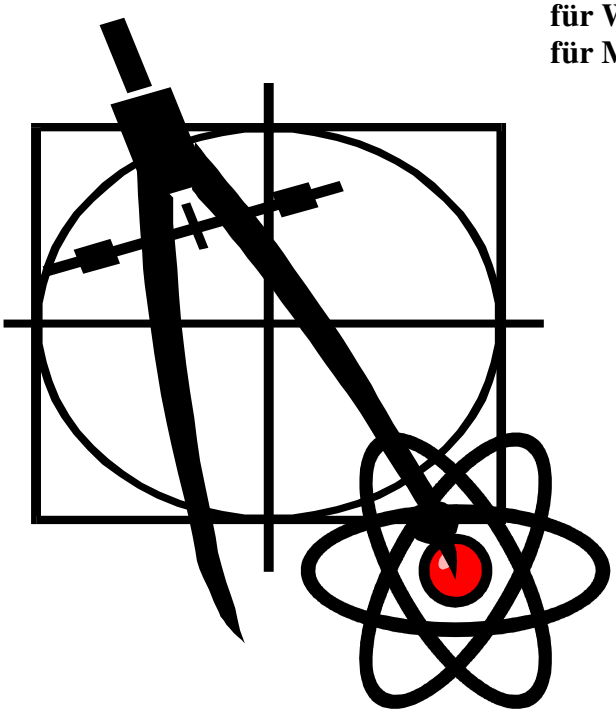
VIERTE DIMENSION

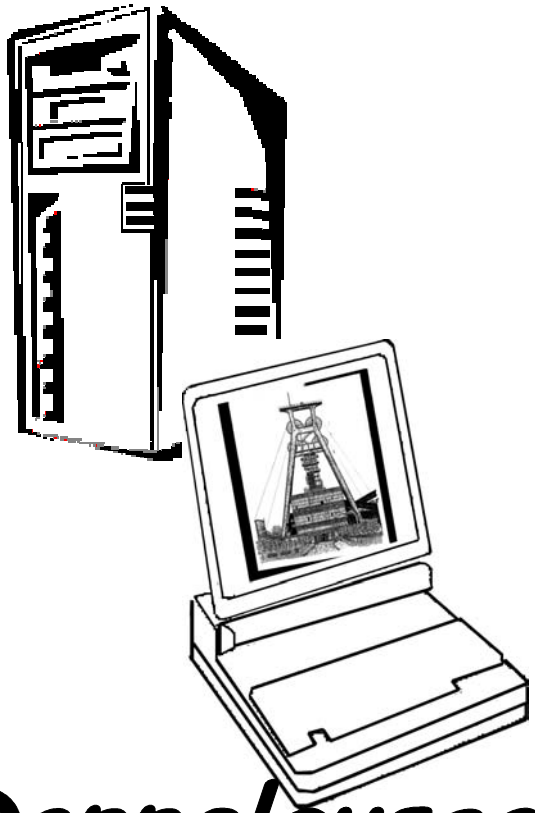
3;4 / 2005

Das FORTH-Magazin

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



In dieser Ausgabe:



Doppelausgabe 3/4 - 2005

21. Jahrgang - **Ausg. 3;4/2005** - 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'!

RetroForth

Linux · Windows · Native

Generic · L4Ka::Pistachio · Dex4u

Public Domain

http://www.retroforth.org
http://retro.tunes.org

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

Hier könnte Ihre Anzeige stehen!

Wenn Sie ein Förderer der Forthgesellschaft sind oder werden möchten, sprechen Sie mit dem Forth-Büro über die Konditionen einer festen Anzeige.

Secratary@forth-ev.de

KIMA Echtzeitsysteme GmbH

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

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

FORTecH Software
Entwicklungsbüro Dr.-Ing. Egmont Woitzel

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

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

Ingenieurbüro
Dipl.-Ing. Wolfgang Allinger

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

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

Ingenieurbüro
Klaus Kohl

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

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

Inhalt

In der nächsten Ausgabe finden Sie voraussichtlich:

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
Haspelstrasse 6 d
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

Manuskripte und Rechte

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

Liebe Leser,

schon zur letzten Ausgabe der VD mußte ich Sie wegen der deutlichen Verspätung der Fertigstellung um Entschuldigung bitten. Dieses Mal halten Sie Ihre Zeitschrift mit noch größerem zeitlichen Verzug in den Händen. Und ich kann Ihnen keine Besserung geloben. Deshalb habe ich die Direktoren der Forthgesellschaft gebeten, mich von meinen Aufgaben rund um die Vierte Dimension zu befreien.

Zu solchen Gelegenheiten werden gerne neben tatsächlichen, vermeintlichen und zu kommunizierenden Gründen diverse Statistiken veröffentlicht. Da läßt sich dann lesen, wie lange und wie oft jemand eine bestimmte Aufgabe übernommen und zugehörige Arbeiten geleistet hat. Ein Rückblick zeigt besondere Leistungen und Schlaglichter aus der Sicht des Laudators auf.

Die Statistiken mag sich der geneigte Statistiker selbst zusammenstellen. Besondere Leistungen meinerseits, im Sinne von unbedingt zu erwähnenden Taten oder Unterlassungen gab es aus meiner Sicht nicht. Aber besondere Glanzlichter durfte ich jede Menge erleben. Jeder Artikel, jeder noch so kleiner Beitrag, selbst jeder Hinweis eines Lesers auf eine im Web gefundene Information war eine echte Freude. Diese Arbeiten, ob im Umfang groß oder klein, sind von den jeweiligen Autoren weit überwiegend neben vielfältigen familiären und beruflichen Pflichten angefertigt worden. Da hat uns, den Mitgliedern der Forthgesellschaft und den Lesern der Vierten Dimension, jemand etwas geschenkt. Ich bewerte jedes dieser Geschenke sehr hoch. Und ich danke allen Autoren an dieser Stelle herzlich für die Zusammenarbeit der vergangenen Jahre.

Zusammenarbeit meint in diesem Kontext die ständige Pflege unterschiedlich intensiver Kontakte zu den Autoren. Das gemeinsame Entwickeln von thematischen Vorstellungen zu einzelnen Ausgaben, Anfragen nach möglichen Beiträgen, terminliche Absprachen mit allen Beteiligten und das gesamte „Drumherum“ der Organisation und Koordination erfordern in der Summe einen zeitlichen Einsatz, den ich aktuell und auch in absehbarer Zeit nicht mehr zu leisten vermag.

Die Direktoren der Forthgesellschaft sind bereits bemüht, eine Lösung zu entwickeln, die Ihnen wieder eine pünktlich erscheinende Vierte Dimension garantiert. Sie, liebe Leser, können einen ganz wesentlichen Beitrag dazu leisten. Arbeiten Sie an Ihrer Zeitschrift mit, gerade dann, wenn Sie bisher noch nicht zum Autorenstamm gehören.

In diesem Sinne danke ich allen Autoren und Lesern nochmals für die vergangenen Jahre, in denen ich „rund um die VD“ arbeiten durfte.

Ich verbleibe mit
herzlichem Glückauf,
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	Kontakte: Direktorium@Forth-ev.de
Dr.	Ulrich Hoffmann	
Dipl. Inf.	Bernd Paysan	

Forthtagung 2006

Ankündigung und Call for Papers:

Die nächste Jahrestagung der Forthgesellschaft wird in der Nähe von Witten stattfinden. Witten, am Südrand des Ruhrgebietes zwischen Bochum und Dortmund gelegen, bewirbt sich wie die ganze Region um den Titel der Kulturhauptstadt 2010. Menschen einander näher zu bringen durch Kultur, mit diesem Ziel präsentieren sich jährlich unterschiedliche europäische Städte als internationale Kulturzentren und völkerverbindende Begegnungsorte. Im Jahr 2010 will das Ruhrgebiet, mit seinen 53 Städten und Gemeinden, zum kulturellen Austausch einladen. Ein kultureller Ballungsraum, der in seiner Dichte einzigartig in Deutschland und Europa ist. Aus dem einstigen Industrierevier ist inzwischen eine vielseitige Region entstanden. Wir als Forthgesellschaft sind schon vielseitig von Anfang an. Die Art dieser Programmiersprache, ihre Gestaltbarkeit, bringt das mit sich.

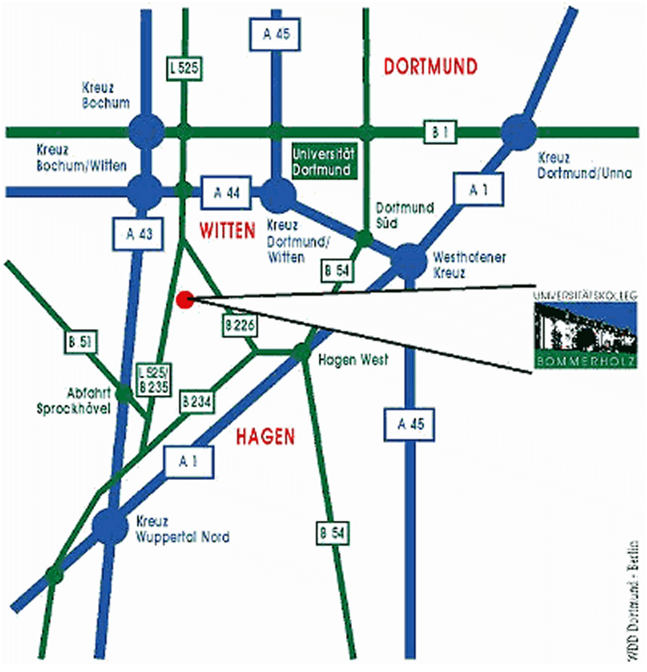
Forthtagung in Witten

Fr. 12. – So. 14. Mai 2006

Bei der Suche nach einem Tagungsort hier in der Gegend fiel die Wahl auf das Kolleg der Universität Dortmund, weil es so gut ausgestattet ist, und mit all den Annehmlichkeiten aufwarten kann, die wir auf unseren Forthtagungen inzwischen erwarten. Gepflegte Unterbringung, gute Küche und Gastlichkeit in einem Haus für uns, ungestört, auch bis tief in die Nacht nicht ohne Getränke. Im voll ausgestatteten Tagungsraum, mit WLAN und DSL ins Internet, kann alles, was die Teilnehmer erfahrungsgemäß so mitbringen, betrieben werden. Beamer, Tafeln, Ausstellung, kleinere Räume für Gruppen, eben alles da. Dabei liegt das Haus am Wald, hat eine Umgebung, die zum Spazieren einlädt, es gibt Museen, Shopping, Wellness, Theater. Aber auch moderne Fabriken wie Opel oder das Edelstahlwerk sind hier ansässig. Nur die rauchenden Schlote der Zechen sind verschwunden; Hochöfen, Kokereien, Krupp sind nicht mehr hier. Die Region ist seitdem im Umbruch und ist es noch.

Bommerholzer Str. 60
58456 Witten-Bommerholz

Die Forthtagung macht also nun auf ihrer Wanderschaft durch Deutschland wieder Station tiefer im Westen. Auch Forth hat hier Spuren hinterlassen in etlichen Köpfen und Projekten. Wir wollen auf die Spurensuche gehen und auch diesen Teil der **Forth-Geschichte** beleuchten, Forth – vom Kult zur Kultur. Alle, die dazu beitragen möchten, sind herzlich dazu eingeladen. Dann aber soll es weiter gehen in die Bereiche **Datenschutz und Privacy** und ich hoffe, es finden sich dazu Beiträge auch aus der Perspektive des Forth. Dieses Thema hat einzelne in der FG in diesem Jahr sehr beschäftigt und beschäftigt uns ja alle außerordentlich im Lande. Und natürlich wird **die Kunst Forth zu implementieren** gepflegt, auf neuen Wirten und als eigens gefertigten Chips. Ich hoffe, es gibt eine Ausstellung dazu. Sodann gebührt den **Applikationen in Forth** unsere Aufmerksamkeit und - last not least - die Frage, wie wir als Gesellschaft Forth hier bekannt machen und bereit stellen können, und wie Forth derzeit international aufgenommen wird. Weitere Vorschläge sind willkommen.



So rufe ich Euch auf, Beiträge nun einzureichen und sich frühzeitig anzumelden.

Euer *M.Kalus.*

Das Forthbüro meldet uns ein neues Mitglied im Verein. Ein herzlicher Willkommensgruß geht an

Armin Herold.

Rolf Schöne

Betreff: forth / frohnhof
Von: Kalus Michael

Hier was für die VD.

Das Ende von Frohnhof (=C++ und??) und aufwärts gehts mit Forth. Ist das nicht schön! Da ist dem Daniel Ciesinger ein wahrer Schnappschuss geglückt :-)

Wo ist das gewesen? Ein paar Kilometer nördlich von Nürnberg liegt an der B2 der Ort Eckental mit den Ortsteilen Brand, Eckenhain, Forth, Büg und Ebach. Die nördlich angrenzende Gemeinde ist Fronhof. Die B2 verläuft von Eckental über Forth nach Frohnhof und heißt dort Forther Hauptstraße. Nachzulesen unter {http://www.stadtplandienst.de/}, Ortsteilsuche.

Viele Grüße, *Michael*

Die Russische-Bauern-Multiplikation für 32/32/64 Bit

in 16-Bit-Turbo-Forth-Assembler

Fred Behringer
<behringe@ma.tum.de>

Ziel der Arbeit: Nix "russisch"

Martin Bitter hat uns diese "russische" Methode des fortgesetzten Halbierens und Verdoppelns, bei dem ansonsten nur Additionen vorkommen, im VD-Heft 1/2001 erklärt [Bi1].

In der Fachliteratur ist man voller Bewunderung für dieses "Verfahren", das auch "ägyptische Multiplikation" oder "Fellachenmultiplikation" genannt wird (z.B.[Mit],[Röc], [Zie]): "Die Komplexitätsanalyse", sagt [Zie] auf Seite 181, "beweist die Schläue der russischen Bauern". Nun ja, ich möchte hier darauf aufmerksam machen, dass dieses "schlaue Verfahren" nichts weiter ist als das übliche Multiplizieren, das wir alle im Schulunterricht gelernt haben. (Ob wir die zu multiplizierenden (natürlichen) Zahlen dabei in Dezimaldarstellung, in Hexadezimaldarstellung oder eben in Binärdarstellung betrachten, ist ja schließlich von untergeordneter Bedeutung.) Ich möchte behaupten, dass die Funktionsweise (der russischen Multiplikation) damit hinreichend erklärt ist (Verifikation des Algorithmus) und dass wir uns nichts zu merken und dass wir nichts Neues hinzuzulernen brauchen.

Als Nebenprodukt: 32/32/64-Bit-Multiplikation für Turbo-Forth

Um eine "schnelle Hardware-Multiplikation" (vergleiche die Zielsetzung in [Bi1]) geht es mir hier nicht, wohl aber um eine Erweiterungsmöglichkeit für (die 16-Bit-Ausführung von) Turbo-Forth. (Was für Turbo-Forth gilt, gilt weitestgehend auch für ZF und natürlich auch für den gemeinsamen Vorfahren F83.) Für die (überlaufverlustfreie) Multiplikation mit doppeltgenauem Ergebnis (ich beschränke mich auf vorzeichenlose Ganzzahlen) gibt es in Turbo-Forth das Wort UM* (u1 u2 – du). Hier sind u1 und u2 einfachgenau, du doppeltgenau. Eine Multiplikation doppeltgenauer Zahlen, ob mit Vorzeichen oder ohne, ob mit vierfachgenauem Ergebnis oder nur doppeltgenauem, gibt es (in Turbo-Forth) nicht. Im Verlaufe der Diskussionen bringe ich weiter unten eine CODE-Definition UD*Q (ud1 ud2 -- qu), einmal "russisch" (nur Additionen), einmal mit direkten Intel-Mitteln (MUL). Hierbei sind ud1 und ud2 doppeltgenau (32 Bit) und qu vierfachgenau (64 Bit). Für Assembler-Ungewohnte gebe ich dann noch ein High-Level-Forth-Programm an, dass aber einige 32-Bit-Vorbereitungen benötigt und daher auch nicht viel "einfacher" ist – und wesentlich mehr Zeit erfordert.

Multiplikation mit Bleistift und Papier ...

Das Schulmultiplizieren, das ja für jedes Stellenwertsystem (mit fester Basis!) funktioniert, auf die Basis 2 angewandt. Möglichst den längeren (Ganzzahl-)Faktor als Multiplikand nehmen, den kürzeren als Multiplikator. Letzterer bestimmt die Anzahl der Additionen (siehe unten). Den Multiplikator (wie bei Martins Binärbeispiel) von rechts nach links abarbeiten (nicht, wie bei Martins Dezimalbeispiel, von links nach rechts). Ausnützen, dass Bitverschiebung nach rechts bei geraden Zahlen (nur bei diesen!) einer Division durch 2 gleichkommt, bei ungeraden Zahlen der Subtraktion einer 1 (mit Aufbewahrung im CF-Flag) und anschließender Division durch 2. Beim Aufaddieren nicht warten, bis alle Teilergebnisse vorliegen (unmögliche Anzahl von benötigten Registern), sondern Addition in jedem einzelnen Schleifenschritt "sofort" vornehmen.

... und die Russische-Bauern-Regel ...

Ich darf sie wie folgt formulieren:

(0) Setze c := 0 und initialisiere a und b.
(1) Ist b gerade, so verdopple a und halbiere b.
(2) Ist b ungerade, so addiere a zu c, verdopple a, ziehe 1 von b ab und halbiere dann b.
(3) Ist b = 0, dann gehe zu (5).
(4) Ist b <> 0, dann gehe zu (1).
(5) In c liegt das Produkt. Ende.

... ist nichts weiter als das Schulmultiplizieren

Ich nehme die Zahlen aus Martin Bitters Beispielen [Bi1], a = 100111001 = 313 für den Multiplikanden, b = 101101 = 45 für den Multiplikator, binär und dezimal ausgedrückt, und c = 0 für das (doppeltlange) Produkt. Für die einzelnen Schleifendurchgänge im Beispiel 313 * 45 ergibt sich:

(i)	a =	100111001	b =	101101
	c =	100111001		
(ii)	a =	1001110010	b =	10110
	c =			
(iii)	a =	10011100100	b =	1011
	c =	11000011101		
(iv)	a =	100111001000	b =	101
	c =	111111100101		
(v)	a =	1001110010000	b =	10
	c =			
(vi)	a =	10011100100000	b =	1
	c =	11011100000101		
(vii)	a =	100111001000000	b =	0
	c =			

Dass das genau die übliche Multiplikation ist (so, wie ich sie im Schulunterricht gelernt habe) - auf Binärzahlen angewandt - , ist klar. Mit "Bleistift und Papier" würde man die zu geraden b-Werten gehörigen a-Werte weglassen (gar nicht erst hinschrei-

ben) und die verbleibenden, jeweils um eine Stelle nach links verschobenen a-Werte ganz am Ende der Schleifendurchgänge "in einem Rutsch" addieren. Das ist dann aber eine reine Frage der Notierung. Wir können also das "Besondere" an dieser "Bauernregel" vergessen. Sie IST das schulübliche Multiplikationsverfahren – auf Binärzahlen angewandt.

Funktioniert immer: Distributivität und Kommutativität

Und dass das schulübliche Multiplizierverfahren funktioniert (stets richtige Ergebnisse liefert und terminiert), liegt ganz einfach in der Distributivität von Multiplikation und Addition begründet:

a*b = (a1+a2) * (b1+b2) = a1*b2 + a2*b2 + a1*b1 + a2*b1.

Und die Kommutativität von Multiplikation und Addition sorgt dafür, dass das Ausmultiplizieren der Klammern in beliebiger Reihenfolge erledigt werden kann:

a*b = (a1+a2) * (b1+b2) = a1*b1 + a2*b1 + a1*b2 + a2*b2.

Und dass die gleich folgende Umsetzung der Multiplikation zweier 32-Bit-Faktoren zu einem 64-Bit-Produkt im 16-Bit-Turbo-Forth-Assembler genau dem üblichen Programmierschema bei der Software-Emulation der Multiplikation (für Prozessoren ohne MUL) entspricht, ist genauso klar.

Auch für beliebige Zahlendarstellungen

Es ist klar, dass das Verfahren (falls b ungerade, 1 von b abziehen und a zu c addieren; sodann b halbieren und a verdoppeln) von dem speziellen Zahlensystem, in welchem a dargestellt wird, unabhängig ist. Auch Nichtstellenwertsysteme kommen in Frage, haben aber natürlich in der Praxis nur dann einen Sinn, wenn die Addition hinreichend leicht zu bewältigen ist. Was wir brauchen, ist lediglich die Möglichkeit, beim jeweiligen b zu entscheiden, ob es gerade ist oder ungerade. Das Hinschreiben oder Weglassen des immer wieder mit 2 malgenommenen Multiplikanden a (bei "Bleistift und Papier") läuft auf eine implizite Binärentwicklung von b und ein (in der so entstehenden Binärdarstellung von b) stellenweises Multiplizieren des jeweiligen a mit 2 und eine Addition zum bisherigen Teilergebnis, wenn das betreffende b-Bit eine 1 ist, hinaus. Kein Grund also zur Mystifizierung.

Softwaremäßige Maschinendarstellung überflüssig?

