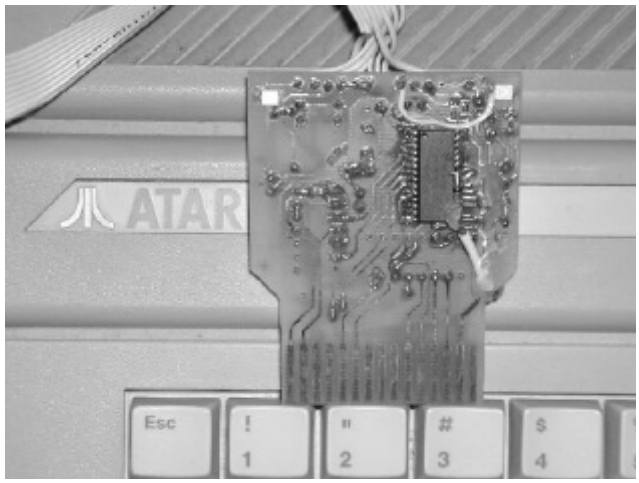
Klein & Fein – Layouts in verarbeitbarer Qualität gibt es bei Carsten Strotmann (im Beitrag genannte WEB-Adresse).



Der LINUX-Kernel „sieht“ den Joystick
Anschluß auf dem Slotblech

USBTOOL arbeitet im ATARI
Kleine Hardware-Bastelei





Wie ein Drache in die FG kam...

Drachen - wir kennen sie von klein auf an: Flitze Feuerzahn, der von seinen Dracheneltern allein im Wald zurückgelassen wurde, Grisu, der mehr am Feuerlöschen als am Feuerspeien interessiert ist und deshalb lieber Feuerwehrmann werden will, und Tabaluga, auf seinem Weg erwachsen zu werden.
Aber wir kennen sie auch aus chinesischen Märchen, Geschichten und Legenden in denen sie meist in Form von schlangenähnlichen Monstern – manchmal auch mit Beinen oder Flügeln ausgestattet - vorkommen, die des Feuerspeiens mächtig waren und oft wundersame Zauberkräfte besaßen.

Drachen sind Geschöpfe unserer Fantasie und stehen für Kreativität. Sie stehen aber auch für Mut und Durchhaltevermögen. Klar dass wir den Drachen - wohlgemerkt ein grüner - als Maskottchen unserer Gesellschaft gewählt haben. Klar, alle die die Geschichte um den Swap kennen, wissen es. Das Zusammentreffen war zufällig - aber dennoch: Liebe auf den ersten Blick.

Zu unserem Jubiläum in Fehmarn würden wir gerne alle Verwandten und Bekannten unseres Swap einladen. Also bringt bitte alles mit was ihr da so zu Hause habt zum Thema "Grüner Drachen." Genauso ist es mit Ideen für neue Themen, Witzen, Grüßen und Rätselspaß dazu, für unsere Zeitung.
Bis dann...

Euer Drachenhüter und Mitglied des Festkomitees

M. Kalus

Leserbrief:

Dear Fred,

ich danke Ihnen für Ihre Antwort. Was meine Bitte (an die Leser der Vierten Dimension) über Forth-Expert-System-Shells und Toolkits und andere Forth-Artificial-Intelligence-Software betrifft, gilt: Wenn ich für die nächste Ausgabe der VD zu spät dran bin, macht das gar nichts.
Ich kann warten. Ich habe über diese Dinge nun schon monatelang nachgedacht. Nun kommt es auf einen weiteren Monat auch nicht mehr an.

Woran ich hauptsächlich interessiert bin, ist die Beantwortung folgender Fragen:

Verwendet irgendwer Forth für Anwendungen in Künstlicher Intelligenz, speziell für Roboter und Echtzeitprobleme?

Verwendet irgendjemand Forth-KI in seiner Arbeit mit Studenten?

Weiß irgendwer, ob es preisgünstige oder kostenlose Software für Forth-KI-Entwicklungen gibt?