Außerdem ist klar, dass das gleich folgende Turbo-Forth-Assembler-Programm beim 80486 und höher keinen Selbstzweck verfolgt. Man braucht dort ja nur die eingebauten Multiplikationsbefehle IMUL (mit Vorzeichen) oder MUL (ohne Vorzeichen) zu verwenden – und steht in den Ausführungszeiten günstiger da (siehe weiter unten). Ich möchte nur zeigen, dass das Russische-Bauern-Verfahren kein Geheimnis enthält. Man könnte den Fellachen oder den russischen Bauern sagen, dass das, was sie tun, auf eine Binärentwicklung ihrer (natürlichen) Zahlen und eine anschließende Multiplikation nach dem auch bei uns ausgeübten Schulverfahren hinausläuft. Die allbekannte Arithmetik wird halt bei Binärzahlen einfach einfacher. Das Verfahren verdient keinen eigenen Namen.

CPUs ohne MUL

Andererseits gab es beispielsweise beim 6502 - noch gar nicht so lange her - gar keine eingebaute Multiplikation. Die Billigst-Controller von heute haben auch keine. Und für die FPGA-Projekte ([Sc1],[Pay]) mit minimalistischem Ansatz ([All]) wird man die Multiplikation aus Ersparnisgründen (zunächst einmal) wohl auch in Software ausführen.

Und geht man auf einem PC (nach der anderen Seite des Betrachtungsspektrums hin) zur Multiplikation übergroßer Langzahlen (64/64/128 Bit) über, dann ist wieder zu überlegen, ob nicht eine Entsprechung des gleich folgenden Programms (Aufgabe!) auch auf dem 80486 und höher dann wieder interessant wird: Dann müsste ja die "direkte" Gesamtmultiplikation in vier Einzelmultiplikationspakete aufgespaltet werden. Natürlich gibt es Methoden, wie die von Karatsuba ([Bi2],[Arn],[Sc2]), die mit drei Teilpaketen auskommt, oder die schnelle Fourier-Transformation ([Arn],[Sc2]), aber dann müsste man erst einmal den zusätzlichen Verwaltungsaufwand, auch wenn er sich bei Karatsuba "nur" in Additionen ausdrückt, genauer unter die Lupe nehmen (interessante Aufgabe für Turbo-Forth).

Trick zur Reduzierung der Ausführzeiten

Ich möchte die Ausführungszeiten so kurz wie möglich halten und verschiebe z.B. a um ein Bit nach links (Multiplikation mit 2) dadurch, dass ich a zu sich selbst addiere. Die Addition zweier 32-Bit-Ganzzahlen benötigt (ab dem 80486) nur einen einzigen CPU-Takt. (Ich orientiere mich in Bezug auf CPU-Takte an Podschun [Pod].)

Direkte Erweiterung von Turbo-Forth

Allerdings wird man sich bei Turbo-Forth, ZF und den anderen F83-Derivaten schnell wundern, warum es keine Möglichkeit gibt, zwei 32-Bit-Ganzzahlen (mit oder ohne Vorzeichen) zu einer 64-Bit-Ganzzahl zu multiplizieren. Der Prozessor macht es ja schon seit langem möglich. Ich mache mir weiter unten über einen Zeitvergleich Gedanken und gebe auch kurz das direkte Programm (über MUL) zur 32/32/64-Multiplikation (als CODE-Definition) an. Diese CODE-Definition kann dann als Punktzahl-Erweiterung von Turbo-Forth betrachtet werden.

Feinheiten: Für 16-Bit-Assembler nur Präfix OP: = 66 C, nötig

Ich verwende nur den Ganzzahlbearbeitungsteil des Prozessors 80x86 ab x = 4 (die CPU) und überlasse es dem geneigten Leser, Möglichkeiten des Einsatzes des Gleitkommateils (der Ganzzahlbefehle der FPU) zu untersuchen. Außerdem möchte ich nur vorzeichenlose (also nichtnegativ zu interpretierende) Ganzzahlen betrachten. Ich benötige zwei 64-Bit- und ein 32-Bit-Register. Ich setze:

a(hi,lo) = EAX,EDI b = EBX c(hi,lo) = ECX,EDX

Der Multiplikand a wird bitweise nach links verschoben und kann (mit angehängten Nullen) bis zu einer Größe von 64 Bit anwachsen. Das jeweilige Hinzuaddieren zu c geschieht in 64-Bit-Breite. Das Produkt c muss also ebenfalls den 64-Bit-Rahmen ausschöpfen können. Der Multiplikator b, bei dem nur immer das rechts herausfallende Bit (beim hier verwendeten Shift-Befehl SHR das Carry-Flag CF) wirksam wird, wird mit jedem Schleifenschritt kleiner und kommt also mit 32 Bit aus.

Aus dem 16-Bit-Register AX mache ich im folgenden Programm durch Voransetzen des zu diesem Zweck konstruierten Wortes "OP:" das 32-Bit-Register EAX usw. Ich kann also mit dem Turbo-Forth-Assembler, der nur 16-Bit-Befehle verarbeitet, 32 Bit breit operieren, ohne den eigentlichen Assembler "erweitern" zu müssen.

An nur zwei Stellen, beide Male 10 # CL MOV, dem Vorbelegen des Shiftanzahlregisters CL mit 10h, habe ich auf "OP:" verzichten und mit "nur" 16 Bit auskommen können. Der "Schönheit" wegen hätte ich aber auch da "OP: 10 # CX MOV" schreiben können.

Das Bleistift-und-Papier-Programm

Achtung:

Die Eingabewerte sind Punktzahlen (doppeltgenaue Ganzzahlen)! Eine Punktzahl d wird in Turbo-Forth und anderen F83-Derivaten auf dem Stack und in den 2Variablen als dhi dlo abgelegt. (Ich spreche in der Sprache der "aufsteigenden" Speicherstellen – bei Variablen UND Stack. Man beachte aber, dass der Stack nach unten wächst, dass also dhi vom Stack herunter als erstes abgeholt wird.) Ist AA eine 2Variable (32 Bit), so liefert also AA @ den höheren 16-Bit-Anteil. (Man beachte, dass bei einer einfachen (16 Bit) Variablen BB die Anweisung BB C@ den lo-Anteil (des Doppelbytes) liefert.) Für vierfach genaue Zahlen übernehme ich diese verdrehte Organisation, also: q = qhi qlo = qhihi qhilo qlohi qlolo – von niedrigen nach höheren Speicherstellen hin gesehen. (Konsistenz ist auch hier sicher ein vertretbares Argument.) Ist CC eine vierfach genaue Variable (64 Bit), so liefert CC 2@ den höheren 32-Bit-Anteil, CC @ von diesem wiederum den höheren 16-Bit-Anteil (qhihi).

Eine doppeltgenaue Zahl (12345678.) kann auch ohne Punkt, in zwei 16-Bit-Häppchen, eingegeben werden: 5678 1234 . Ich möchte den geneigten Leser bitten, diese Umstellung, die ich weiter unten bei 64-Bit-Zahlen (im Zusammenhang mit 4@ und 4!) analog verwenden möchte, zu beachten. Ich weiß mir nicht anders zu helfen. Die Einführung einer Hintereinanderwegschreibung von 64-Bit-Zahlen ("Zweifach-Punkt-Zahlen": 1234567876543219..) hätte einen Riesenaufwand bedeutet, der nicht gerechtfertigt gewesen wäre. Was will ich schließlich zeigen? Ich will zeigen, dass es unsinnig ist, UD*Q in High-Level-Forth aus schon bestehenden Turbo-Forth-Worten zusammenzustückeln.

```
\ RUSS-MUL.FTH

HEX

ONLY FORTH ALSO
ASSEMBLER DEFINITIONS

: OP: ( -- ) 66 C, ;

FORTH DEFINITIONS

CODE UD*Q ( ud1 ud2 -- uqlo uqhi )  \ d1 = b, d2 = a
   OP: AX AX XOR      \ zunaechst: a(hi) = EAX = 0
   OP: DI POP         \ zunaechst: a(lo) = EDI = a
       10 # CL MOV    \ Intel-Format: CL := 10h, dann
   OP: DI CL ROL      \ EDI rotieren.  Nun EDI = a
   OP: BX POP         \ b noch "verdreht".
   OP: BX CL ROL      \ EBX rotieren.  Nun EBX = b
   OP: CX CX XOR      \ zunaechst: c(hi) = ECX = 0
   OP: DX DX XOR      \ zunaechst: c(lo) = EDX = 0
   \ ------------------
   BEGIN
     OP: BX SHR       \ b ungerade: CF=1; b gerade: CF=0
     U<               \ CF=0:              b :=  b/2
     IF               \ CF=1: b := b-1 und dann b :=  b/2
       OP: DI DX ADD  \ a(lo) :=     a(lo) + c(lo)
       OP: AX CX ADC  \ a(hi) := CF + a(hi) + c(hi)
     THEN             \ Weiter: a verdoppeln. D.h.,
     OP: DI DI ADD    \ a(lo) :=     a(lo) + a(lo)
     OP: AX AX ADC    \ a(hi) := CF + a(hi) + a(hi)
     OP: BX BX OR  0=
   UNTIL
   \ ------------------
   OP: CX AX MOV      \ CL wird gebraucht: ECX nach EAX
       10 # CL MOV    \ CL = 10h vorgeben
   OP: DX CL ROL      \ EDX (Produkt low ) rotieren
   OP: DX PUSH        \ und auf Stack legen.
   OP: AX CL ROL      \ EAX (Produkt high) rotieren
   OP: AX PUSH        \ und auf Stack legen.
   NEXT END-CODE
   \ Die folgende Eingabe gibt jetzt das Produkt a*b auf dem
   \ Bildschirm in 64-Bit-Breite in der "richtigen" Reihenfolge
   \ aus:
   \ a ( als Punktzahl ) b ( als Punktzahl ) UD*Q D. D. [ret]
```


Ein direktes Programm für UD*Q

Könnte wie folgt aussehen. Achtung: Die Eingabewerte sind Punktzahlen (doppeltgenaue Ganzzahlen)!

```
HEX

ONLY FORTH ALSO
ASSEMBLER DEFINITIONS

\ : OP: ( -- ) 66 C, ; \ Schon definiert (siehe oben)

FORTH DEFINITIONS

CODE UD*Q^ ( ud1 ud2 -- uqlo uqhi ) \ ud1 = b, ud2 = a
       10 # CL MOV    \ CL = 10h fuer Verschiebung vorgeben.
    OP: AX POP        \ EAX (Multiplikand)
    OP: AX CL ROL     \ rotieren (little endian).
    OP: DX POP        \ EDX (Multiplikator)
    OP: DX CL ROL     \ rotieren (little endian).
    OP: DX MUL        \ EDX * EAX = EDX:EAX
    OP: AX CL ROL     \ EAX (Produkt low ) rotieren
    OP: AX PUSH       \ und auf Stack legen.
    OP: DX CL ROL     \ EDX (Produkt high) rotieren
    OP: DX PUSH       \ und auf Stack legen.
    NEXT END-CODE
```

High-Level-Forth

Und hier noch ein Programm zum Überprüfen des Assembler-Programms zur russischen Multiplikation mit puren Forth-Mitteln. Turbo-Forth (oder ZF oder F83) müsste dann "eben noch schnell" um folgende doppeltgenauen und vierfachgenauen Operationen erweitert werden:

```
: 4VARIABLE VARIABLE 6 ALLOT ;
: 4@ DUP >R 4 + 2@ R> 2@ ;
: 4! DUP >R 2! R> 4 + 2! ;
\ : OP: ( -- ) 66 C, ; \ Schon definiert (siehe oben)
CODE Q+ ( q1lo q1hi q2lo q2hi -- q3lo q3hi )
  10 # CL MOV    \ Fuer Intel-Format (little endian)
  OP: AX POP  OP: AX CL ROL
  OP: BX POP  OP: BX CL ROL
  OP: DX POP  OP: DX CL ROL
  OP: DI POP    OP: DI CL ROL
  OP: DI BX ADD    OP: AX DX ADC
  OP: BX CL ROL    OP: BX PUSH
  OP: DX CL ROL    OP: DX PUSH  NEXT END-CODE
: Q2* 4DUP Q+            ;
: 2AND ROT AND -ROT AND OR ;
: DSHR D2/ 7FFF AND       ;
```

Bemerkung: Es ist mir leider nicht gelungen, in High-Level-Forth einen doppeltgenauen Überlaufschutz bei der Addition zweier vierfachgenauen Zahlen zu verwirklichen. Also konnte ich Q+ nicht aus schon bestehenden D+-Anteilen zusammensetzen.

Und hier das High-Level-Forth-Programm. Vorsicht, ich verwende A, B und C als Variablennamen. Will man einen dieser Buchstaben als (einzelne) Hex-Zahlen-Eingabe verwenden, so tut man gut daran, 0A, 0B und 0C zu schreiben.

```
4VARIABLE A
2VARIABLE B
4VARIABLE C

: UD*Q^^ ( ud1 ud2 -- uqlo uqhi ) \ ud1 = b, ud2 = a
  0. A 4!  B 2!  0. 0. C 4!
  BEGIN
    B 2@ D0= NOT
  WHILE
    B 2@ 1. 2AND IF C 4@ A 4@ Q+ C 4! THEN
    B 2@ DSHR B 2!
    A 4@  Q2* A 4!
  REPEAT
  C 4@ ;
```

Zeitabschätzungen

Um den High-Level-Verwaltungsüberbau so gut wie möglich auszuschalten, führe ich zwei Variablen ein, AA und BB. Ich belege beide je einmal mit "FFFFFFFF." (größtmögliche Anzahl von Schleifendurchgängen beim "russischen" Verfahren) und je einmal mit "12345678." ("mittlere" Verteilung von 0 und 1 beim Multiplikator des "russischen" Verfahrens).

Ich vergleiche erst die beiden CODE-Definitionen UD*Q und UD*Q^ miteinander. Dazu brauche ich eine Doppelschleife, um zu brauchbaren Vergleichszeiten (in Sekunden) zu gelangen. Sodann vergleiche ich die CODE-Definition UD*Q mit der Colon-Definition UD*Q^^. Dazu genügt die innere Schleife, da sonst die Colon-Definitions-Schleife unerträglich lange Zeiten beanspruchen würde: Die CODE-Definition kann gegenüber der Colon-Definition in ihrer Ausführungszeit auf jeden Fall vernachlässigt werden.

```
2VARIABLE AA
2VARIABLE BB

: X1 100 0 DO 0 0 DO AA 2@ BB 2@ UD*Q
                    2DROP 2DROP LOOP LOOP ;
: X2 100 0 DO 0 0 DO AA 2@ BB 2@ UD*Q^
                    2DROP 2DROP LOOP LOOP ;

: X1a       0 0 DO AA 2@ BB 2@ UD*Q
                    2DROP 2DROP LOOP     ;
: X3        0 0 DO AA 2@ BB 2@ UD*Q^^
                    2DROP 2DROP LOOP     ;

: X4 100 0 DO 0 0 DO AA 2@ BB 2@
                    2DROP 2DROP LOOP LOOP ;
\ High-Level-Ueberbau ("Overhead")
```

```
\ FFFFFFFF. FFFFFFFF. AA 2! BB 2! X1     \ 40 sec
\ 12345678. 12345678. AA 2! BB 2! X1     \ 38 sec

\ FFFFFFFF. FFFFFFFF. AA 2! BB 2! X2     \ 26 sec
\ 12345678. 12345678. AA 2! BB 2! X2     \ 25 sec

\ FFFFFFFF. FFFFFFFF. AA 2! BB 2! X4     \ 23 sec
\ 12345678. 12345678. AA 2! BB 2! X4     \ 23 sec


\ FFFFFFFF. FFFFFFFF. AA 2! BB 2! X1a    \ <1 sec
\ 12345678. 12345678. AA 2! BB 2! X1a    \ <1 sec

\ FFFFFFFF. FFFFFFFF. AA 2! BB 2! X3     \ 29 sec
\ 12345678. 12345678. AA 2! BB 2! X3     \ 22 sec
```

Der Vergleich wurde mit einem AMD-K6-2/500-System vorgenommen. Die 0 0 als Schleifenparameter der inneren Schleife sorgen dafür, dass diese (innere Schleife) die größtmögliche Anzahl von Malen durchlaufen wird. Zur Erinnerung: Das von mir verwendete Turbo-Forth ist ein 16-Bit-System.

Ergebnisse

(1) In High-Level-Forth ist das "russische" Verfahren völlig indiskutabel. Die Assembler-Lösung ist derart viel schneller, dass sie neben der High-Level-Lösung total untergeht. Genaueres konnte ich mit den von mir verwendeten groben Mitteln (Sekundenzeiger meiner Armbanduhr) nicht erkunden.

(2) Wenn schon "russisches" Verfahren in High-Level-Forth, dann gilt: Die Zahlen mit "mittlerer" Nullbitverteilung, 12345678., machen sich gegenüber den "Worst-Case-Zahlen", FFFFFFFF., bei denen sämtliche Bitstellen mit 1 besetzt sind, mit 30% weniger Ausführungszeit bemerkbar. Bei 12345678. braucht ja bei den Nullbits im Multiplikator keine Addition zum Zwischenergebnis vorgenommen zu werden.

(3) Bei den reinen Assembler-Lösungen fallen die Unterschiede zwischen FFFFFFFF. und 12345678. nicht ins Gewicht, auch nicht beim "russischen" Verfahren.

(4) Bei den reinen Assembler-Lösungen dauert das "russische" Verfahren (also das, was man normalerweise eine "Software-Multiplikation" nennen würde) nur etwa 30% länger als über die eingebaute "Hardware-Multiplikation" (Maschinenbefehl MUL).

(5) Ich habe die starke Vermutung, dass sich beim Übergang zur 64/64/128-Multiplikation (abermalige Verdoppelung der Bitzahl), bei der man ja auch mit MUL nicht mehr durchkommen würde und zusammenstückeln müsste, die Ausführungszeiten zugunsten der "russischen" Methode ändern.

(6) Die unter Punkt (4) genannte Feststellung gilt für den Fall, dass man unter "reiner Assembler-Lösung" eine in ein "normales" Forth-Programm eingebettete CODE-Definition versteht, bei welcher also die Stackoperationen für Ein- und Ausgabe zur eigentlichen Multiplikation hinzuzurechnen sind. Geht man nicht über den Stack, sondern über Tabellen und bettet man die Assembler-Lösung (mit einer großen Zahl von Multiplikationen, z.B. bei Matrizen- Operationen) in eine größere CODE-Definition ein, so ändern sich die Verhältnisse: Ich müsste dann den über X4 ermittelten High-Level-Aufwand von dem über X1 oder X2 ermittelten reinen Assembleraufwand abziehen und erhielte beim Vergleich "einerseits mit MUL" : "andererseits russisch" ein Verhältnis von 2:15 bis 4:17. Das spricht für die Güte der (beim Pentium-Adäquat K6-2/500) eingebauten Multiplikation MUL. (Man beachte aber, dass meine Messverfahren viel zu grob angesetzt sind und nur die Größenordnung wiedergeben können.)

Literatur

[All] Allwright, Ray: From the Net – Minimal Word Sets. Forthwrite 95, März 1998, S.28-32.

[Arn] Arndt, Jörg, und Christoph Haenel: Pi. Springer-Verlag Berlin 1998, S.100 ff.

[Bi1] Bitter, Martin: Neulich am Moerser Stammtisch. Vierte Dimension 2001, Heft 1, S.26-29.

[Bi2] Bitter, Martin: Karatsuba, Teil 1. Vierte Dimension 2001, Heft 2, S.28-31.

[Mit] Mittelbach, Henning: Turbo-Pascal in Beispielen. Stuttgart 1997, S.37.

[Pay] Paysan, Bernd: b16 - ein Forth-Prozessor im FPGA. Vierte Dimension 2003, Heft 1, S.26-34.

[Pod] Podschun, Trutz Eyke: Das Assembler-Buch. Addison-Wesley, Bonn 1996.

[Röc] Röckrath, Luidger: Microsoft-BASIC: Konzepte, Algorithmen, Datenstrukturen. Franzis-Verlag, München 1985, S.79.

[Sc1] Schleisiek, Klaus: MicroCore. Vierte Dimension 2002, Heft 3, S.9-10.

[Sc2] Schöning, Uwe: Algorithmik. Spektrum Akademischer Verlag, Heidelberg 2001, S.267 ff.

[Zie] Ziegenbalg, Jochen: Algorithmen. Heidelberg 1996, S.180-181.

Haben Sie Spaß an „mathematischen Basteleien“, an pragmatischen Beweisen, an zellulären Automaten oder einfach nur daran, bereits mehrfach Vergessenes immer wieder einmal spielerisch aufzufrischen? Dann sei Ihnen die nachstehende Adresse empfohlen:

`http://www.mathematische-basteleien.de/`

fep

Eine neue Art der Computerprogrammierung

Reaktionen auf einen Beitrag (VD 2/2005)

(siehe auch Seite 36)

Vorwort:
Wenige Beiträge in der Vierten Dimension haben bisher in Anzahl und Vehemenz ähnliche Reaktionen ausgelöst wie der unter der oben genannten Überschrift veröffentlichte Aufsatz von B. Hodson. Von vorsichtigem Interesse über erfreutes Staunen über „ein weiteres Forth“ bis zu offener Ablehnung, die sehr emotional artikuliert wurde, reichte das Spektrum der Rückmeldungen an die Redaktion.

In UK waren sowohl Hodsons Aufsätze als auch seine Intentionen schon vor einigen Jahren bekannt. Chris Jakeman hat sich bereits sehr früh mit Hodson auseinander gesetzt. Seine frühe Bewertung soll hier stellvertretend für alle Leserbriefe abgedruckt werden. *fep*

Zum Artikel "Eine neue Art der Computerprogrammierung" von Bernard Hodson aus dem VD-Heft 2/2005. Die nachfolgende kritische Rezension von Chris Jakeman (FIG UK) erschien im Forthwrite-Heft 96 (Mai 1998). Es ging um das von Bloor Research herausgegebene Buch "The Gene Machine" und um Bernard Hodsons Aktivitäten rund um den Begriff "Genetix". Chris bezieht sich in seiner Analyse auf Neal Bridges, und wir veröffentlichen das mit seiner freundlichen Genehmigung und freudigen Zustimmung.
Übersetzer: *Fred Behringer.*

Genetix - und was dahinter steckt.

Chris Jakeman

Genetix ist ein Programmiersystem, das, so wird uns gesagt, eine Software-Revolution anstoßen könne. Mit einem gefädelten Interpreter ausgerüstet, ist die dahintersteckende Technik für die Mitglieder der FIG UK auf jeden Fall interessant. Als mich Neal Bridges auf diese Dinge aufmerksam machte, ergriff ich also sofort die Gelegenheit, darüber zu berichten.

Genetix Software Inc. ist ein von Professor Dr. Bernard Hodson gegründetes Unternehmen. Hodson ist gebürtiger Engländer, lebt aber in Kanada und emeritierte kürzlich von der Ottawa University. Artikel, die für Hodsons Genetix-System werben sollen, wurden von der unabhängigen Berater-Firma Bloor Research in allen Teilen Großbritanniens in diverse Veröffentlichungsorgane gesetzt:

Okt 97	Computing Newspaper
Nov 97	New Scientist
Nov 97	Computing Newspaper
Feb 98	Computing Newspaper
Mrz 98	Engineering
Mai 98	Personal Computer World

Diese Artikel ähneln einander sehr. Es werden atemberaubende Behauptungen über die Möglichkeiten einer Verfahrensweise angestellt, die ganze 35 Jahre zum Heranreifen brauchte. Unter den geltend gemachten Ansprüchen sind: "ein modernes Textverarbeitungsprogramm ... würde nur ein paar Hundert Kilobytes an Speicher verbrauchen", "die bedeutendste Entwicklung im Computerwesen seit der Erfindung des Compilers" und "zeigt einen Weg auf, die Computer wesentlich effizienter zu gestalten, leichter zu programmieren und in der Lage, bei weitem mehr an Information zu speichern." Kürzlich hat Bloore ein Buch herausgebracht, "The Gene Machine" (95 engl. Pfund, bei Bloor Research), das etwas Licht in die Geschichte bringt.

Der Genetix-Begriff geht auf die frühesten Tage der Computerei zurück und stellt, so wird behauptet, eine Implementation der Universalmaschine dar, des theoretischen Computers, so wie er von Alan Turing beschrieben wurde, bevor der Computergedanke auf der Basis des Von-Neumann-Modells in die Praxis umgesetzt wurde.

Der Grundgedanke scheint in der Wiederverwendung von Software-Elementen zu liegen. Vereinigt man das mit der Kompaktheit von gefädeltem Code, so gelangt man zu Software mit drastisch reduziertem Aufwand an Ressourcen. Hält man sich vor Augen, wie klein der äußere Aufwand war, den Niklaus Wirth für Oberon benötigte (Oberon macht von Software-Wiederverwendung Gebrauch, verwendet aber keine Fädelung), so bin ich davon überzeugt, dass eindrucksvolle Ergebnisse möglich sind. Im Buch werden jedoch keinerlei Vergleiche angestellt und Bloor scheint nicht gewillt zu sein, auf Erfolge in Forth zu hören, die Genetix untergraben könnten.

Technisches

Genetix verwendet indirekte Fädelung, die von einer virtuellen Maschine ausgeführt wird. Wie Hodson glaubt, hat er das früher auf die Beine gestellt als Charles Moore. Genetix hat seine eigene Terminologie, die aber recht gut mit derjenigen von Forth verglichen werden kann. Der Hauptunterschied besteht darin, dass Genetix keinen Datenstack verwendet. Die Operatoren arbeiten stattdessen mit einem, zwei oder drei Operanden.

Forth	Genetix
Wort	Gen
Primärwort	Grund-Gen
Dictionary	Gen-Bank

In gewisser Weise macht Genetix einfach einen bizarren Eindruck. An Stelle von IF ... THEN oder DO ... LOOP gibt es eine "512-stufige geschachtelte Folge von Schaltern", die man als endlichen Automaten ansehen kann, welcher von bestimmten Genetix-Primärworten gesteuert wird.

Es wird Festkomma-Arithmetik verwendet. Zur Darstellung werden 128 Bytes (!) benötigt, für jede Ziffer der betreffenden Zahl ein ganzes Byte. Das höchstwertige Byte enthält das Vor-

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.

zeichen und legt die Stellung des Dezimalkommas fest.

Eine Genetix-Anweisung kann beispielsweise wie folgt aussehen: DRWLN 20, 30, 50,408, [goto] 9 . Und Gene sind: ascii blank copyf deque epass grply hits? jdays lcmps msq25 nextu . grply scheint Polygone zu zeichnen.

Genetix hat einen Kern von Bibliotheksfunktionen. Der Prototyp enthält davon etwa 26 Kilobytes, mit 600-700 Genen.

Möglichkeiten

Ich habe Neal Bridges gefragt, ob er es für möglich hält, dass Genetix vorteilhaft zu einer neuen Art der Programmierung führen könne. Vom Studium des Buches her gesehen, sagt Neal "nein". "Hodson verkauft nur an Leute, denen nicht klar ist, dass das, was er ihnen verkauft, bereits gratis erhältlich ist. Man sollte sie mit Forth und endlichen Automaten beeindrucken. Man sollte ihnen klarmachen, dass diese Dinge nicht neu sind und dass sie nicht von Hodson erfunden wurden."

Anwendung

Das Buch gibt keinen Hinweis darauf, warum Genetix so lange gebraucht hat, um von seinen Wurzeln, die 35 Jahre zurückliegen, heranzureifen. Die Erfolge waren bisher bescheiden. Es gibt ein noch nicht ausgereiftes Entwicklungssystem und es wurde die Arbeit an einem MPEG-Decoder zur Komprimierung und Dekomprimierung von Video- und Sound-Signalen begonnen.

Schlussfolgerung

Hodson macht das Beste aus seinem bescheidenen Angebot und er versteht es gut, Aufmerksamkeit für sein Anliegen zu erringen, ohne allzu viel von seinen eigentlichen Ideen preiszugeben. Die technischen Aspekte, von denen kaum etwas veröffentlicht wurde, scheinen nicht isoliert in der Welt zu stehen und einzig und allein ein (erfolgreicher) Vergleich mit den zur Zeit bestehenden Techniken könnte mich davon überzeugen, dass die Genetix-Geschichte es verdient, weiterverfolgt zu werden.

Da ich Forthler bin, haben natürlich Pioniere meine Sympathie, aber meines Erachtens ist Hodson (und Bloor) ganz schlecht darüber informiert, was alles auf dem Gebiet der Embedded-Software bereits erreicht wurde, und die Zukunft von Genetix wird sich als recht kurz erweisen.

Chris Jakeman, FIG UK

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

Gehaltvolles

zusammengestellt und übertragen
von Fred Behringer

VIJGEBLAADJE der HCC Forth-gebruikersgroep, Niederlande

Nr. 51, August 2005

Forth vanaf de grond 8
Ron Minke

Das ist, wie der Autor sagt, die achte (und vorläufig letzte) Folge des Versuchs, ein Atmel-AVR-Forth-System "from scratch" hochzuziehen. Ein allerletztes Problem: Was tun, wenn man aus der Atmel-Reihe einen Prozessor wählt, der kein externes RAM zulässt, oder wenn man aus anderen Gründen kein externes RAM verwenden will? Das vorgefertigte Forth bleibt im Flash-Speicher, ein paar Dinge werden im spärlichen Onboard-RAM erledigt, FIND muss angepasst werden. Das interne RAM beispielsweise beim Mega162 liegt im Bereich von 0100 bis 0500. Der in Frage kommende Teil des Flash-Speichers geht von 0500 bis 1FFF. Speicherzugriffe unterhalb 0500 werden mit den Ld-Befehlen erledigt, Zugriffe oberhalb 0500 mit den Lpm-Befehlen.

Doe meer met EXIT
De Schoolmeester

Ein Lieblingsthema des Autors (der seine Tutorials im Vijgeblaadje unter dem Pseudonym "Der Schulmeister" veröffentlicht): Kontrollstrukturen sind bis auf IF...THEN überflüssig. Insbesondere kann man auf die CASE-Konstruktion von Eaker, so der Autor, gut und gern verzichten. Genauer: Mit IF THEN EXIT und einem weiteren Wort, das er RE nennt, kann man alles erschlagen. Und das sogar noch übersichtlicher als beim Eaker-Konstrukt - wenn man sich an bestimmte Darstellungsregeln hält. Der Autor bringt uns das in der von seinem Buch (siehe unten) her bekannten Leo-Theo-Dialogform nahe. Ein Beispiel für die Entbehrlichkeit von ELSE:

Die Definition

```
: VORZEICHEN ( n -- ) 0< IF ." negativ" ELSE ." positiv"
                        THEN ;
```

kann als

```
: VORZEICHEN ( n -- ) 0< IF ." negativ" EXIT THEN
                              ."  positiv" ;
```

geschrieben werden.

EXIT lässt in der Aussprungrichtung Spaghetti-Code zu. EXIT kümmert sich aber nicht um die paarweise Übereinstimmung von Kontrollelementen und kann dadurch helfen, komplizierte Situationen übersichtlich zu gestalten. Mit EXIT ist es leicht, einen binären Abfragebaum aufzubauen: Die THENs werden auf allen Abfrageebenen in die Verästelungen eingebunden und es ist am Schluss keine THEN-Akrobatik mehr nötig. Interessant ist aber der Einwand des Advocatus Diaboli, im Dialog des Autors heißt er Leo, nämlich dass die Verwendbarkeit von EXIT zum Aussprung dadurch erkauft wird, dass auf jeder Abfrageebene die betreffende Definition durch ein ";" unmittelbar hinter der Antwort von IF bei Nichterfülltsein der Bedingung beendet werden muss. Es ist klar, das führt uns der Autor mit einem weiteren Beispiel drastisch vor Augen, dass das auch für jegliche Art von CASE-Konstruktion gilt. Im Übrigen können bei der direkten CASE-Konstruktion über IF THEN EXIT auch Kleiner-als- oder Größer-als-Bedingungen, wie uns der Autor zeigt, ganz leicht und unmittelbar einsichtig eingebaut werden. (Der Rezensent: Beim Eaker-Konstrukt müsste man andere Abfragen als die auf Gleichheit über ein ganzes Spektrum von OFs (<OF >OF XOR-OF AND-OF etc) auffangen, und das wäre sicher auch nicht einprägsamer.)

Reklame für ein Buch

-- voor Fred --

Een Forthprogrammeur met stackangst in Zweden
heeft jarenlang de stack gemeden.
Zijn werkwijze was een discutabele:
hij maakte voor alles een variabele.
Er kwam een einde aan de gekte
toen hij 't woordje dup ontdekte *)
en verlicht de woorden sprak:
leve de stack vanaf heden!

*) dankzij onze Forthcursus, die hij zich ten einde raad had aangeschaft in de Duitse vertaling van Fred Behringer.
"De programmeertaal Forth"
www.forth.hccnet.nl/product.htm
"Die Programmiersprache Forth" -- www.mv-buchhandel.de

Albert Nijhof

Und hier ein Übertragungsversuch:

-- für Fred --

Ein Forth-Programmierer von den Hebriden
hatt' jahrelang stets den Stack vermieden.
Seine Vorgehensweis' war 'ne diskutable:
Er machte für alles 'ne Variable.
Sein Spleen fand jedoch ein jähes Ende,
als DUP er entdeck- *)
te und hell erleuchtet dann sprach: Ich verwende
jetzt nur noch den Stack.
Und er klang sehr entschieden.

Fred Behringer

*) Dank unseres Forth-Arbeitsbuches, das der Programmierer sich in Form der deutschen Übersetzung von Fred Behringer angeschafft hatte, als er nicht mehr weiterwusste.

28. September 2005

Hallo Friederich!
Fred hat mich freundlicherweise daran erinnert, dass Du mit der nächsten Ausgabe der Vierten Dimension beginnen willst. Ich will schauen, ob ich dich noch mit einigen Berichten über SVFIG-Treffen einfangen kann, die ich dir bisher nicht gesandt habe.
Ich sehe, dass ich dir in meinem letzten Bericht am 21. Mai gesagt habe, dass Du die nächsten drei Monate wohl nichts von mir hören wirst. Da inzwischen über vier Monate vergangen sind, mögen einige Erklärungen (oder Entschuldigungen?) in Ordnung gehen. Da sich aber deine Leser wohl weniger dafür interessieren, werde ich versuchen, so weit wie möglich beim Thema Forth zu bleiben, auch wenn ich über meine eigenen Experimente damit berichten muss. Bitte streiche alles, was deiner Meinung nach nicht in eure Zeitschrift passt.

Ich ging nur zur Morgensitzung am 25. Juni und nutzte den Nachmittag, um einigen meiner Schachstudenten bei einem Turnier zuzusehen. Um ehrlich zu sein, die beiden Stunden, die Dr. Ting über seine momentanen Projekte berichtete, waren wieder außerhalb meines Interesses und meines Verständnisses. Die anderen acht Teilnehmer folgten seinen Ausführungen wohl mit voller Aufmerksamkeit. Hier sind einige Notizen, die ich von Dr. Tings Vortrag mitbrachte: „Einen ADuC7020 und ADM202 kann man in einen DB9-Stecker unterbringen, um so einen sehr kleinen Computer aus einem PC COM-Port zu bilden.“ und „Jedes Forth-System ist ein redundantes Forth-System“.

Aufgrund meiner Schachaktivitäten habe ich das Juli-Treffen komplett verpasst. Soweit ich weiß, hat eine kleine Gruppe Forther am Morgen die Firma DigiCom besucht und die Wahlen der SVFIG-Offiziellen am Nachmittag abgehalten.

Im August gab es kein Treffen. Aber, ich verbrachte den ganzen letzten Samstag beim September-Treffen. Dr. Ting hat am Morgen über sein neues Projekt, welches er als Hyper Massiv Parallel Prozessor (HMPP) Array bezeichnet, gesprochen. Dies stellt ein `massives Update` des GAPP (geometrisch-arithmetischen Parallel Prozessor) Projektes dar, an dem er vor 30 Jahren bei Lockheed gearbeitet hat. Jetzt kann er 1 Million Prozessoren in einem 1000x1000 Array auf einem einzelnen Chip unterbringen. Auf dem ursprünglichen GAPP waren dies 72! 1 Milliarde (1000 Millionen) Prozessoren sind eine Möglichkeit in der Zukunft.
Tings Vortrag hatte drei oder vier Nicht-SVFIG-Mitglieder (frühere oder momentane Lockheed-Beschäftige) angelockt. Sie verliessen uns allerdings am Nachmittag und ließen ein Dutzend oder so von unseren regulären „Kern“-Forthern zurück, die das Treffen mit einigen Diskussionen und dem Vortrag von Dr. Glen Hayden fortsetzten. Glen war gebeten worden, sein ‘Ohren-Trainings-Programm‘ für Musikstudenten zu beschreiben, welches er mit seinem eigenen MVP-Forth geschrieben und auf der 1990 FORML Konferenz präsentiert hatte (Es erschien auch in Volume XII/5 der Forth Dimensions, Jan/Feb. 1991).

Es gibt einen Konsens in der SVFIG, dass es eine gute Idee wäre, das Programm zur Arbeit auf den aktuellen Rechnern zu aktualisieren. Weiterhin könnte man es in Public Domain als nützliches Werkzeug geben und als Demonstration der Effektivität der Forth-Programmierung. Das ursprüngliche Programm basierte auf der Ansteuerung des PC-Lautsprechers durch den 8253 Timer, welcher anscheinend nicht mehr verfügbar ist. So suchen wir nach dem einfachsten Ersatz zur Erzeugung musikalischer Töne auf einem PC, wobei wir den Rest von Glens Programm und seinen Kommentaren im wesentlichen unverändert lassen wollen.

Ich bin froh, dass wir Glen zu dem Vortrag überreden konnten, da er uns, einmal begonnen, für über eine und eine halbe Stunde mit Geschichten, Anekdoten und Weisheiten unterhielt, die er bei diesem Projekt gesammelt hatte. Ich muss Glen für alle die guten Tipps über lesenswerte Bücher danken, die er uns gab. Als Konsequenz des Samstags-Treffens habe ich eine Kopie von Platos „Republik“ für mich gekauft und suche nach Lakoff und Johnsons “Philosophy in the Flesh” (ich glaube Glen sagte, die deutsche Übersetzung lautet “Die Weisheit des laechelnden Lebens”).

Wir sind nicht sicher, ob es genügend Interesse für ein Treffen im nächsten Monat gibt. Auf jeden Fall erwarten wir den jährlichen Forth-Tag am 19. November. Wir hoffen, dass Chuck Moore Zeit hat, diesen mit uns zu verbringen.

Ich vergaß zu erwähnen, dass ich am 8. September die Gelegenheit hatte, das Museum für Computer-Geschichte in Mountain View zu besichtigen und dort ein Reihe von Experten-Vorträgen über die Geschichte des Computer-Schachs zu hören. Einige der Köpfe hinter Kasparovs Schreckensgegner – Deep Blue – waren dort. Monty Newborn hat moderiert, John McCarthy, David Levy, Ed Feigenbaum und Murray Campbell waren dabei. Ich erkannte Donald Knuth und Donald Norman im Publikum, ebenso wie auch einige unserer SVFIG-Mitglieder. Das Museum macht sich ganz gut, mit einer Menge Unterstützung aus dem „Valley“.

Friederich, ich wollte dir über meine eigenen Kämpfe mit Forth berichten, die im letzten Frühjahr in eine neue Runde gingen. Das war als Fred und Martin mir halfen, etwas Leben in meine LEGO-RCX-Roboter einzuhauchen. Dies hat sich mit Freds geduldigen Lektionen fortgesetzt (nachdem ich erkannt hatte, dass sogar Ralph Hempel sein pbFORTH nicht mehr länger zu unterstützen scheint). Wir nutzen dazu ein altes französisches Forth, ins Deutsche übersetzt, welches unter DOS läuft; mit meinen eigenen englischen ‚Untertiteln’. Ich habe es immer noch und ich mag es. Endlich, habe ich etwas mit meinen begrenzten Fähigkeiten zum Laufen gebracht, aber nur da Fred mich mit seiner helfenden Hand gelenkt hat. Ich erkenne jetzt, dass Forth etwa so wie ein Fahrrad ist, bei dem man den Lenker entfernt hat. Man kann damit wirklich zu einigen Zielen fahren, aber nur, nachdem man die Kombination für das Sicherheitsschloß gefunden hat, welches jemand am Antriebszahnrad befestigt hat.

So, das war es für heute. Streich den letzten Absatz, wenn du willst. Unglücklicherweise fällt es mir leichter, mich an die ganzen Frustrationen, die ich mit Forth hatte, zu erinnern, als eine überzeugende Notiz (vielleicht sogar an mich selbst) zu schreiben über die Tugenden von Forth in der heutigen Welt des „Hyperallesmöglichen“. Die Hauptvorteile von Forth waren immer die Verbindungen zu den Leuten, die ich auf dem Weg getroffen habe; die Leute, die mich unterhalten und weitergebildet haben und die willens waren, ihr Wissen und ihre Zeit mit mir zu teilen.

With best regards, *Henry*

Hallo Henry,
Es ist schön, zu lesen, dass der älteste Forth-Novize der Welt positive Erfahrungen mit Forth gemacht hat. Vielleicht berichtest du uns über deine Erfahrungen mit den RCXen?

Glückauf, *Fritz*

Liebe Freunde,

fast drei Monate sind seit meinem letzten Brief an euch über die SVFIG-Aktivitäten vergangen. Dieser Fakt kommt mir langsam zu Bewußtsein, da ich mich immer schuldiger fühle, dass ich euch so lange nicht geschrieben habe. Nun, nachdem ich die Webseite der SVFIG überprüft habe, sehe ich, dass Ihr weitere drei Monate keine Berichte bekommen würdet. Daher will ich zumindest eine kurze Notiz senden, um euch zu versichern, dass die SVFIG noch aktiv ist.