Warum frage ich?:

Ich sehe einen Weg, wie ich Forth wieder in den Hörsaal und ins Praktikum bringen kann. Ich lehre Künstliche Intelligenz und Robotik für den BSc-Studiengang (Bachelor of Science). In diesem Umfeld ist der Druck, sich an C/C++ zu halten, nicht ganz so groß. Wenn ich also eine Expert-System-Shell oder ein Toolkit, das in Forth geschrieben ist, hätte, könnte ich das verwenden. Ich habe das Buch "Real Time Expert Systems" von Townsend and Feucht. Weiß irgendjemand, ob diese Arbeit von irgendwem weiterentwickelt wurde? Die Software sollte kostenlos oder nicht zu teuer sein und unter Windows XP oder auf einem 68HC12 laufen.

Best wishes,

Graeme <g.r.a.dunbar@rgu.ac.uk>

Übersetzt von Fred Behringer. Graeme Dunbar lehrt an der Robert Gordon University, School of Engineering, in Aberdeen. Graeme kümmert sich seit Jahren um den Buchverleih der FIG UK (den ich häufig in Anspruch genommen habe) und hat kürzlich (zeitweilig?) auch noch die Redaktion der Forthwrite übernommen, da Chris Jakeman mit der "Schönen neuen Welt" zu kämpfen hat, die fleißig dabei ist, fähige Menschen aus Fleisch und Blut durch Roboter zu ersetzen. Über Graeme (auch über Graeme) hat die Forth-Gesellschaft seit Jahren guten Kontakt zur Forth Interest Group in Großbritannien. - Fred

Hi Graeme,

Du hast in der Vierten Dimension nach verschiedenen Dingen fragen lassen. Darunter auch nach billigen Robotern für Schüler und Studenten.
Ich habe vor einiger Zeit einen preiswerten gefunden und bestellt, habe aber bis jetzt leider nicht die Zeit gefunden, ihn wirklich zu bauen. Der Zusammenbau sollte nicht länger als einen halben Tag dauern - vielleicht nur ein paar Stunden. Ich habe das dazugehörige Buch gelesen und denke, er wäre eine mögliche Lösung für Dich.
Das Ganze heißt: Stiquito - Advanced experiments with a simple and inexpensive ROBOT by James M. Conrad and Jonathen W. Mills. Edited by IEEE Computer socieity Los Alamos California ISBN 0-8186-7408-3

Das Buch enthält den Bausatz für den Roboter. Es gibt eine dazugehörige Website:

http://www.stiquito.com/

Ich hoffe, Dir geholfen zu haben.

Gruß Martin

Selbstverständlich hat Martin Bitter diese Mail an Graeme in dessen Muttersprache geschrieben. *fep*

Liebe und Tensoralgebra

Stanislaw Lem

(...) Klapauzius dachte angestrengt nach. Schließlich nickte er und sagte: "Gut. Es soll von Liebe und Tod handeln, aber alles muß in der Sprache der höheren Mathematik ausgedrückt sein. Im wesentlichen Tensor-Algebra, vielleicht ein wenig höhere Topologie und Analysis. Dabei erotisch stark, sogar etwas gewagt, und natürlich ganz im Geiste der Kybernetik."

"Hast du den Verstand verloren? Liebe und Tensor-Algebra? - Ja weißt du denn noch, was du sagst?" schrie Trurl, verstummte jedoch gleich wieder, denn der Elektrobarde begann zu deklamieren:

Komm, laß uns tanzen in den Banach-Raum,
Wo Punktepaare wohlgeordnet sind,
Und Riemannsche Blätter rascheln im Wind,
Gefaltet, geheftet, schön wie im Traum.

Ich pfeife auf Bernoullis Fixpunktsatz,
Was soll'n mir Hilbert, Euler oder Venn
Mit ihren Indizes von eins bis n,
Wenn du mich liebst, mein rationaler Schatz !

Fixpunkte trämen von Kontraktionen,
Vektor schmeichelt der schönen Matrize,
Spalten bringt er in siedende Hitze,
Heiß und ergodisch glühen die Zonen.

Mordells Vermutung ist kein leerer Wahn,
Denn deine Kurven sind mein höchstes Ziel,
Ich zählte süßer Punkte endlich viel,
Und meine Graphen kreuzten ihre Bahn.

Du bist mein maximales Ideal,
Der Zustand meiner Liebe ist stabil,
Doch deine Kovarianten sind labil
Und unbestimmt wie Eulers Integral.

In deinen Augen glänzt der Eigenwert,
In jedem Seufzer schwingt ein Tensor mit,
Du weißt nicht wie mein Operator litt,
Hast du ihm doch Funktionen stets verwehrt.

Den Ring aus Polynomen gab ich dir,
Dazu die Markov-Kette mit dem Stein,
All deine Tensorfelder waren mein,
Nur dein Quotientenkörper fehlte mir.

Lösch mich nicht, denn was wird von mir bleiben ?
Parabeln, deren Brennpunkt niemand weiß,
Abszissen, zwei Mantissen und ein Kreis.
Laserstrahl wird mich zu Staub zerreiben.

Erstarren meine positiven Glieder,
Näht man mein topologisch Leichenhemd,
Vergiß mich nicht, werd mir nicht teilerfremd
Und sing am Grab mir lineare Lieder !

"Liebe und Tensoralgebra" ist eins von vielen Gedichten aus der Kurzgeschichte "Die Reise Eins A oder Trurls Elektrobarde" (Übersetzer : Jens Reuter) aus dem Buch "Kyberiade" von Stanislaw Lem, Insel Verlag, 1983

Manueller Computervirus

Bitte das Couvert keinesfalls öffnen, unter absolut keinen Umständen den Brief entnehmen, niemals ausdrucken. Versehentlich geöffneten Brief SOFORTh vernichten (Feuer, Säure oder Reißwolf).

> Hallo,
>
> dies ist ein manueller Mail Virus.
> Sein Entwickler hat leider keine Ahnung und
> keine Zeit um einen Echten zu programmieren.
>
> Wählen Sie einfach die ersten 50 Adressen aus
> Ihrem Adressbuch und senden Sie diesen Virus
> weiter.
> Dann löschen Sie einige Dateien aus Ihrem
> Systemverzeichnis.
>
> Falls heute Freitag der 13. ist, formatieren Sie bitte
> Ihre Festplatte.
>
> Danke für Ihre Mitarbeit!
>
>
>
>

Kennen Sie WebAdressen mit spaßigen Inhalten (nein, NICHT Microsoft.COM), an denen wir allesamt Spaß haben könnten? Bitte schreiben oder mailen Sie uns – aber nicht SO!

fep

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

Jens Wilke
Tel.: (0 89) - 8 97 68 90
Treffen: jeden 4. Mittwoch im Monat
Ristorante Pizzeria Gran Sasso
Ebenauer Str. 1
80637 München

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)

Gruppengründungen, Kontakte

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

µP-Controller Verleih

Thomas Prinz
Tel.: (**0 62 71) - 28 30** (p)
micro@forth-ev.de

Forth-Hilfe für Ratsuchende

Jörg Plewe
Tel.: (**02 08) - 49 70 68** (p)

Jörg Staben
Tel.: (**0 21 03) - 24 06 09** (p)

Karl Schroer
Tel.: (**0 28 45) - 2 89 51** (p)

Spezielle Fachgebiete

Arbeitsgruppe MARC4
Rafael Deliano
Tel./Fax: (**0 89) - 8 41 83 17** (p)

FORTHchips
(FRP 1600, RTX, Novix)
Klaus Schleisiek-Kern
Tel.: (**0 40) - 37 50 08 03** (g)

F-PC & TCOM, Asyst (Meßtechnik), embedded Controller (H8/5xx// TDS2020, TDS9092), Fuzzy
Arndt Klingelnberg, Consultants
akg@aachen.kbbs.org
Tel.: **(00 32)(0 87) - 63 09 89** (pgQ)
Fax: **- 63 09 88**

KI, Object Oriented Forth, Sicherheitskritische Systeme
Ulrich Hoffmann
Tel.: (**0 43 51) - 71 22 17** (p)
Fax: **- 71 22 16**

Forth-Vertrieb
vlksFORTH
ultraFORTH
RTX / FG / Super8
KK-FORTH
Ingenieurbüro
Klaus Kohl
Tel.: (**0 82 33) - 3 05 24** (p)
Fax :(**0 82 33) - 99 71**
mailorder@forth-ev.de

Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen hinausgehen?