Wie Ihr seht, wurde unser Mai-Treffen auf den Juni verschoben, da es Konflikte mit dem Zeitplan unseres Gastgebers, des Cogswell College gab. Dann am 25. Juni bin ich wahrscheinlich den ganzen Tag mit einem Kinder-Schachturnier beschäftigt. Im Juli bin ich für den Ersatz einer meiner Hüften vorgesehen und es kann wohl sein, dass ich bis zum August zu keinem anderen Treffen komme. Die Zeit vergeht wie im Fluge und vergeht noch viel schneller für mich, seit Martin Bitter und Fred Behringer die Herausforderung angenommen haben, mir mit den LEGO-Mindstorm-Robotern, pbForth und Forth im allgemeinen zu helfen. Wenn das so weiter geht, könnte sich mein Status vom ältesten Forth-Neuling zum ältesten Forth-Studenten unter der Anleitung zertifizierter Lehrkräfte wandeln. Jedoch, die Graduierung durch die "Forth-Akademie" ist noch ein ferner Traum für mich.

Mein letzter Bericht an euch war vom SVFIG-Treffen am 22. Januar. Am 26. Februar schrieb ich an Friederich:
"Heute war überhaupt nicht mein Tag. Ich eilte heute morgen zum SVFIG-Treffen am üblichen Platz im Cogswell College und fand überhaupt keinen Forther dort! Weißt du, mit all meinen Problemen mit dem AOL-Internet Service hatte ich die Ankündigung verpasst, dass das heutige Treffen im Museum für Computer-Geschichte stattfinden sollte. Ok, ich bin da nicht noch hingegangen und so endet mein Bericht für heute bei Null. Du kannst unter _www.forth.org/svfig/_ selber sehen, was für heute geplant war."

Ich habe es am 26. März allerdings besser gemacht und habe die beiden Stunden am Morgen damit verbracht Dr. Tings Fortschritte mit seinem Forth Digital-Speicher Oszilloskop anzuhören. Eine andere Verpflichtung zur Unterstützung bei einem Schul-Schachturnier hielt mich von den Nachmittagsveranstaltungen fern.

Aber ich repräsentierte ungefähr 7% der Teilnehmer des Treffens am 30. April. Das blieb recht konstant bis zum Ende des Treffens; dieses mal eine Stunde eher als sonst; wieder aufgrund von Schwierigkeiten mit dem Zeitplan des Cogswell College.
Ting sprach über die Modellierung mittels SPICE, speziell WinSpice, welches er als wertvollen Download aus dem Web empfahl. Tom Gregory gab eine Einführung in die finanziellen Aspekte einer neuen Produktentwicklung: Geschäftsplan, Finanzierung, Gewinnquoten, die benötigt werden, um Investoren anzuziehen usw. Nach einer langen Mittagspause, Einführungen und mehr oder weniger zufälligen Diskussionen hat Ting das Treffen mit der Besprechung eines Buches, das er kürzlich las, beendet.

Wie ich schon vorher gesagt habe, wären die SVFIG-Treffen zumeist eine gesellige Zusammenkunft für "Forther", so etwa wie bei den Wochenend-Cricket-Wettbewerben, die in und um das Silicon Valley recht populär sind. Ja, unsere Sportkultur wird Tag für Tag immer internationaler. Ich habe euch vielleicht schon geschrieben, dass jemand eine Statistik erstellt hat, dass in der Stadt Fremont, quer über die Bucht vom Silicon Valley, ca. 150 Sprachen von den 210.000 Einwohnern gesprochen werden.
Die anderen, nahe benachbarten Städte im San Francisco Bay Gebiet sind ethnisch nicht ganz so verschiedenartig, jedenfalls wenn wir die Leute, die hier nur arbeiten, nicht zählen.

Friederich, ich hoffe du bist mit deinem Umzug fertig und hattest die Chance, dich wieder auszuruhen. Ich wünschte, ich hätte mehr interessante Dinge zu berichten. Vielleicht später, falls UND sobald ich Dr. Behringers Kurs über die RCX-Roboter bewältigt habe?
(Das sollte ein Bool'sches UND sein!)

Mit herzlichen Gruessen, *Henry*

Übersetzer: Thomas Beierlein

Hinweis:

Das Buch von George Lakoff und Mark Johnson, “Philosophy in the Flesh”, wurde unter anderem von Harper Collins Publisher veröffentlicht und bisher nicht ins Deutsche übersetzt.

“Weisheit des lächelnden Lebens” ist ein Buch von Lin Yutang, erschienen im Insel Verlag und wird in einer Kurzbeschreibung als “Standardwerk der Lebenskunst” und “Bestandsaufnahme der westlichen Zivilisation” bezeichnet .
Viel Spaß beim Lesen.

Birgit Prinz

Teil II

MicroCore anwenden

Report über eine zutiefst befriedigende Instantiierung

Klaus Schleisiek

kschleisiek@send.de
www.microcore.org

Teil I dieses Beitrages ist in der VD 02/2005 erschienen.

5 Spezialinstruktionen

Eine weitere Möglichkeit für Codevereinfachungen bietet die Implementation von Spezialinstruktionen, die sehr anwendungsbezogen sein können. Außer, dass sie den Code vereinfachen und ihn lesbarer und deshalb zuverlässiger machen, sparen sie auch noch Energie, weil die Anzahl der Instruktionen, die ausgeführt werden müssen, um eine bestimmte Datentransformation zu verwirklichen, möglicherweise soweit reduziert werden kann, dass der gesamte Prozessor mit der halben Taktgeschwindigkeit oder sogar noch darunter betrieben werden kann.

5.1 Komplexe Mathematik

Dies sind keine "Spezialinstruktionen" im engeren Sinne, aber ich führe sie hier auf, da ich sie bisher noch nirgends beschrieben hatte. Dieses sind "Mathematik-Schritt-Operationen", die für jedes einzelne Bit des Datenworts wiederholt ausgeführt werden müssen, um das Ergebnis zu produzieren. Alle diese Operationen arbeiten gleichzeitig auf drei Registern: den beiden obersten Elementen auf dem Datenstack (TOS und NOS) und dem obersten Returnstackelement (TOR).

```
MULTS NONE ALU Opcode: mults
```

Bevor MULTS wiederholt ausgeführt werden kann, müssen der Multiplikand, der Multiplikator und das zukünftige Produkt in "ihren" Registern bereitgestellt werden, und am Ende müssen die Stacks aufgeräumt werden. Dazu sind 4 Instruktionen erforderlich und deshalb benötigt das Forthwort UM* data_path_width + 4 Zyklen.

```
0DIVS NONE ALU Opcode: 0divs
      \ Registerinitialisierung
UDIVS NONE ALU Opcode: udivs
      \ vorzeichenloser Divisionsschritt
LDIVS NONE ALU Opcode: ldivs
      \ letzter Divisionskorrekturschritt
```

Das Initialisieren und Bereitstellen der Register für die Division ist verglichen mit der Multiplikation sehr viel komplexer und deshalb gibt es die Instruktion 0DIVS. Zusätzlich wird bei der Division ein letzter, irregulärer "Korrekturschritt" benötigt und mit LDIVS realisiert. Werden diese drei Instruktionen genutzt, dann dauert das Forthwort UM/MOD gleichfalls data_path_width + 4 Zyklen.

5.2 Byteverarbeitung

Diese Instruktionen sind auch ziemlich generell und dienen dazu, Bytes zu verarbeiten. Sie könnten durch mehrfache Anwendung der Bitshiftopcodes emuliert werden, wurden aber aus Effizienzgründen als Einzyklus-Instruktionen realisiert. Ihre Semantik sollte eigentlich auf Grund der Namensgebung klar sein.

```
UP   NONE ALU Opcode: 256*  ( n -- n*256 )
DOWN NONE ALU Opcode: u256/ ( u -- u/256 )
6    BOTH USR Opcode: 256/  ( n -- n/256 )
```

5.3 Pseudo-DMA

```
0 PUSH USR Opcode: ide@
    ( bufaddr -- bufaddr+1 )
0 POP  USR Opcode: ide!
    ( bufaddr -- bufaddr+1 )
```

Diese zwei Instruktionen wurde speziell implementiert, um 16-Bit Daten zwischen der IDE-Diskschnittstelle und dem Pufferspeicher zu transferieren. Die jeweilige Pufferspeicheradresse befindet sich auf dem Stack und IDE@ überträgt ein 16-Bit Datenwort von der Platte in den Puffer, und IDE! überträgt ein 16-Bit Datenwort vom Puffer auf die Platte. Gleichzeitig wird dabei die Adresse auf dem Stack inkrementiert, so dass sich der nächste Transfer unmittelbar anschließen kann. Ein Sektor auf der Platte ist 512 Bytes bzw. 256 Worte lang, so dass diese "DMA-Instruktionen" 256-mal hintereinander ausgeführt werden müssen, um einen vollständigen Sektor in 256 Zyklen zu übertragen, wobei der Prozessor während dieser Übertragung jederzeit ohne Verzögerung noch auf Interrupts reagieren kann.

5.4 Datenkodierung und -speicherung

Die Datenkodierung des MCS ist eigentümlich und auf Störsicherheit optimiert. Die Platte wird wie ein Magnetband behandelt, d.h. dass die Daten ohne Blockstruktur kontinuierlich gespeichert werden. Statt dessen ist jedes einzelne Datum mit einer variablen Anzahl von Markierungsbits gekennzeichnet, die in den höherwertigen Bits abgelegt sind. Dadurch wird erreicht, dass bei Ausfall eines Sektors nur die Daten aus diesem Sektor verloren sind. Der Markierungscode ist so konstruiert, dass der Dekodierungsprozess sich schnell wieder auf den Datenstrom einsynchronisieren kann.

Die abgetasteten Daten werden folgendermaßen kodiert: Der neueste Wert wird vom vorhergehenden Wert subtrahiert. Wenn die vorzeichenbehaftete Differenz in sieben Bits unter-

gebracht werden kann, wird sie als ein Byte mit dem Markierungscode '0' gespeichert. Wenn die Differenz nur in 14 Bits passt, wird sie als 16-Bit-Wort und dem Markierungscode '10' gespeichert. Ansonsten wird die gesamte vorzeichenbehaftete 24-Bit-Zahl als ein 32-Bit-Datum und dem Markierungscode '11110' gespeichert. Neben den Daten werden andere Markierungen genutzt, um Zeit- und Statusinformationen zu speichern. Darunter ist insbesondere ein 32-Bit langes Synchronisationswort. Egal wie "verwirrt" der Decoder ist, nach der Synchronisationsmarkierung ist er wieder auf den Datenstrom einsynchronisiert.

Von der Datenverarbeitung her gibt es zwei Probleme mit diesem Ansatz:

1. Die Datenkodierung
2. Die Speicherung dieser Daten variabler Länge in einem 16-Bit breiten Pufferspeicher.

Beide Prozesse sind unter den Gegebenheiten eines traditionellen Instruktionssatzes aufwändig zu berechnen und in den Vorläufersystemen beschränkten diese beiden Prozesse den maximalen Datendurchsatz. Um die Vorteile der Spezialinstruktionen deutlich zu machen, werde ich hier den erforderlichen Code ohne und mit den Spezialinstruktionen darstellen.

Dies hier ist der uCore Code, der gebraucht wird, um mit den Standardinstruktionen die Daten zu kodieren und zu speichern:

```
: split  ( 32b -- l16b h16b )
  dup $FFFF and swap u256/ u256/ ;
: wsplit ( 16b -- l8b h8b )
  dup $FF and swap u256/ ;
: buf8! ( 8bit -- )
  >r idebuf_ptr @ 2/
  carry IF    ld swap r> 256* or swap !
        ELSE  r> $FF and swap !
        THEN
  idebuf_ptr ld swap 1+
  dup [ #idebuf_top 2* ] literal u>
  IF  idebuf_flush drop #idebuf_addr 2*
 THEN
 swap ! ;

: buf16! ( 16bit -- )
  wsplit buf8!  buf8!  ;
: buf24! ( 24bit -- )
  split  buf8!  buf16! ;
: buf32! ( 32bit -- )
  split  buf16! buf16! ;
```

Diese vier Worte sind erforderlich, um die kodierten Daten mit variabler Breite im Puffer zu speichern, und wenn er voll ist, wird der Puffer mit dem Wort IDEBUF_FLUSH auf die Festplatte geschrieben. Die aktuelle Position im Puffer ist in der Variablen IDEBUF_PTR abgelegt.

Unter Nutzung der obigen vier Pufferspeicheroperatoren können die Daten nun folgendermaßen kodiert und gespeichert werden:

```
: abs  ( n -- u )
  neg IF  0 swap-  THEN ;
: encode ( sample channel# -- )
  under             \ val val ch
  samples +         \ val val addr
  ld >r swap r> !   \ val old
           \ store val in samples buffer
  under -           \ val difference
  dup abs           \ val difference
                    \ |difference|
  dup $40 <    IF drop $7f and buf8!
                  drop EXIT
               THEN
  dup $1000 < IF drop $1FFF and $C000
                                or buf16!
                  drop EXIT
               THEN
     $80000 < IF $FFFFF and $E00000
                       or buf24!
                  drop EXIT
               THEN
  drop $7FFFFFF and $F0000000 or buf32! ;
```

Diese Anhäufung komplexen Codes kann nun mit fünf Spezialinstruktionen drastisch vereinfacht werden - vier Speicheroperationen, die sich auf eine Adresse in einem internen Register beziehen, sowie einer Operation für die Kodierung.

```
1 POP  USR Opcode: buf8!    ( 8b -- )
2 POP  USR Opcode: buf16!   ( 16b -- )
1 NONE USR Opcode: buf24!   ( 24b -- 16b )
2 NONE USR Opcode: buf32!   ( 32b -- 16b )
```

Dies sind die vier Speicheroperatoren, die die Probleme der variablen Datenbreite lösen. Das Problem der Byteausrichtung wird durch ein internes Zwischenspeicherregister im FPGA gelöst. BUF_POINTER ist ein internes Register, das die nächste freie Adresse im Pufferspeicher enthält und es wird durch obige Instruktionen passend inkrementiert.

Was aber passiert, wenn der Puffer voll ist? Das Ende des Datenpuffers ist eine absolute Hardwarekonstante, die in das FPGA hineinsynthetisiert ist. Deshalb "weiß" das FPGA, wenn es den BUF_POINTER jenseits des verfügbaren Speichers zu erhöhen versucht. In dem Augenblick wird `#sema_buf` zurückgesetzt (siehe "Semaphore" weiter oben) wodurch der Prozess, der den Puffer auf die Festplatte schreibt, aktiviert wird, und gleichzeitig wird eine Exception ausgelöst, wenn versucht wird, eine der vier Speicheroperationen auszuführen. Sobald der gesamte Pufferspeicher auf die Festplatte geschrieben ist, wird mit `#idebuf_addr BUF_POINTER !` der Pufferspeicherzeiger wieder an den Anfang des Puffers gesetzt und gleichzeitig wird `#sema_buf` wieder gesetzt, so dass der Speicherungsprozess gesperrt und die Pufferspeicheroperatoren wieder möglich werden.
Die Datenkodierung wird mit der Ein-Zyklus-Instruktion TAG realisiert, die in VHDL sehr viel einfacher (und damit zuverlässiger) als in uCore Forth-Code realisiert werden kann.

```
ENCODE NONE ALU Opcode: tag
  ( sample previous -- sample code )
```

TAG erwartet das aktuelle und das vorhergehende Sample auf

dem Stack und erzeugt daraus den Code gemäß den Kodierungsregeln. Die Information, ob die Information in einem Byte, in einem 16-Bit Wort oder in einem 32-Bit Wort kodiert ist, wird im Carry- und im Overflow-Flag abgelegt. Wenn man diese Spezialinstruktionen nun zusammen nutzt, entsteht daraus eine elegante und effiziente ENCODE Routine:

```
: encode ( val ch# -- )
   samples + ld >r tag
   carry IF   buf8!   ELSE   ovl IF   buf32!
    THEN  buf16!  THEN
   r> ! ;
```

Dies ist ein echter Energiesparer, weil es dadurch möglich wird, den Prozessor nur noch mit einem Viertel der Taktgeschwindigkeit laufen zu lassen, so dass der Stromverbrauch für die digitale Steuerung auf 20 mW reduziert wird.

6 Lehren aus dem Projekt

Die Migration von uCore 1.10 (wie auf der Website www.microcore.org publiziert) zu uCore 1.20 war immer noch ein reines Simulationsexperiment, ohne den Code wirklich in Hardware zu testen. Trotzdem wurde durch die Realisierung des "Top-of-Return-Stack" (TOR) Registers, das wegen der komplexen Arithmetikinstruktionen erforderlich wurde, klar, dass etwas mit der Anbindung des Datenspeichers und des Returnstacks an den Kern fundamental falsch war. uCore 1.20 ist das Ergebnis einer grundlegenden Restrukturierung der Speicherschnittstelle. Dadurch ist es einfach geworden die Speicherschnittstelle so, wie wir es von konventionellen Prozessoren gewohnt sind, sowohl für interne Register als auch für externe Peripheriechips zu nutzen.

Trotzdem, uCore 1.20 war noch immer ein simuliertes "Papierexperiment". Nichtsdestotrotz war es ein sinnvoller Ausgangspunkt, um uCore auf dem Prototypingboard zu instantiieren.

Die Portierung auf die Geolon-MCS Hardware und die Erfahrungen einer realen Anwendung haben eine ganze Reihe interner Änderungen herbeigeführt, so dass der VHDL besser strukturiert ist und damit der Code einfacher auf unterschiedliche Hardware angepasst werden kann. Als nächstes muss jetzt noch dieser Code als "uCore 1.30" in eine allgemeingültige Form "zurückportiert" werden.

Jedoch, die wichtigsten Lehren dieser Entwicklung hören sich für Softwareingenieure nur allzu bekannt an:

Versuche niemals, eine allgemeingültige Lösung zu realisieren, wenn in Wirklichkeit nur ein spezielles Problem gelöst werden muss. Die sich langsam entwickelnde Wirklichkeit ist üblicherweise vollkommen anders, als man sich zunächst vorstellt. Löse nur das Problem, das konkret gestellt ist; in möglichst verständlicher Form. Die Software muss sowieso vollständig neu geschrieben werden, wenn der Funktionsumfang erweitert wird. Das trifft auch auf VHDL-Code zu.

7 Nachtrag

Inzwischen ist uCore auf ein weiteres FPGA portiert, um einen vierkanaligen Analog-Digitalwandler zu steuern und die einlaufenden Signale zu filtern und zu dezimieren. Dabei findet uCore vollständig "innerhalb" des FPGA statt: Sowohl der Datenstack als auch der Return-Stack, der Datenspeicher und der Programmspeicher liegen in internen RAM-Blöcken und der Programmspeicher wird bereits während der Konfiguration des FPGA mit "seinem" Programm belegt. Es handelt sich also um eine Anwendung, deren Programmsteuerung nach Außen hin gar nicht mehr sichtbar ist. (Es gibt aber nach wie vor für die Weiterentwicklung des Programms eine Debugschnittstelle über ein RS232-Interface mit 115 kBaud, die jederzeit aktiviert werden kann.)

Hier ist uCore erstmalig in einem FPGA mit reichlich Hardwaremultiplizierern (Virtex2) realisiert, so dass `UM*` in einem Zyklus (32x32->64 Bit) ausgeführt wird.

Bei dieser Anwendung sind folgende neue Instruktionen entstanden:

Die Instruktionen `Z!` und `FIR`, die gebraucht werden, um ein vierfach dezimierendes digitales Tiefpassfilter zu realisieren. Bei 309 Taps beträgt seine Sperrdämpfung -110dB bei einer Durchlasswelligkeit von 0,05 dB und einer Eckfrequenz von 0,46*Abtastrate.

Mit `Z!` wird jeder einlaufende Abtastwert in einem Prozessorzyklus im zyklischen Z-Puffer gespeichert und `FIR` ist eine "Autorepeat"-Instruktion, die automatisch die Vektorsumme eines FIR-Filters bildet und dabei je Tap nur einen Zyklus braucht. Alle vier Kanäle können bei einer Abtastrate von 1000 Samples/Sekunde bequem mit einem externen Prozessortakt von 6 Mhz entsprechend einem Prozessorzyklus von 330 ns abgearbeitet werden. Möglich wäre mit der vorhandenen Hardware ein externer Takt von 25 MHz.

Aus dem "autorepeat" der `FIR`-Instruktion ist die `TIMES` Instruktion entstanden, mit der eine nachfolgende Instruktion n+1-Mal wiederholt wird. Da der Wiederholungszähler im Statusregister untergebracht ist, das bei einem Interrupt sowieso automatisch gesichert wird, ist diese Instruktionswiederholung jederzeit unterbrechbar.

Dadurch kann z.B. eine allgemeine Shiftoperation so realisiert werden:

```
: rshift  ( n1 cnt -- n2 )   0=EXIT 1 -
times 2/
;
```

Konsequenterweise wäre nun als nächstes z.B. die "Butterfly-Instruktion" fällig, die auch als Autorepeat-Instruktion eine FFT mit einem Butterfly/Zyklus abarbeiten könnte.

Klaus Schleisiek

Approximation für Quadraturpaare

Rafael Deliano

Für die Umrechnung von kartesischen auf Polar-Koordinaten muß für die Zeigerlänge quadriert und dann die Wurzel berechnet werden (Bild 1).

In der Signalverarbeitung wird diese Funktion häufig benötigt; bei DFT bzw. FFT oder auch Modems (Bild 2). Hier ist maximale Genauigkeit meist nicht erforderlich, höhere Geschwindigkeit aber erwünscht. Die direkte Berechnung der Formel wird durch eine Approximation ersetzt, weil die Berechnung von SQRT sequentiell und damit langsam ist.

Die gängige Grundvariante [1] bestimmt erst die Absolutwerte beider Werte und vergleicht dann ihre Grösse. Passend umgeordnet werden dann die Faktoren a, b multipliziert und beide Werte summiert (Bild 3).

Für Festkommarechnung eignen sich Faktoren, die man als Shifts ausführen kann (Tabelle 1). Die krummeren Werte dort kann man aus Shifts und einer Addition bzw. Subtraktion erzeugen.

Mit einem echten Multiplikationsbefehl kann man Konstanten verwenden, die bessere Approximation ermöglichen (Tabelle 2).

Durch Unterteilung in zwei Bereiche erhöht man die Genauigkeit [3] [4] (Bild 4, Tabelle 3, 4).

Plot

Was dabei „optimal“ ist, hängt auch von der Definition des Fehlers ab. Typisch sorgt man dafür, dass die maximale Abweichung minimiert wird. Man kann aber auch auf beste durchschnittliche Abweichung optimieren.

Anschaulicher als eine Kennzahl für Qualität ist ein Plot, normiert auf den Sollwert, den die echte Formel ergibt (Bild 5). Als Laufvariable dient der Winkel, und wegen der Periodizität beschränkt man sich auf einen halben Quadranten.

Integer

Ein Test mit Float berücksichtigt nicht die Rundungsfehler, speziell durch Divison mit Shifts, wie sie dann bei implementierten Routinen (Listing QUAD.F74) auftreten, weshalb man den Test dann besser nochmal mit diesen durchführt (Tabelle 5). Man beachte, dass die Referenzfunktion Y0 hier auch Abweichungen haben kann, weil SQRT-Routinen meist nicht runden.

Hardware

Es gab bereits ein IC [6] das polar/kartesische Wandlungen 16x16 Bit mit 25 MOPS bei einem Fehler von ca. +/-4 LSB durchführte. Dort machte der Hersteller den statistischen Test mit gleichverteilten Zufallszahlen. Wirklich exakte Werte sind also nicht leicht erreichbar. Implementierung mittels Distributed Artithmetic, wie sie auch für FPGAs wieder relevant sein können, finden sich auch [7] in der Literatur.

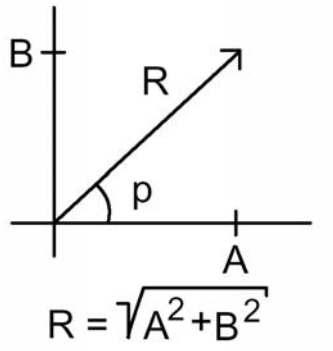


Bild 1 (links):

Wandlung kartesisch auf polar, Formel für die Zeigerlänge

Bild 2 (unten):

Frequenzdetektor für FSK-Demod ulator

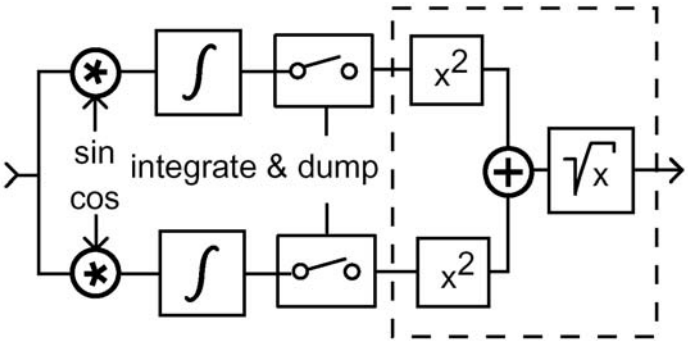


Bild3:

Einfache Approximation...

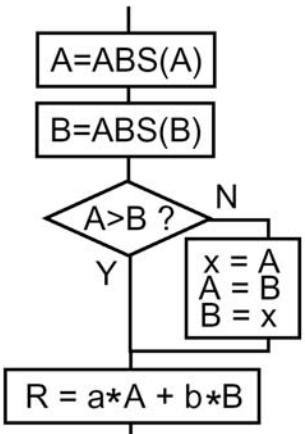


Bild 4:

...in zwei Bereiche unterteilt

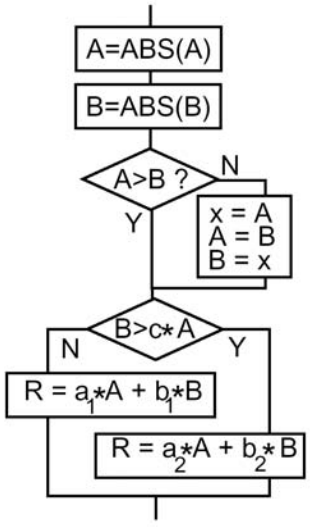


Tabelle 1: „gerade“ Konstanten für Bild 3

a	b		
1	1/2	[1]	y1
1	1/4		y2
1	3/8		y3
7/8	7/16		y4
15/16	15/32		y5

Tabelle 2: „krumme“ Konstanten für Bild 3

a	b		
1	0,2673	[2]	y6
1	0,3	[2]	y7
0,947544	0,392485		y8

Tabelle 3: „gerade“ Konstanten für Bild 4

c	a1	b1	a2	b2		
1/2	1	1/4	3/4	3/4		y9
1/4	1	0	7/8	1/2	[3]	y10
11/29	1	1/8	53/64	37/64		y11
3/8	1	1/8	27/32	9/16		y12

Tabelle 4: „krumme“ Konstanten für Bild 4

c	a1	b1	a2	b2		
1/2	0,986	0,233	0,817	0,586	[3]	y13
0,4142135	0,99	0,197	0,84	0,561	[3]	y14

Tabelle 5: Integer-Testlauf

in		out			
sin	cos	Y0	Y1	Y2	Y6
0000	7FFF	7FFF	7FFF	7FFF	7FFF
0648	7FD8	7FFF	82FC	816A	7FD8
0C8C	7F61	7FFF	85A7	8284	7F61
12C8	7E9C	7FFF	8800	834E	7E9C
18F9	7D89	7FFF	8A05	83C7	7D89
1F1A	7C29	7FFF	8BB6	83EF	7C31
2528	7A7C	7FFF	8D10	83C6	7DC1
2B1F	7884	7FFF	8E13	834B	7F03
30FB	7641	7FFF	8EBE	827F	7FF6
36BA	73B5	7FFF	8F12	8163	809C
3C56	70E2	7FFF	8F0D	7FF7	80F1
41CE	6DC9	7FFF	8EB0	7E3C	80F7
471C	6A6D	7FFF	8DFB	7C34	80AE
4C3F	66CF	7FFF	8CEE	79DE	8015
5133	62F1	7FFF	8B8A	773D	7F2C
55F5	5ED7	7FFF	89D1	7454	7DF7
5A82	5A82	7FFF	87C3	7122	7C73
5ED7	55F5	7FFF	89D1	7454	7DF7

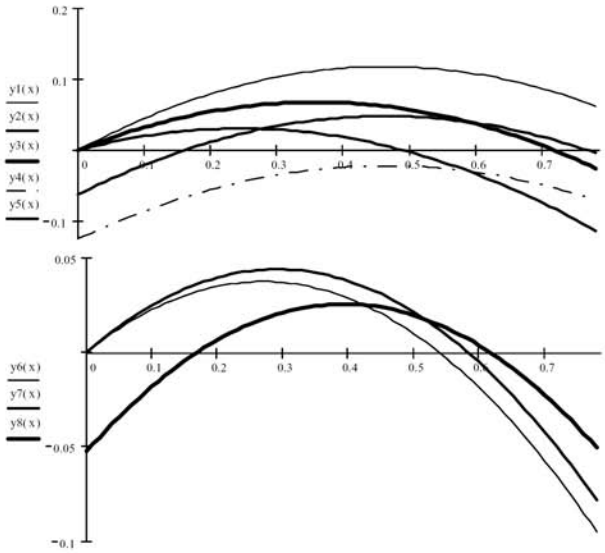
Listing QUAD.F74

```
: SQRT ... ; \ ( UD1 — UN1 ) \ UD1 = 0 - 7FFE0002
: NEGATE NOT 1+ ; \ ( N1 — N2 )
: ABS DUP 8000 AND IF NEGATE THEN ; \ ( N1 — UN1 )
: Y0 \ ( N1 N2 — UN3 ) N1,2 = 8001 ... 7FFF
  ABS DUP U* ROT ABS DUP U* D+ SQRT ;
: Y1 \ ( N1 N2 — UN3 ) N1,2 = 8001 ... 7FFF
  ABS SWAP ABS 2DUP U> IF SWAP THEN
  SWAP 1SHIFT> + ; \ a=1 b=0,5
: Y2 \ ( N1 N2 — UN3 ) N1,2 = 8001 ... 7FFF
  ABS SWAP ABS 2DUP U> IF SWAP THEN
  SWAP 1SHIFT> 1SHIFT> + ; \ a=1 b=0,25
: Y6 \ ( N1 N2 — UN3 ) N1,2 = 8001 ... 7FFF
  ABS SWAP ABS 2DUP U> IF SWAP THEN
  \ ( — min max )
  2DUP 1SHIFT> 1SHIFT> \ ( — min max min max/4 )
  U< IF SWAP DROP \ a=1 b=0
  ELSE DUP 1SHIFT> 1SHIFT> 1SHIFT> - \ a=7/8
  SWAP 1SHIFT> + THEN ; \ b=1/2
```

$x := 0, 0.01 .. \frac{\pi}{4}$ $A(x) := \cos(x)$ $B(x) := \sin(x)$ $r(x) := \sqrt{(A(x))^2 + (B(x))^2}$

$Y(x) := (|A(x)| \; |B(x)|)$ $a1 := 1$ $b1 := \frac{1}{2}$

$r1(x) := a1 \cdot \max(Y(x)) + b1 \cdot \min(Y(x))$ $y1(x) := r1(x) - r(x)$



$x := 0, 0.01 .. \frac{\pi}{4}$ $A(x) := \cos(x)$ $B(x) := \sin(x)$ $r(x) := \sqrt{(A(x))^2 + (B(x))^2}$

$Y(x) := (|A(x)| \; |B(x)|)$ $a1 := 1$ $b1 := \frac{1}{4}$ $a2 := \frac{3}{4}$ $b2 := \frac{3}{4}$ $c := \frac{1}{2}$

$r1(x) := a1 \cdot \max(Y(x)) + b1 \cdot \min(Y(x))$

$r2(x) := a2 \cdot \max(Y(x)) + b2 \cdot \min(Y(x))$

$r3(x) := \text{wenn}(\max(Y(x)) < c \cdot \min(Y(x)), r1(x), r2(x))$ $y9(x) := r3(x) - r(x)$

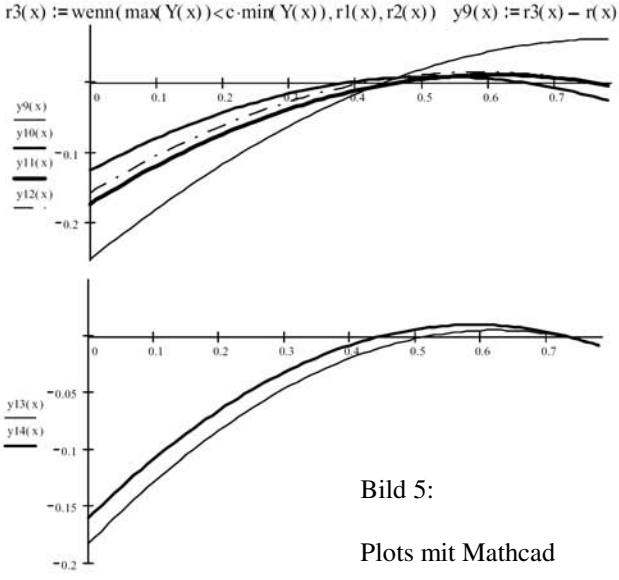


Bild 5:

Plots mit Mathcad

Literatur

[1] Robertson, „A Fast Amplitude Approximation For Quadrature Pairs“ Bell Sys. Tech. J. Oct 1971
[2] Onoe, „Fast Amplitude approximation yielding either exact mean or minimum deviation for quadrature pairs“ Proc. IEEE July 1972
[3] Filip, „Linear approximations to SQRT(x^2+y^2) having equiripple error characteristics“ IEEE Trans. Audio Electroacoustic. Dec 1973
[4] Brady, Adams, „Magnitude Approximations for Microprozessor Implementation“ IEEE micro Oct.1983
[5] Allie, Lyons, „A Root of Less Evil“ IEEE Signal Processing Mag. March 2005
[6] Datenblatt „TMC2330 CMOS Coordinate Transformer“ TRW Inc. 1990
[7] Büttner „Some Improvements in Digital Hardwired Approximation of the Amplitude of Quadrature Pairs“ AEÜ 4/1976

DTMF-Decoder auf Controllern

Rafael Deliano

Die hier beschriebene Variante folgt [1], damals schon auf einem 4 MHz Z80 implementiert.

Eine Filterbank bestimmt dabei die Energie jeder einzelnen Frequenz (Bild 1). Die Teilfilter werden über DFT realisiert (Bild 2). Für jede Einzelfrequenz wird das Eingangssignal in ein Quadratursignal zerlegt und in einer Integrate&Dump-Stufe akkumuliert, in Leistung umgerechnet und an den Klassifikator ausgegeben.

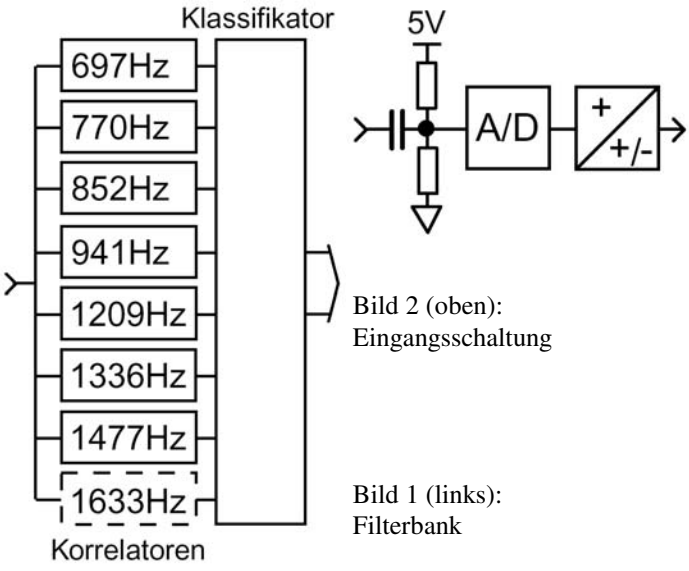


Bild 2 (oben): Eingangsschaltung

Bild 1 (links): Filterbank

Hier wird der 68HC908GP32 mit 2,45 MHz Busfrequenz verwendet.

Frequenzdetektor

Das Eingangssignal muß gleichspannungsfrei sein, hier durch RC-Kopplung und einen Spannungsteiler in Hardware gelöst (Bild 2). Das Signal des A/D-Wandlers ist unipolar. Es muss noch auf 2er-Komplement gewandelt werden.

Die Referenzwerte für Sinus und Cosinus würde man bei konventioneller Implementierung aus Tabellen entnehmen (Bild 4; und Tabellen).

Da wegen der begrenzten Geschwindigkeit des Controllers die Samplerate auf etwa 8kHz beschränkt ist, bestehen die Tabellen nur aus 5 bis 12 Samples. Dementsprechend macht es keinen Sinn, die Amplituden sehr fein zu skalieren. Damit ist eine Implementierung möglich, die Multiplikation in der Integrationsphase eliminiert (Bild 5). Nachteilig ist hier der üppige Bedarf an RAM. Die benötigte Rechenleistung in der Integrationsphase sinkt aber erheblich. Das empfangene Sample muss nur abhängig von der Sinus-Zeigerposition in die Teil-Integratoren summiert werden. Die expliziten Sinus-Datentabellen im ROM entfallen. Nur die durchlaufenden Zeiger bleiben. Identische Koeffizienten kann man auf den gleichen Teil-Integrator legen. Für den Faktor 0 entfällt die Summierung.

Sinus

Man kann auf einem Controller die Abtastfrequenz beliebig wählen und über Timer und Interruptroutine nahezu auf 0,5µsec genau einstellen. Am einfachsten berechnet man abhängig von der Abtastfrequenz, bzw. der Samplerate, die Zahl der Samples in den Sinustabellen für die DTMF-Frequenzen (Tabelle 1). Krumme Zahlen von Samples kann man auf Werte von 0,5 bzw. 0,25 oder 0,75 Samples runden (Tabelle 2). Diese sind durch doppelte bzw. vierfache Tabellenlänge darstellbar und enthalten dann 2 oder 4 Sinuskurvenzüge. Benötigen dann zwar mehr Programmspeicher, aber nicht mehr Rechenzeit.

Nachteilig ist, daß die Zahl der benötigten Amplitudenstufen steigt. Hier wird auf +/-8 Stufen quantisiert; wie sie mit Shifts und Additionen gut darstellbar sind (Tabelle 3). Berechnet man die Tabellen (Tabelle 4) entsprechend, muss man auf Rundungsfehler nachprüfen und korrigieren: die Summe aller Koeffizienten einer Tabelle sollte den Wert 0 ergeben, sonst entsteht ein DC-Bias. Ferner sollten sich die Summen der positiven und negativen Werte der jeweiligen Sinus- und Cosinustabellen entsprechen. Bei Nachbesserung von Hand biegt man auf die von der Berechnung her günstigen Werte 0 bzw. 8.

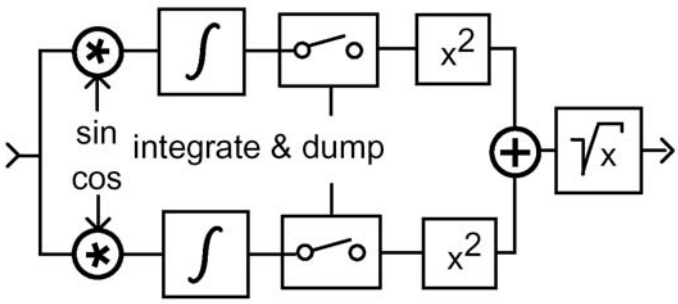
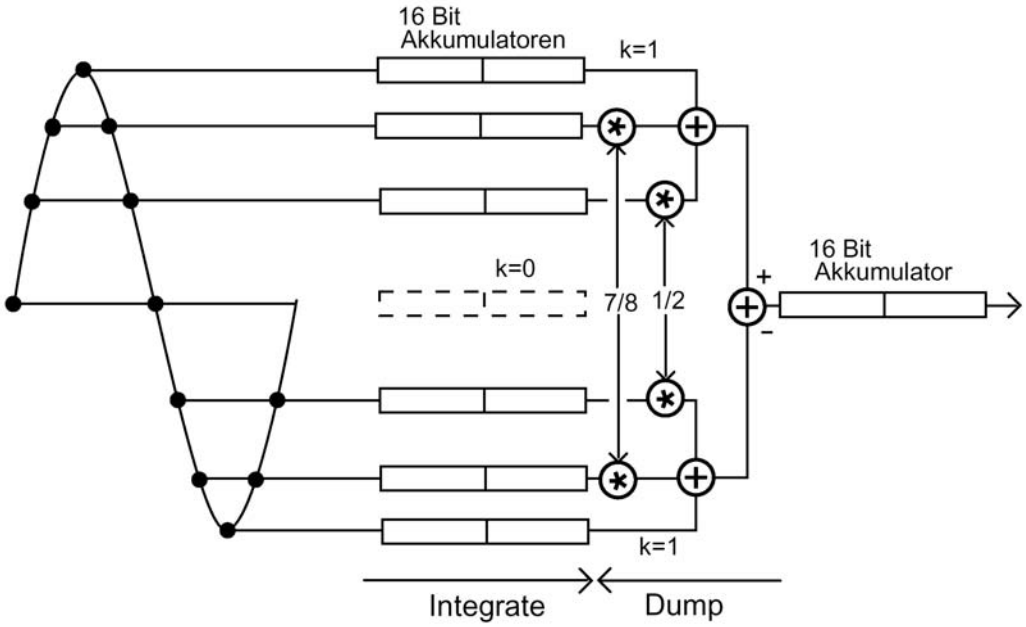


Bild 3: Frequenzdetektor



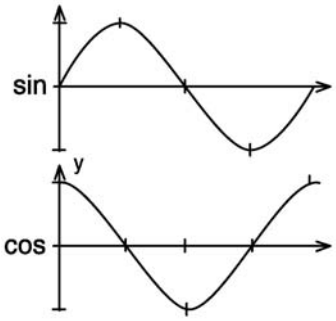


Bild 4 (oben):
Sinus, Cosinus

Bild 5 (links):
Integration

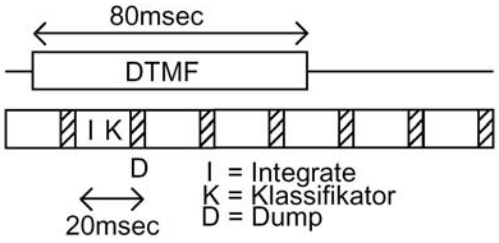


Bild 6.
Timing

Tabelle 1: Rohdaten Sinustabellen

Sample	1633Hz						697Hz		
usec	samples								
150,5	4,0	4,4	4,9	5,4	7,0	7,7	8,6	9,5	
151,0	4,0	4,4	4,9	5,4	7,0	7,7	8,6	9,5	
151,5	4,0	4,4	4,9	5,4	7,0	7,7	8,5	9,4	<-
152,0	4,0	4,4	4,9	5,4	6,9	7,7	8,5	9,4	
152,5	4,0	4,4	4,9	5,4	6,9	7,6	8,5	9,4	
...									
166,0		4,0	4,5	4,9	6,4	7,0	7,8	8,6	
166,5		4,0	4,4	4,9	6,3	7,0	7,8	8,6	
167,0		4,0	4,4	4,9	6,3	7,0	7,7	8,5	<-
167,5		4,0	4,4	4,9	6,3	7,0	7,7	8,5	
168,0		4,0	4,4	4,9	6,3	6,9	7,7	8,5	

Tabelle 2: gewählte Sinustabellen

mit	4,0	4,5	5,0	5,5	7,0	7,75	8,5	9,5	samples
1633Hz		*2		*2		*4	*2	*2	
		9		11		31	17	19	samples
ohne	4,0	4,5	5,0	6,25	7,0	7,75	8,5		samples
1633Hz		*2		*4		*4	*2		
		9		25		31	17		samples

Tabelle 3: durch Shifts darstellbare Amplituden

0	1	2	3	4	5	6	7	8	Amplitude
0	0,125	0,25	0,375	0,5	0,625	0,75	0,875	1,0	Faktor

Tabelle 4: Tabellen in Amplitudenwerten

samples	sin
4	0 8 0 -8
5	0 8 5 -5 -8
7	0 6 8 3 -3 -8 -6
2*4,5	0 8 3 -7 -5 5 7 -3 -8
2*5,5	0 7 6 -2 -8 -4 4 8 2 -6 -7
2*8,5	0 5 8 6 1 -4 -8 -8 -3 3 8 8 4 -1 -6 -8 -5
2*9,5	0 5 8 7 4 -1 -6 -8 -7 -3 3 7 8 6 1 -4 -7 -8 -5
4*6,25	0 4 7 8 7 5 1 -3 -6 -8 -8 -5 -2 2 5 8 8 6 3 -1 -5 -7 -8 -7 -4
4*7,75	0 6 8 5 -1 -6 -8 -5 2 7 8 4 -2 -7 -8 -3 3 8 7 2 -4 -8 -7 -2 5 8 6 1 -5 -8 -6

cos

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

Arithmetik im Galoisfeld

Rafael Deliano

Neben der Anwendung bei fehlerkorrigierenden Codes auch in der Kryptografie und digitalen Signalverarbeitung benötigt. Hier wird mit Blick auf einen BCH-Decoder nur der binäre Fall dargestellt.