Schreiben Sie einfach der VD - oder rufen Sie an - oder schicken Sie uns eine E-Mail!

Hinweise zu den Angaben nach den Telefonnummern:

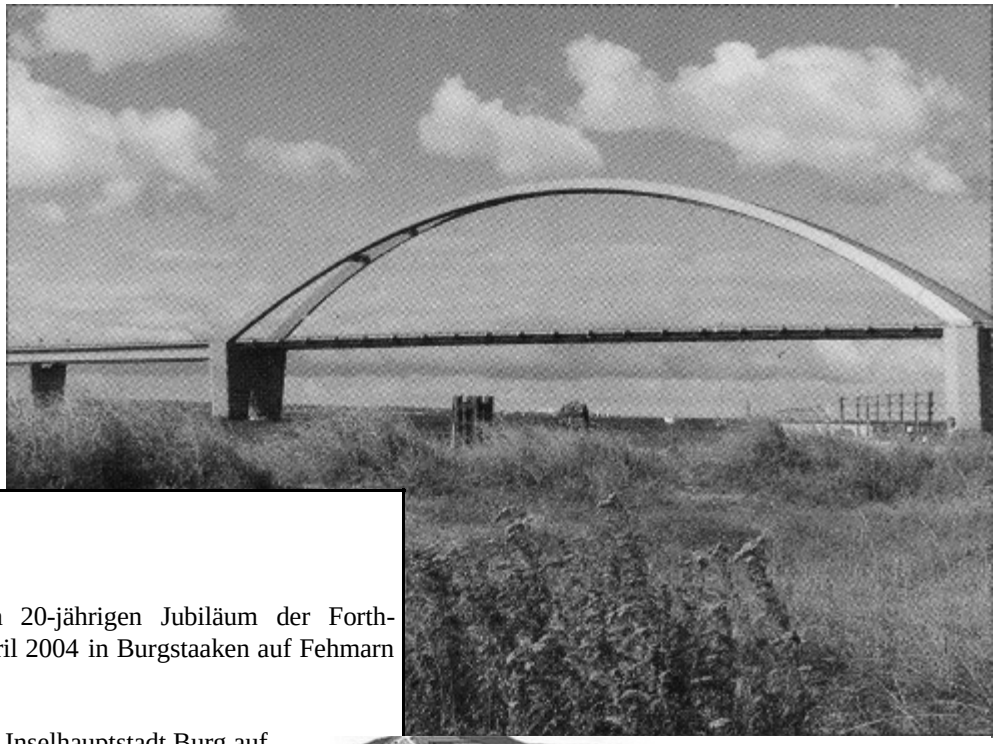
Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeitszeiten
g = geschäftlich

Die Adressen des Büros der Forthgesellschaft und der VD finden Sie im Impressum des Heftes.

4 / 2003

20 Jahre Forthgesellschaft

Auf Wiedersehen auf Fehmarn



Reif für die Insel?

Die Jahrestagung, 20-FG-04, zum 20-jährigen Jubiläum der Forth-Gesellschaft, findet vom 16.-18. April 2004 in Burgstaaken auf Fehmarn statt.

Burgstaaken ist der Fischerhafen der Inselhauptstadt Burg auf Fehmarn und bietet eine reizvolle, maritime Atmosphäre.

Wir werden im Hotel Schützenhof untergebracht sein, das uns Platz für gut 50 Tagungsteilnehmer bietet.

Fehmarn, die einzige Schleswig-Holsteinische Ostsee-Insel, ist über die Fehmarn-Sund-Brücke mit dem Festland verbunden und kann daher bequem mit dem Auto erreicht werden. Zudem gibt es mit dem Fährhafen und Bahnhof Puttgarden die Möglichkeit, mit dem Zug direkt bis auf die Insel zu reisen. Fehmarn lädt zu ausgedehnten Strandspaziergängen und auch zum Bummeln im traditionellen Innenstadtbereich von Burg ein.

Die formale Einladung zur Tagung und zur Mitgliederversammlung werden wir in der kommenden 1/2004 Ausgabe der Vierten Dimension abdrucken, der dann auch der Anmeldebogen beiliegt. Aktuelle Entwicklungen gibt's wie immer unter **www.forth-ev.de**.

Ulrich Hoffmann