Die Elemente im Feld sind also Bits mit Wert 0 und 1.

Praktisch verwendet wird z.B. die Größe GF(2^8); also kompakt genug für tabellengestützte Verfahren. Für Wortlängen wie in der Kryptografie benötig, sind andere Rechenverfahren üblich, als hier dargestellt.

Aus Gründen der Übersichtlichkeit erfolgt hier die Darstellung anhand des kleinen GF(2^4) = GF (16), das in Tabelle 1 gelistet ist. Der Eintrag Null am Anfang ist per Definition vorgegeben. Der Inhalt des Rests der Tabelle wird durch das verwendete primitive Polynom bestimmt und kann automatisch durch einen LFSR-Generator Typ I erzeugt werden, der auf 0001b initialisiert wird (Bild 1). Genau wie in der Funktion des LFSR-Generators wird angenommen, dass das sich das Muster zyklisch wiederholt. Also ist x^15 wieder 0001b (Bild 11).

Man beachte, dass in der Literatur das unterste Bit im Datenwort meist links angeordnet ist. Hier mit Blick auf Software aber rechts. Grundlegene Operationen im Feld sind Addition und Multiplikation. Im Galoisfeld bleibt die Wortlänge immer konstant.

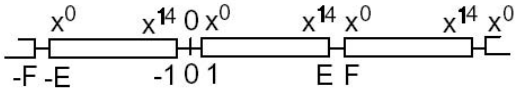


Bild 11 (oben):
Zyklische Wiederholung

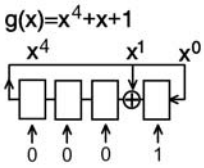
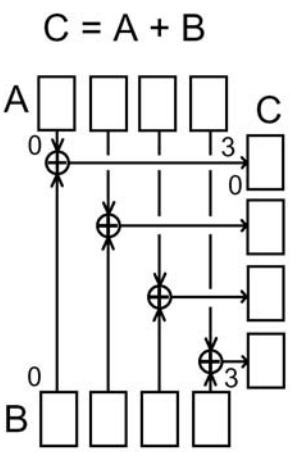


Bild 1 (links):
Generator

Addition



Bei binären Daten entspricht sie der Subtraktion und ist ein bitweises XOR der binären Datenworte (Bild 2). Oft auch als Summierung benötigt (Bild 3). Ist in Software einfach mit dem XOR-Opcode realisierbar.

Bild 2:
Addition und Subtraktion
für ein 4-Bit Wort

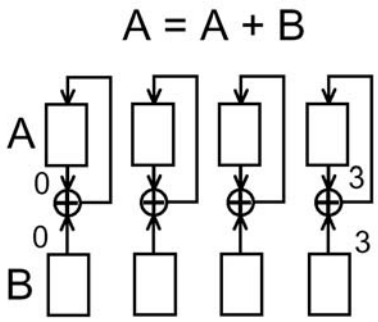


Bild 3:
Summierung für ein
4-Bit Wort

Multiplikation

Für zwei Variablen als Eingangswert gibt es keinen einfachen Al gorithmus. Einen Ausweg bildet die Addition von Logarithmen (Bild 4).

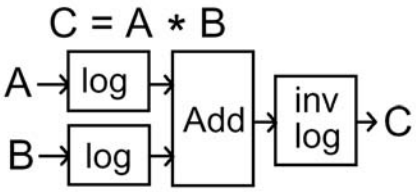


Bild 4:
Multiplikation

Die Zuordnung der Datenworte zu Logarithmen ist in Tabelle 1 mitaufgeführt. Man muss die Daten nur noch geeignet anordnen. Die Routine, die Tabelle 1 erzeugt, kann deshalb auch die Tabellen berechnen. In der Logarithmustabelle (Tabelle 2) ist der erste Eintrag unbenutzt, denn die Eingangswerte A=0 bzw. B=0 werden separat ausgewertet (Bild 5).

Nach der Addition hat der Zeiger in die Tabelle für den inversen Logarithmus (Tabelle 3) den Wert 0 – 1C. Praktisch ist die Tabelle nur für die Werte 0 – 0F ausgeführt. Deshalb muss man für größere Werte den Zeiger so verbiegen, dass er wieder in die Tabelle zeigt. Diese Funktion wird hier als SCALE bezeichnet.

Tabelle 1: GF(2^4) aus Polynom x^4 + x + 1

Poly.	Exponent	Logarithmus/Vektor
0000	0	-
0001	x^0	0
0010	x^1	1
0100	x^2	2
1000	x^3	3
0011	x^4	4
0110	x^5	5
1100	x^6	6
1011	x^7	7
0101	x^8	8
1010	x^9	9
0111	x^10	10
1110	x^11	11
1111	x^12	12
1101	x^13	13
1001	x^14	14
0001	x^15	-
0010	x^16	-
0100	x^17	-

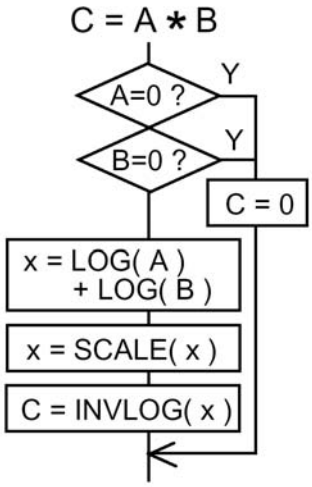


Bild 5:
Flußdiagramm Multiplikation

Tabelle 2: LOG-Tabelle

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
FF	00	01	04	02	08	05	0A	03	0E	09	07	06	0D	0B	0C

Tabelle 3: INVLOG-Tabelle

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
01	02	04	08	03	06	0C	0B	05	0A	07	0E	0F	0D	09	01

Multiplikation einer Konstanten

Häufig benötigt und in Hardware relativ einfach zu realisieren (Bild 6). Aber meist sind es nicht so wenig XOR Gates, dass man so mit Bitbefehlen programmieren kann. In Software ist eine Tabelle schneller. Die kann man bequem mit dem Multiplikationsbefehl aus Bild 4 füllen.

Von praktischer Bedeutung ist auch die Variante, wo der Ausgang ins Eingangsregister zurücksummiert wird. Das kann zu einer etwas undurchsichtigen Darstellung führen (Bild 7).

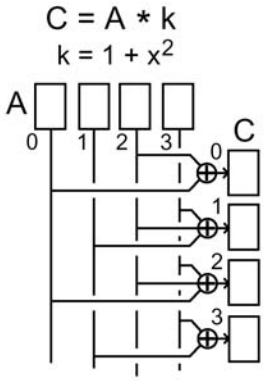


Bild 6 (links):
Multiplikation mit Konstante

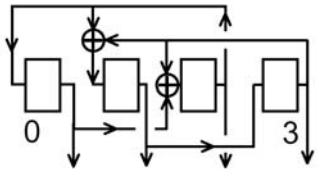


Bild 7 (oben):
Multiplikation mit Konstante und Summierung

Division

Entspricht der Multiplikation, nur dass der Divisor subtrahiert wird (Bild 8). Das Resultat hat danach den Wert –D...+D, und die negativen Werte müssen so gepatcht werden, dass sie in die Tabelle zeigen.

Scale

Einige Controller, wie der 68HC08 haben schnelle, vorzeichenlose Divisionsbefehle 16/8 mit 8 Bit Ergebnis und 8 Bit Rest. Bei Division durch 0F gibt dieser Rest den gewünschten skalierten Zeiger in die Tabelle. Negative Zahlen kann man durch Addition eines Offsets, z.B. 8 * F = 78h, in eine vorzeichenlose Zahl wandeln und damit an die Division anpassen.

Zech Logarithmen

In Bild 4 wechselt man vom Polynom-Zahlenformat mit den Werten 0 - E in die Exponentialdarstellung, die auch als Vektor-Format bezeichnet wird, mit den Werten 1 - F . Dem Polynom-Wert 00h entspräche dort der Wert „Minus-Unendlich“, also für numerische Rechnung unbrauchbar.

Addition ist jedoch mittels des Zech Logarithmus (Tabelle 4, 5) auch im Vektor-Format möglich. Dafür ist allerdings eine weitere Tabelle nötig, deren Erzeugung Bild 10 zeigt.

Für die Berechnung (Bild 9) wird die Differenz von A - B berechnet, weshalb man beide gegebenenfalls tauscht, damit nach der Subtraktion eine positive Zahl herauskommt. Da die ZLOG-Tabelle für A=B und damit Differenz = 0 nicht verwendbar ist, wird dieser Fall aussortiert.

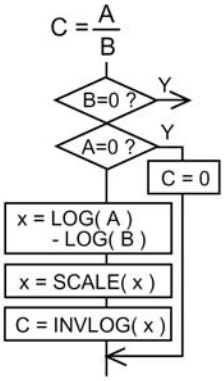


Bild 8:
Flußdiagramm Division

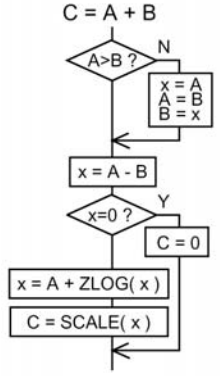


Bild 9:
Flußdiagramm Addition / Zech

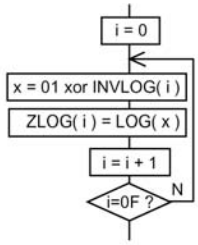


Bild 10 (oben):
Flußdiagramm ZLOG erzeugen

Tabelle 4: Zech Logarithmus

Vektor	Zech
0	-
1	4
2	8
3	14
4	1
5	10
6	13
7	9
8	2
8	7
10	5
11	12
12	11
13	6
14	3

In vielen Anwendungen versucht man, komplett im Vektor-Format zu rechnen, weil z.B. Petenzieren dort einfach möglich ist

$$(x^6)^3 = x^{6*3}$$

Tabelle 5 : ZLOG-Tabelle

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
FF	04	08	0E	01	0A	0D	09	02	07	05	0C	0B	06	03	FF

Test

GF(16) hat den Vorteil, dass man die Rechnung noch von Hand, z.B. anhand der Tabellen 6, 7 kontrollieren kann. Wenn man geeignet implementiert, kann man durch Änderung von Konstanten auf GF(256) skalieren, ohne dort nochmal detailliert testen zu müssen.

Tabelle 7: Multipliktion Zahlenformat Polynom

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
01	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
02	00	02	04	06	08	0A	0C	0E	03	01	07	05	0B	09	0F	0D
03	00	03	06	05	0C	0F	0A	09	0B	08	0D	0E	07	04	01	02
04	00	04	08	0C	03	07	0B	0F	06	02	0E	0A	05	01	0D	09
05	00	05	0A	0F	07	02	0D	08	0E	0B	04	01	09	0C	03	06
06	00	06	0C	0A	0B	0D	07	01	05	03	09	0F	0E	08	02	04
07	00	07	0E	09	0F	08	01	06	0D	0A	03	04	02	05	0C	0B
08	00	08	03	0B	06	0E	05	0D	0C	04	0F	07	0A	02	09	01
09	00	09	01	08	02	0B	03	0A	04	0D	05	0C	06	0F	07	0E
0A	00	0A	07	0D	0E	04	09	03	0F	05	08	02	01	0B	06	0C
0B	00	0B	05	0E	0A	01	0F	04	07	0C	02	09	0D	06	08	03
0C	00	0C	0B	07	05	09	0E	02	0A	06	01	0D	0F	03	04	08
0D	00	0D	09	04	01	0C	08	05	02	0F	0B	06	03	0E	0A	07
0E	00	0E	0F	01	0D	03	02	0C	09	07	06	08	04	0A	0B	05
0F	00	0F	0D	02	09	06	04	0B	01	0E	0C	03	08	07	05	0A

Tabelle 6: Addition Zahlenformat Polynom

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
01	01	00	03	02	05	04	07	06	09	08	0B	0A	0D	0C	0F	0E
02	02	03	00	01	06	07	04	05	0A	0B	08	09	0E	0F	0C	0D
03	03	02	01	00	07	06	05	04	0B	0A	09	08	0F	0E	0D	0C
04	04	05	06	07	00	01	02	03	0C	0D	0E	0F	08	09	0A	0B
05	05	04	07	06	01	00	03	02	0D	0C	0F	0E	09	08	0B	0A
06	06	07	04	05	02	03	00	01	0E	0F	0C	0D	0A	0B	08	09
07	07	06	05	04	03	02	01	00	0F	0E	0D	0C	0B	0A	09	08
08	08	09	0A	0B	0C	0D	0E	0F	00	01	02	03	04	05	06	07
09	09	08	0B	0A	0D	0C	0F	0E	01	00	03	02	05	04	07	06
0A	0A	0B	08	09	0E	0F	0C	0D	02	03	00	01	06	07	04	05
0B	0B	0A	09	08	0F	0E	0D	0C	03	02	01	00	07	06	05	04
0C	0C	0D	0E	0F	08	09	0A	0B	04	05	06	07	00	01	02	03
0D	0D	0C	0F	0E	09	08	0B	0A	05	04	07	06	01	00	03	02
0E	0E	0F	0C	0D	0A	0B	08	09	06	07	04	05	02	03	00	01
0F	0F	0E	0D	0C	0B	0A	09	08	07	06	05	04	03	02	01	00

Compact Flash an Controllern

Rafael Deliano

Der ideale Weg um einen Einplatinencomputer mit so etwas wie einer Floppy auszustatten, d.h. einen wechselbaren Speicher mit dem man Files von und zu PCs übertragen kann.

Die Vorzüge gegenüber einer Floppy sind erheblich. Die Schnittstelle passt direkt an den Bus einer 8-Bit CPU. Die Bauform ist klein. Es ist sowohl 3,3V als auch 5V Versorgung möglich. Stromaufnahme ist mit ca. 50 mA erträglich.

Die beiden Varianten CF I und CF II unterscheiden sich nur in der Bauhöhe. Die dünnere CF I paßt auch in CF II Stecker. 1994 mit 4-Mbyte Karten eingeführt, denen 1996 die heute übliche kleinste Grösse 8 Mbyte folgte. Übliche Speicherdichten für kleine Varianten sind 8, 16, 32 MByte.

Hardware

Wegen des benötigten RAMs ist ein Einplatinencomputer sinnvoller als ein Einchip-Controller. Die CF-Karte liegt dann wie ein Peripheriebaustein im Speicher. Wegen der Zugriffszeit von 150 nsec darf der Bus nicht zu schnell sein.

Damit „hot plugging“ möglich ist, muss er über Treiber, die man tristate schalten kann, abgetrennt sein (Bild 1). Das verbessert auch den ESD-Schutz etwas. Die Treiber sind zudem nötig, weil an der Schnittstelle nicht TTL sondern CMOS-Pegel gefordert sind. Die Beschaltung dekodiert 16 Register. Benötigt werden typisch aber nur 8. Also könnte man A3 auch auf GND legen. Neben der Busanschaltung sind auch noch einige Portpins erforderlich.

Mit /EN schaltet die CPU die Versorgung des Speichers an. Über den Pin RESET kann er den Speicher rücksetzen. Die Pulsbreite muss größer 10 µsec sein. Es kann im Betrieb 100 µsec dauern, bis die Karte wieder bereit ist. Unmittelbar nach Powerup kann es auch 400 µsec dauern. Typische Zeiten schwanken von 5 - 250 µsec.

Verfügbar, aber derzeit noch teuer, sind auch wesentlich grössere Ausführungen. Da ursprünglich für Laptops gedacht, passt der 50pol Steckverbinder über einen passiven Adapter in PCMCIA-Karten. Neben dieser Anschaltung ist alternativ ‘True DIE‘ möglich. In diesem Fall emuliert man genauso direkt eine Harddisk. Hier ist aber ‘hot plugging‘, also Stecken und Entfernen der Karte bei laufendem Gerät, nicht vorgesehen. Auf diese wünschenswerte Eigenschaft will man aber ungern verzichten.

Die dritte Beschaltung ‘Common Memory‘ ist für Controller besonders geeignet. Man hängt direkt memory mapped am Bus der CPU.

Zwar haben die Karten nominell eine interne Resetschaltung, aber normalerweise wird man bei Anlegen der Versorgung RESET=1 schalten und erst nach Verzögerung mit RESET=0 freigeben. Wenn man den Pin nicht ansteuern will, verbindet man ihn fest mit GND.

Der RDY/BSY-Pin zeigt an, wann die Karte den Reset beendet hat und bereit ist. Der Pin muss nicht zwingend auf Port gelegt sein. Man kann das Bit auch in einem Register abfragen.

/CD1 ist im Speicher mit GND verbunden. Damit prüft die CPU, ob eine Karte steckt. Er wird dann die Versorgung anschalten und damit auch die Treiber freigeben.

Schaltung

Der hier verwendete Mitsubishi-6502 führt jeden Schreibzugriff als read-modifywrite Zyklus aus. Das macht bei Peripheriebausteinen fast immer Probleme. Deshalb muss man Schreiben und Lesen in unterschiedliche Speicherbereiche legen und die Dekodierung entsprechend ausführen (Bild 2).

Die beiden Ports sind hier als 74HC-Bausteine ausgeführt. Beim Register 74HC175 wird durch den Reset /RES der CPU

die Schnittstelle inaktiv geschaltet. Da die Stromaufnahme der Karte typisch nur etwa 50 mA beträgt, genügt ein Wald&Wiesen-Transistor als Schalter. Echte Harddisks in CF-Bauform könnten jedoch bis 500 mA benötigen und werden hier nicht berücksichtigt. Das rote LED zeigt dem Benutzer an, dass die Karte unter Spannung steht und nicht gezogen werden darf.

Die Pullups an 74HC244 und 74HC245 werden gebraucht, damit sie bei gezogener Karte definierte Pegel am Eingang haben. In der Karte haben viele Pins Pullups und müssen deshalb am Stecker nicht zwingend mit 5V verbunden werden. Sie sind in Tabelle 1 mit „r“ gekennzeichnet. Dort ist links der offizielle Name sowie Signaltyp und rechts das in der Beschaltung angelegte Signal vermerkt.

Sockel

Die üblichen CF-Sockel mit ihren 0,64 mm SMD-Kontakten sind für Breadboards ungünstig. Von den PCMCIA-Sockeln gibt es eine Variante, die in Laptops huckepack über dem unteren Sockel auf der Leiterplatte montiert wird (Bild 3). Dessen bedrahtete Pins haben dann 2,54 mm Raster. Es wird allerdings zusätzlich ein handelsüblicher PCMCIA auf CF Adapter benötigt. Dessen modifizierte Pinbelegung ist im Anhang der CF-Spec angegeben [1] (Tabelle 2).

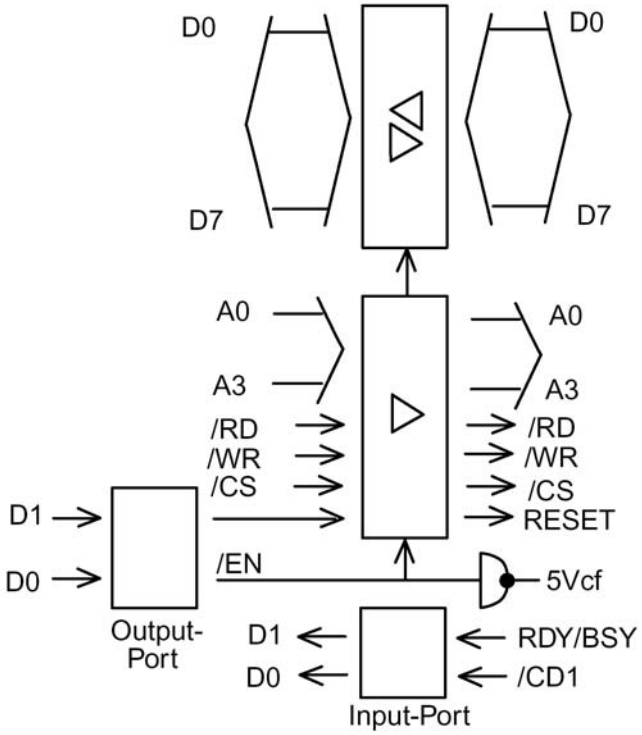
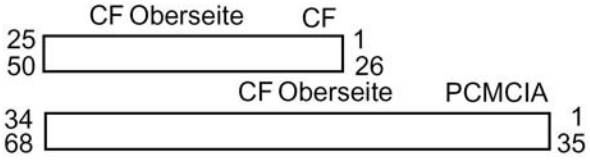


Bild 1 (oben): Blockschaltbild

Bild 3 (unten): Nummerierung der Pins,
Blick auf die Rückseite der Stecker



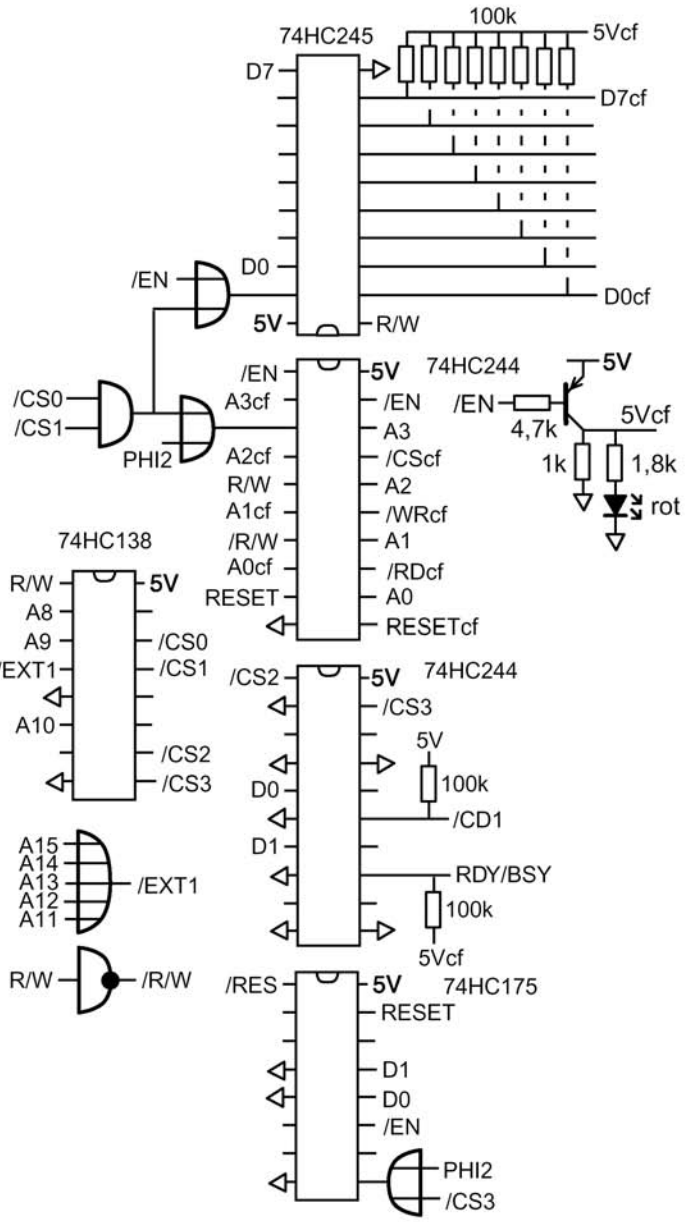


Bild 2 (oben): Schaltung

Bild 4 (unten): CF in PCMCIA-Sockel

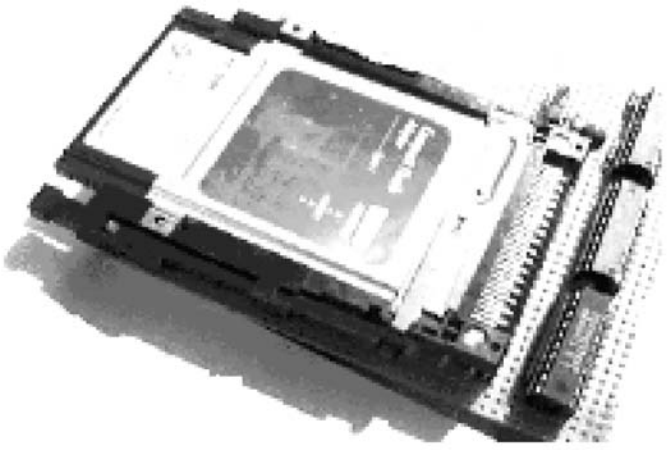


Tabelle 1: Pins der CF-Karte in Memory Mode, sowie deren Beschaltung

1	GND		GND	26	/CD1	O	/CD1
2	D3	I/O	D3cf	27	D11	I/O	nc
3	D4	I/O	D4cf	28	D12	I/O	nc
4	D5	I/O	D5cf	29	D13	I/O	nc
5	D6	I/O	D6cf	30	D14	I/O	nc
6	D7	I/O	D7cf	31	D15	I/O	nc
7	/CE1	I r	/CScf	32	/CE2	I r	5Vcf
8	A10	I	GND	33	/VS1	O	nc
9	/OE	I r	/RDcf	34	/IORD	I r	nc
10	A9	I	GND	35	/IOWR	I r	nc
11	A8	I	GND	36	/WE	I r	/WRcf
12	A7	I	GND	37	RDY/BSY	O	RDY/BSY
13	Vcc		5Vcf	38	Vcc		5Vcf
14	A6	I	GND	39	/CSEL	I	GND
15	A5	I	GND	40	/VS2	O	nc
16	A4	I	GND	41	RESET	I r	RESETcf
17	A3	I	A3cf	42	/WAIT	O	nc
18	A2	I	A2cf	43	/INPACK	O	nc
19	A1	I	A1cf	44	/REG	I r	5V cf
20	A0	I	A0cf	45	BVD2	I/O	nc
21	D0	I/O	D0cf	46	BVD1	I/O	nc
22	D1	I/O	D1cf	47	D8	I/O	nc
23	D2	I/O	D2cf	48	D9	I/O	nc
24	WP	O	nc	49	D10	I/O	nc
25	/CD2	O	nc	50	GND		GND

Tabelle 2: Adapter PCMCIA auf CF

PCM	CF		PCM	CF		PCM	CF	
1	1	GND	29	20	A0	57	40	/VS2
2	2	D3	30	21	D0	58	41	RESET
3	3	D4	31	22	D1	59	42	/WAIT
4	4	D5	32	23	D2	60	43	/INPK
5	5	D6	33	24	WP	61	44	/REG
6	6	D7	34	50	GND	62	45	BVD2
7	7	/CE1	35	1	GND	63	46	BVD1
8	8	A10	36	26	/CD1	64	47	D8
9	9	/OE	37	27	D11	65	48	D9
10	-		38	28	D12	66	49	D10
11	10	A9	39	29	D13	67	25	/CD2
12	11	A8	40	30	D14	68	50	GND
13	-		41	31	D15			
14	-		42	32	/CE2			
15	36	/WE	43	33	/VS1			
16	37	RDY/BSY	44	34	/IORD			
17	13	Vcc	45	35	/IOWR			
18	-		46	-				
19	-		47	-				
20	-		48	-				
21	-		49	-				
22	12	A7	50	-				
23	14	A6	51	38	Vcc			
24	15	A5	52	-				
25	16	A4	53	-				
26	17	A3	54	-				
27	18	A2	55	-				
28	19	A1	56	39	/CSEL			

Zugriff

Spätestens in Software hat man das modifizierte ATA-Interface vor sich. Studium von ATA-2 [3] als Vergleich ist empfehlenswert. Die „Common Memory"-Betriebsart ist letztlich eine Untermenge davon. Es sind zwar mehr Register vorhanden, praktisch genügen aber die ersten 8 Register des „task files" (Tabelle 3).

Der eigentliche Speicher ist fest in Sektoren zu 512 Byte eingeteilt. Diese Daten kommen sequentiell per Protokoll durch das DATA Register. Im echten ATA-Interface ist der Datenbus 16 Bit breit. Bei dem hier vorliegenden 8-Bit Interface kommen die Datenworte little-endian, was für den 6502 gut passt, aber bei big-endian Controllern zu berücksichtigen ist.

Wenn man beim Zugriff nicht nur einen, sondern mehrere Sektoren übertragen will, stellt man das vorher in SECTOR-COUNT ein:

```
00h  256 Sektoren
01h    1 Sektor
02h    2 Sektoren
...
```

Hier wird immer nur 1 Sektor übertragen, also Einstellung 01h.

In den Registern x3 bis x6 wird die Adresse des ersten Sektors vorgegeben. Es gibt dafür zwei Varianten:

CHS („Cylinder, Head, Sector") und
LBA („Linear Block Adressing").

Hier wird die einfachere LBA-Adressierung verwendet. Sie ist simpel ein 28-Bit Adreßwort, das so über die Register verteilt ist:

```
                      bit
SECTOR-NUMBER=   7 -   0
CYLINDER-LO    = 15 -   8
CYLINDER-HI    = 23 - 16
DRIVE/HEAD     = 27 - 24
```

Die obersten 4 Bit von DRIVE/HEAD müssen fest auf 1110b eingestellt werden. Einschalten der Versorgungsspannung erfolgt hier (Listing CF1.F74) mit UP , Abschalten mit DOWN. Mit dem Befehl *TF.* kann man die Register auslesen. Das Ergebnis sieht ca. so aus:

```
 0  1  2  3  4  5  6  7
00 01 01 01 00 00 E0 50
```

Wichtig ist der Wert 50 h im STATUS-Register. Er zeigt an, dass die Karte bereit für Befehle ist (Tabelle 4). Das ERROR-Register kann man als Erweiterung von STATUS ansehen. Im normalen Betrieb ist für Zugriff das BUSY-Flag wichtig, sowie das Flag DRQ, das anzeigt, wann ein Datenbyte gelesen bzw. geschrieben werden kann.

Wenn Zugriff möglich ist, lädt man in Register x1 - x6 die passenden Einstellung. Soweit die LBA-Adresse nicht komplett erforderlich ist, muss man nach Reset zumindest die oberen 4 Bit von DRIVE/HEAD initialisieren. Dann schreibt man den Befehls-Token in das COMMAND-Register, wodurch der Befehl ausgeführt wird.

Von den 40 Befehlen des Standards sind normalerweise in der

CF-Karte nur 30 implementiert; in der Ansteuerungssoftware hier auf 4 Befehle reduziert. Die Übergabe der LBA-Adressen erfolgt dabei nicht über Stack, sondern die 32-Bit Variable LBA . Wenn man sich auf 32-MB Karten beschränkt, sind nur die unteren 16 Bit nötig und alles oberhalb statisch 000h.

Tabelle 3: Register „task-file“

Adr	Read	Write
x0	DATA	
x1	ERROR	FEATURES
x2	SECTOR-COUNT	
x3	SECTOR-NUMBER	
x4	CYLINDER-LO	
x5	CYLINDER-HI	
x6	DRIVE/HEAD	
x7	STATUS	COMMAND

Tabelle 4: STATUS-Register, ERROR-Register

	Bit	
STATUS	7	1 = BUSY
	6	1 = RDY Ready (wait for RDY=1 after Reset)
	5	1 = DWF Write Fault
	4	1 = DSC Ready
	3	1 = DRQ Data Request
	2	1 = CORR Correctable Error was corrected
	0	1 = ERR Error in Register ERROR
ERROR	7	1 = Bad Block
	6	1 = Uncorrectable Error
	4	1 = ID error
	2	1 = Abort , invalid command
	0	1 = General Error

Die Variable >CACHE zeigt auf den Beginn des Speichers im RAM des Controllers von dem der Sektor gelesen, bzw. wohin der Sektor geschrieben wird. Ihr unteres Byte ist immer 00h.

Daten lesen

READ-SECTOR \ (—)
\ opcode 20

Die LBA-Variablen entsprechend der Adresse des ersten Sektors werden in die Register x3 bis x6 der Karte kopiert. In SECTOR-COUNT speichert man die Zahl der Sektoren, die man lesen will. Hier soll immer nur ein Sektor übertragen werden, also 01h. Dann schreibt man den Befehlscode 20h und wartet darauf, dass in STATUS das BSY-Flag gelöscht und das DRQFlag gesetzt ist. Während des Lesens der folgenden 512 Byte aus DATA, müssen die Flags in STATUS nicht geprüft werden.

Daten schreiben

WRITE-SECTORS \ (—)
\ opcode 30

WRITE-SECTORS funktioniert wie READ-SECTORS mit geänderter Richtung der Datenbewegung. Das Schreiben der Bytes einzelner Sektoren erfolgt ungebremst, weil diese im RAM der Karte gepuffert werden. Da das Schreiben von Flash aber dauert, kann dann eine Pause von 700 µsec („Class 2 Befehl“) auftreten, bis das DRQ-Flag wieder gesetzt wird. Hier wird explizit gewartet, bis die Karte wieder bereit ist. Sonst könnte man z.B. mit DOWN irrtümlich die Versorgung abschalten, während die Karte noch am speichern ist.

Einstellungen lesen

IDENTIFY-DRIVE \ (—)
\ opcode EC

Ähnlich wie mit dem Read-Befehl werden 512d Byte Einstellungen und Herstellerinformationen aus der Karte ausgelesen (Tabelle 5). Speziell für Formatierung kann man so die Speichergrösse bestimmen.

IDENTIFY-DRIVE.\ (—)
Der zugehörige Druckbefehl (Tabelle 6) am Beispiel einer 8-MB Karte.

Standby

SLEEP \ (—) opcode 99
Reduziert den Stromverbrauch auf < 1 mA. Reaktivierung erfolgt durch beliebigen Befehl. Da echte Harddisks Stromfresser sind, hatte ATA diverse Befehle für Standby definiert. Weil sich CF-Karten automatisch recht schnell runterschalten, sind solche Befehle hier fast nicht nötig. Abschalten der Versorgung ist wegen der langsamen Einschaltzeit vieler Karten aber auch unattraktiv.

Einschränkungen

Die Schaltung funktioniert nicht für alle Karten. Ein Exemplar Casio 8MB mit Hitachi-Innereien war nicht ansprechbar. Funktioniert aber an Kartenleser am PC. Erklärung findet sich eventuell in Angaben im Hitachi-Handbuch zur Behandlung des /OE-Pins beim Powerup. Der Aufwand auch pathologische ältere Karten zu berücksichtigen, scheint aber nicht gerechtfertigt.

[1] www.compactflash.org
„CF+ and CompactFlash Specification Revision 2.0“

[2] www.sandisk.com
„Compact Flash Memory Card Product Manual“
„Using SanDisk Flash ATA Components with an 8051 Microcontroller“

[3] www.t13.org
„Working Draft X3T10/0948D“
ATA-2

[4] www.renesas.com
„Hitachi Flash Cards User’s Manual“
„Q&A on Hitachi CF and ATA Cards“

Tabelle 5: Identify Drive

Wrd Adr	Default Data	Bytes	
0	848Ah	2	general configuration bits
1	xxxx	2	default number of cylinders
2	0000	2	reserved
3	xxxx	2	default number of heads
4	0000	2	unformatted bytes per track (obsolete)
5	0240	2	unformatted bytes per sector (obsolete)
6	xxxx	2	default number of sectors per track
7	xxxx	4	MSW number of sectors per card
8			LSW
9	0000	2	reserved
10	aaaa	20 d	serial number in ASCII right justified
20	0002	2	buffer type = dual ported (obsolete)
21	0002	2	buffer size in 512 byte increments: 1k (obsolete)
22	0004	2	# of ECC bytes passed on R/W long
23	aaaa	8	firmware revision (big endian)
27	aaaa	40 d	model number (big endian)
47	0001	2	maximum of 1 sector on R/W multiple
48	0000	2	double word not supported (obsolete)
49	0200	2	DMA not supported , LBA not supported
50	0000	2	reserved
51	0200	2	PIO data transfer cycle timing mode
52	0000	2	DMA data transfer cycle time. not supported (obsolete)
53	0003	2	field validity
54	0003	2	current number of cylinders
55	xxxx	2	current number of heads
56	xxxx	2	current sectors per track
57	xxxx	4	LSW current capacity in sctors
58			MSW
59	010xh	2	multiple sector setting is valid
60	xxxx	4	total number of sectors addressable
61			in LBAmode
62	xxxx	4	reserved
63			
64	0003	2	advanced PIO modes supported
65	0000	4	reserved
66			
67	0078	2	minimum PIO transfer without flow control
68	0078	2	minimum PIO transfer with IORDY flow control
69	0000	130d	reserved
128	0000	64d	reserved vendor unique bytes
160	0000	192d	reserved

Tabelle 6: Test IDENTIFY-DRIVE

—— —— —— | UP \ switch power on
—— —— —— | IDENTIFY-DRIVE \ read data
—— —— —— | IDENTNTIFY-DRIVE. \ print data

general configuration bits	n48A
default number of cylinders	00F6
default number of heads	0002
number of unformatted bytes per track	0000
number of unformatted bytes per sector	0200
default number of sectors per track	0020
number of sectors per card	0000 3D80
serial number	X0102
20030724025939	
buffer type = dual ported	= 0002 0002
buffer size in 512 byte increments	0002
# of ECC bytes passed on R/W long	0004
firmware revision	Rev 2.00
model number	
	Hitachi XXM2.2.0
maximum of 1 sector on R/W multiple	0001
double word not supported	= 0000 0000
DMA not supported , LBA not supported	= 0200 0200
PIO data transfer cycle timing mode	= 0200 0100
DMA data transfer cycle tim. not supported	= 0 0000
field validity	0001
current number of cylinders	00F6
current number of heads	0002
current sectors per track	0020
current capacity in sectors	0000 3D80
multiple sector setting is valid	= 010x 0100
total number of sectors addressable in LBAmode	0000 3D80
advanced PIO modes supported	= 0003 0000
minimum PIO transfer without flow control	= 0078 0000
minimum PIO transfer with IORDY flow cntrl	0078 0000

—— —— —— | DOWN \ power down

ComapctFlash Fileformat

Rafael Deliano

Soweit man Files von und zu PCs übertragen will, muss man das in MS-DOS übliche Format verwenden.

Ein universeller Treiber für Speicherkarten beliebiger Grösse wird aber umfangreich. Beschränkt man sich auf 8-, 16-, 32-MByte Karten, entfallen schon mal die Partitions. Erst mit denen kam MS-DOS 3.x dann über 32 MByte. Wegen der bei 10 MByte üblichen Grenze zwischen FAT12 und FAT16 sind diese beiden Varianten von Interesse, aber FAT32 entfällt. Soweit man die Karten selbst formatiert, kann man auch FAT12 eliminieren, da man in dem Speicherbereich auch nur mit FAT16 arbeiten kann. Jedoch wird man ohne FAT12 viele fremdformatierte Karten nicht lesen können. Eine weitere Vereinfachung ist, auf Directories zu verzichten.

Hardwareschnittstelle

Der Speicher besteht aus 512-Byte Sektoren, die über Register im task-file adressiert werden. Die alte CHS-Adressierung (Cylinder, Head, Sectors) findet sich zwar in Beschreibungen des Filesystems, und die Software im PC wird auch noch so

rechnen. Letztlich ist das einfachere LBA-Format aber technisch äquivalent. Man beachte jedoch, dass der Speicher mit CHS (0,0,1) beginnt, während man alternativ erwartungsgemäß mit LBA (0) anfängt. Die Zahl der verfügbaren LBA-Sektoren, also die Grösse des Speichers, wird mit dem „Identify Drive"-Befehl aus der Karte gelesen. Nur für Neuformatierung der Karte nötig.

Bereiche

Alle Strukturen oberhalb der Sektorgrösse 512 Byte in Software definiert und gehen auf MS-DOS zurück. Datenworte werden deshalb little-endian gespeichert, wie für 8086 Prozessoren üblich. Bei der Formatierung werden die 4 Teilbereiche des Filesystems passend initialisiert (Bild 1).

Bild 1: Bereiche

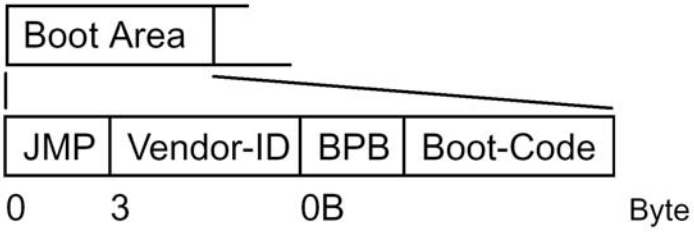


Bild 2: Bootsektor

Tabelle 1: Bios Parameter Block

Offset	Byte	
11d	2	BytePerSec
13	1	SecPerClus
14	2	RsvedSecCnt
16	1	NumFATs
17	2	RootEntCnt
19	2	TotSec16
21	1	Media
22	2	FATSz16
24	2	SecPerTrk
26	2	NumHeads
28	2	HiddSec
32	4	TotSec32
36	1	DrvNum
37	1	Reserved1
38	1	BootSig
39	4	VolID
43	11	VolLab
54	8	FilSysType

Cluster

Die direkte Adressierung einzelner 512 Byte Sektoren im „Data Area" wäre zu ineffizient, die Speicherblöcke dort müssen grösser sein. Es werden dafür mehrere Sektoren zu einem 'Cluster' zusammengefaßt. Wie groß eine solche „allocation unit" ist, wird bei der Formatierung festgelegt.

Tabelle 2: Beschreibung zu BPB

BytePerSec	Immer 512d Bytes/Sektor
SecPerClus	Sektoren/Cluster können 1, 2, 4, 8, 16d, 32d, 64d sein.
RsvedSecCnt	In FAT16 und FAT32 immer 0001h
NumFATs	Immer 02h, also zwei FATs
RootEntCnt	Maximal mögliche Zahl der Files im File Directory. Da jeder Eintrag 32 Byte hat und man Sektoren voll belegen will, also minimal 16d und Vielfache davon. Die Zahl ist für FAT16 vorzugsweise 512d.
TotSec16	16 Bit Zahl der Sektoren des Speichers. Inclusive Boot Area, FATs, File Directory.
Media	Media Descriptor F8h für Harddisks, fest installierte Speicher und CF. F0h für moderne Floppies und wechselbare Speicher. Legal sind auch F9 ... FF, die hauptsächlich historische Floppies spezifizierten.
FATSz16	Anzahl der Sektoren in einer (der beiden) FATs
SecPerTrk	Sektoren/Track
NumHeads	Zahl der Heads
HiddSec	00h, da keine Partitions
TotSec32	00000000h 32-Bit Zahl der Sektoren des Speichers.
DrvNum	Drive Number. Z.B. 80h für harddisk, 00h für Floppies verwenden.
Reserved1	00h
BootSig	Extended Boot Signature Byte zeigt an, das die nächsten 3 Einträge existieren: 00h zeigt an, das sie nicht existieren.
VolID	Die Volume-ID ist eine 32 Bit Zufallszahl. Meist dadurch erzeugt, dass man die 16-Bit Worte für Date und Time reinschreibt.
VolLab	Text mit 11 Buchstaben: „NO NAME „
FilSysType	Text mit 8 Buchstaben: „FAT12 „ oder „FAT16 „

Boot Area

Das „reserved sectors area“ (Bild 2) beginnt mit dem ersten Sektor der Karte, also LBA (0) . Für FAT12 und FAT16 belegt er ohnehin nur diesen einen Sektor. Den Sprungbefehl in JMP und das Programm im Boot Code ist für eine „system disk“, von der aus der PC booten kann. Auch für einen reinen Datenspeicher braucht man jedoch zumindest einen gültigen Jump-Code. Das Programm in Boot-Code kann entfallen. Die drei Bytes EBh, xxh, 90h oder E9h, xxh, xxh sind legale Jumps, wobei xxh für ein beliebiges Byte steht.

Die Vendor-ID enthält vorzugsweise einen ASCII-String, wie z.B. „`IBM 2.0`“, „`CCC 2.1`“ oder moderner „`MSWIN4.1`“. Der Text gibt nominell Hersteller und DOS-Version des Systems an, das den Speicher formatiert hat. Das sollte keine technische Bedeutung haben. Aber schlechte Treiber haben angeblich Probleme, wenn sie keinen gängigen Namen finden.

BPB

Im „BIOS Parameter Block“ findet sich der Kern der Information über die Formatierung der Karte. Der Teil ab TotSec32 wurden nicht anno DOS 2.0, sondern erst später definiert [1], um zu FAT32 kompatibler zu werden. Die beiden letzten Byte des Sektors müssen das Wort AA55 enthalten.

FAT

Der „File Allocation Table“ existiert doppelt: hinter der aktiven Ausgabe liegt eine zweite identische Version (Bild 1). Das war als Sicherheitsfunktion gedacht, um beschädigte Disketten reparieren zu können. Die Funktion wurde von Microsoft nicht implementiert. Der Bereich sollte aber trotzdem immer identisch gehalten werden.

Die Wortadressen der FAT dienen implizit als Zeiger auf die Cluster im Datenbereich (Bild 3). Ihr Umfang hängt also von der Zahl der Cluster und damit von der Speichergrösse ab. Die Festlegung erfolgt während der Formatierung. Die Zahl der von der FAT belegten Sektoren findet sich im Bootsektor in ‘FATSz16‘ (Tabelle 1). Wieviel Zeiger man in einem Sektor unterbringt, hängt von dessen Wortlänge ab. In FAT16 ist sie 16 Bit in FAT12 nur 12 Bit.

Ob FAT12 oder FAT16 verwendet steht nicht direkt im Bootsektor. Es wird durch die Zahl der Datencluster festgelegt:

FAT12 : 0 - 4084d
FAT16 : 4085d - 65524d
FAT32 : > 65524d

Laut [1] soll man bei der Formatierung von diesen Grenzwerten um +/-16 Abstand halten, weil es eine Menge defekter Treiber gibt.

Der Inhalt der FAT sind Zeiger und Token (Tabelle 3). Ausnahme ist das “media descriptor“-Byte in FAT(0). Es entspricht „Media“ im Bootsektor (Tabelle 1), aber es werden die die oberen Bits gesetzt, also Fxx (FAT12) bzw FFxx (FAT16). Ihm folgt in FAT(1) ein EOC. Dieses „End Of Chain“ ist der Stopper einer Teilliste (Bild 4).

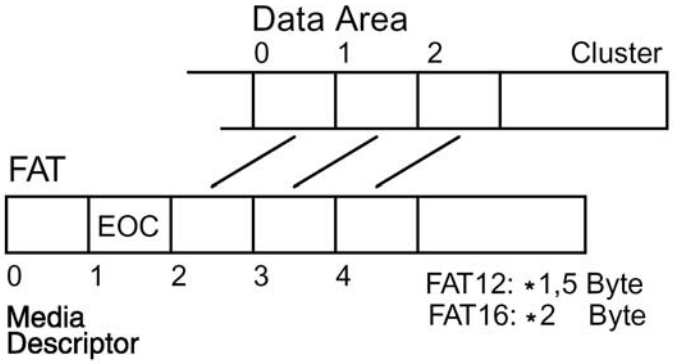


Bild 3: Zeiger in FAT

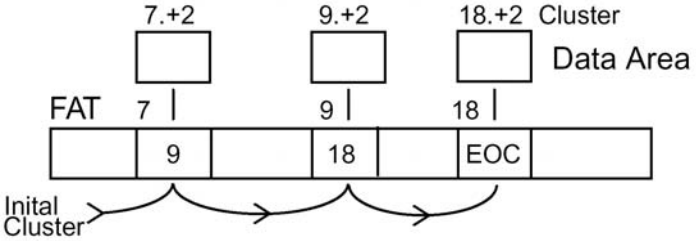


Bild 4: Liste in FAT

Tabelle 3: FAT-Token

FAT12	FAT16	
000	0000	free
001 - FEF	0001 - FFEF	in-use
FF0 - FF6	FFF0 - FFF6	reserved
FF7	FFF7	bad
FF8 - FFF	FFF8 - FFFF	end-of-chain EOC

Die Token „reserved“ und „bad“ werden bei der Formatierung fest vergeben, ansonsten sind dann alle Cluster ‘free’. Auf einer CF-Karte gibt es für den Treiber keine Sektoren, die “bad“ sind, weil dafür der Controller im Inneren der Karte zuständig ist. Er simuliert einen zusammenhängenden Bereich guter Sektoren. Normalerweise sind einige Reservesektoren verfügbar. Aber es ist denkbar, daß die Karte im Lauf des Betriebs immer mehr Sektoren als defekt aussondern muss und damit die Kapazität sinkt.

Zweck der FAT ist es, für Files Speicherbereiche variabler Länge zu ermöglichen. Diese werden als verkettete Liste dargestellt (Bild 4), wobei die Adresse in der FAT den belegten Cluster angibt, während der Inhalt entweder auf den nächsten Cluster zeigt, oder der Stopper FFF (FAT 12) bzw. FFFF (FAT16) ist. Die anderen EOC-Werte sind ungebräuchlich.

Man beachte, daß normalerweise einige Slots am oberen Ende der FAT auf nichtexistierende Cluster zeigen. Genauso existieren oberhalb der gültigen Cluster meist noch einige Sektoren, deren Zahl nicht für einen weiteren kompletten Cluster gereicht hat. Man könnte durch geschickte Einstellung bei der Formatierung das Root-Directory um diese Sektoren erweitern.

Root Directory

Der Einsprung in die “cluster chain“ der FAT für ein File befindet sich im Directory. Auch der Umfang dieser Liste wird bei der Formatierung fest eingestellt, alle Bytes werden auf 00h gelöscht.

Diese Struktur ist simpler aufgebaut als die FAT. Es ist einfach eine Liste, in der jeder Eintrag die feste Länge 32 Byte hat (Bild 5). War der Slot noch nie benutzt, ist das erste Byte 00h. Wenn ein File gelöscht wurde, wird es mit E5h bzw. 05h gekennzeichnet. Sucht man später durch diese Liste und findet einen leeren 00h-Kopf, kann man davon ausgehen, das Ende des belegten Bereichs erreicht zu haben und kann die Suche beenden.

Wenn der Slot belegt ist, befindet sich in den beiden ersten Feldern der Filename in ASCII im bekannten 8.3 Format. Namen bzw. Extension sind linksbündig angeordnet, und alle unbelegten Zeichen sind 20h. Es empfiehlt sich für beide die Beschränkung auf Großbuchstaben und Ziffern, obwohl nominell einige Sonderzeichen möglich wären. Die Verarbeitung im PC wird durch Verwendung dort üblicher Extensions vereinfacht, z.B. für ein ASCIITextfile TXT.

Im „File Attribute Byte“ (Tabelle 4) ist 20h „normal read-write“ die typische Einstellung. Das „archive bit“ ist normalerweise immer gesetzt, hat aber sonst keine praktische Bedeutung. Mit „readonly“ könnte man schreibgeschützt einstellen. Wenn ein File mit „hidden“ gekennzeichnet ist, sollte es durch den DIR-Befehl, der die Files des Speichers auflistet, dem Benutzer nicht angezeigt werden.

Der PC erzeugt beim Formatieren einen ersten Eintrag an DIR(0) „volume ID“ (Bild 1), der keine weitere Funktion hat. Das Namensfeld hat alle Bytes auf FFh gesetzt.

Im Bereich “reserved“ können alle Bytes auf 00h bleiben. Man kann, wie in [1] beschrieben, weitere Datumsangaben dort unterbringen. Diese waren ursprünglich nicht definiert und werden deshalb hier nicht berücksichtigt.

Zeit & Datum sind offensichtlich nur interessant, wenn der Controller eine Uhr hat (Bild 7). Die Zahlen sind binär dargestellt. Für die Sekunden ist der Takt 2 sec. Deshalb reichen 5 Bit. Die Jahre werden als Offset auf das Jahr 1980 gespeichert.

Das Feld “Inital Cluster“ enthält den Einsprung in den FAT, d.h. eine 12- oder 16-Bit Zahl.

Die Angabe von “Size“ bezieht sich auf die Bytes, die tatsächlich Daten enthalten. Ein Textfile, das nur 10 Buchstaben enthält, würde trotzdem einen kompletten Cluster Speicher verbrauchen. Die Darstellung dieser 32 Bit ist little-endian.

Analyse

Für Tests empfiehlt sich der Blick auf eine Karte, die vom PC formatiert wurde. Vorher sollte man die Karte am Einplatinencomputer komplett auf FFh löschen. Im Listing befindet sich dafür der Befehl `MASS-ERASE`. Danach läßt man den PC formatieren und lädt noch ein oder zwei Textfiles als Testdaten auf die Karte. Eine 8 MB wird typisch auf FAT12, eine 32 MB auf FAT16 eingestellt. Vgl. Ausdrucke mit Befehl `FAT.` (Tabelle 1 , Tabelle 2). Windows XP verwendet FAT32 und ist für solche Tests ungeeignet.

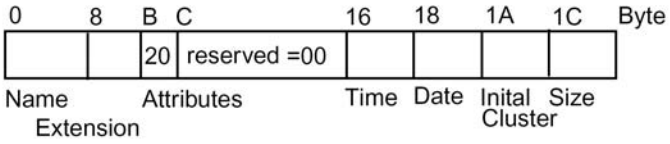


Bild 5: File-Eintrag im Directory

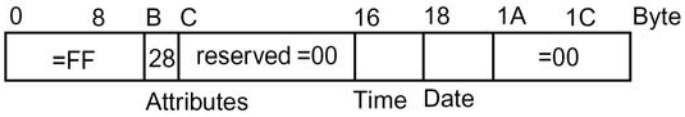


Bild 6: Volume-ID

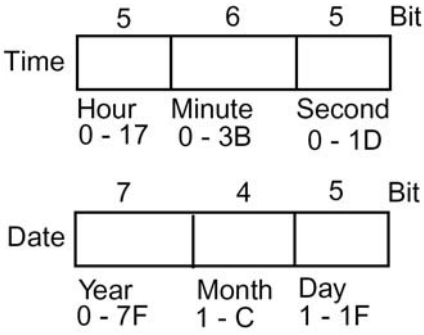


Bild 7: Datumsformat

Tabelle 4: File Attribute Byte

```
b b b b b b b b
              ^ read-only file
             ^ hidden
            ^ system file
           ^ volume ID
          ^ directory
         ^ archive bit = 1
        ^ unused = 0
^ unused = 0
```

Tabelle 5: Typischer Bootblock FAT16 auf 32 Mbyte

```
JMP-instruction ....... EB 3E 90
vendor-ID ............. +GYZfIHC
bytes/sector ......0200 0200
sectors/cluster ....... 04
reserved sectors = 0001 0001
number of FATs . = 02 02
max number of files.... 0200
number of sectors ..... E800
media descriptor ...... F8
sectors in one FAT .... 003A
sectors/track ......... 0020
number of heads ....... 0040
hidden sectors . = 00 0000
FAT32 sectors =00000000 0000 0000
drive nummer .......... 80
boot signature ........ 29
volume-ID ............. 3950 3950
volume label ..........
FAT text .............. FAT16
the end ........ = 55AA 55 AA
```


Tabelle 6: Typischer Bootblock
FAT12 auf 8 Mbyte

```
JMP-instruction ....... EB 3E 90
vendor-ID ............. +AfaiIHC
bytes/sector ......0200 0200
sectors/cluster ....... 08
reserved sectors = 0001 0001
number of FATs . = 02 02
max number of files.... 0200
number of sectors ..... 3000
media descriptor ...... F8
sectors in one FAT .... 0005
sectors/track ......... 0020
number of heads ....... 0040
hidden sectors . = 00 0000
FAT32 sectors =00000000 0000 0000
drive nummer .......... 80
boot signature ........ 29
volume-ID ............. 0D37 0D37
volume label ..........
FAT text .............. FAT12
the end ........ = 55AA 55 AA
```


[1] „FAT: General Overview of On-Disk Format“
fatgen102.pdf
[2] „Microsoft Extensible Firmware Initative FAT32 File System Specification“
fatgen103.pdf
[3] Lai „Writing MS-DOS Device Drivers“
Addison-Wesley 1987

Forth von der Pike auf

Teil 1

Ron Minke

Die hier mit freundlicher Genehmigung der HCC-Forth-gebruikersgroep wiederzugebende achtteilige Artikelserie erschien im vergangenen und im laufenden Jahr in der Zeitschrift "Vijgeblaadje" unserer niederländischen Forth-Freunde. Übersetzung: *Fred Behringer*.

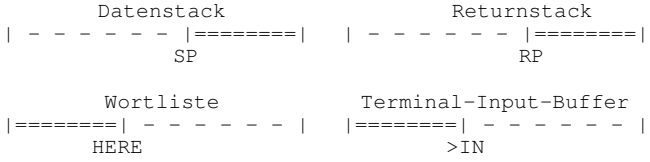
Die folgenden Zeilen stellen den Versuch dar, ein Forth-System auf die Beine zu stellen, dessen Voraussetzung “überhaupt nix”, oder auf gut Deutsch “from scratch”, lautet.

Zunächst ein paar Worte über die Vorgeschichte: 1997 bringt die Firma Atmel aus ihrer AVR-Serie den Mikroprozessor AT90S8515 auf den Markt. Laut Atmel ist die AVR-Serie eine Serie von revolutionären Mikroprozessoren, die zwar die Vorteile von ähnlichen Prozessoren auf diesem Marktsegment von 8-Bittern haben, aber nicht deren Nachteile. Überdies haben Chipbäcker und Software-Spezialisten (sprich C-Compilerbauer) beim Entwickeln des Prozessors von Anfang an zusammengearbeitet. Weitere Informationen findet man auf der Atmel-Website.

Ein neuer Prozessor! Für den echten Forth-Enthusiasten ist das natürlich eine Herausforderung: Da muss Forth drauf laufen! Und damit begann es ...

Erst flugs den Befehlssatz des neuen Prozessors durchstudieren. Und was hat er denn an Timern, Ports, seriellen I/O-Anschlüssen und so weiter schon an Bord? Und wie sieht denn ein Forth-System auf unterster Ebene noch gleich aus? All die verschiedenen Pointer ...

Wie baut sich Forth auf? Der virtuelle Forth-Computer ist ein Programm, das im Arbeitsspeicher eines echten Computers, im vorliegenden Fall unseres AVR-Mikroprozessors, läuft. Der vorhandene Speicher ist in Bereiche für verschiedene Aufgaben aufgeteilt und das Ganze sorgt dafür, dass der echte Computer den Forth-Kommandostrom verarbeiten kann.



Das ist eine schematische Wiedergabe der Funktionsteile des einfachsten Falles eines virtuellen Forth-Computers. Er besteht aus einer Wortliste (dem Dictionary), zwei Stacks, einem Terminal-Input-Buffer und eventuellem Disk-IO (Input-Output-Anschlüsse).

Die virtuelle Forth-Maschine verwendet einen Satz von Registern, um die hauptsächlichsten Informationen über den Verlauf des Programms zu steuern. An Registern treten auf:

SP Datenstack-Pointer
RP Returnstack-Pointer
IP Interpreter-Pointer (wo wird gerade gearbeitet)
W Word-Pointer (zum laufenden Wort)
PC Program-Counter (Machinenprogramm-Zähler)

Sodann beginnt die “echte Arbeit”: Wie wollen wir die Interna unseres Mikroprozessors (Register und dergleichen) zur Implementation der virtuellen Forth-Maschine einsetzen? Um diese Frage gut beantworten zu können, müssen wir erst ausfindig machen, wie die virtuelle Forth-Maschine ihre Register verwendet. Zur Verdeutlichung haben wir einen Pseudobefehlssatz für einen Pseudo-Assembler definiert.

Dieser Befehlssatz enthält nur drei Befehle:

MOV dest,src	Kopiere das Datenregister src ins Register dest
INC dest	Vergrößere den Inhalt des Registers dest um 1
DEC dest	Verkleinere den Inhalt des Registers dest um 1

Es existiert auch eine indirekte Form: Das Setzen von dest oder src in Klammern signalisiert, dass es hier “um den Inhalt von” geht, und nicht um das Register selbst. Die Klammern deuten

eine Indirektionsebene an.

MOV dest,(src)	Kopiere Daten vom INHALT des src-Registers, d.h., wohin src zeigt, ins Register dest.

Die Wortliste von Forth ist eine verkettete Liste von aneinandergereihten Wortdefinitionen. Man erkennt diverse eigenständige Teile (Felder):

Namensfeld	Hier steht der Name des Wortes
Linkfeld	Ankopplung an das vorherige Wort
Codefeld	Zeiger auf ausführbaren Maschinencode
Parameterfeld	Was muss dieses Wort tun?

Die oben stehenden Pseudobefehle brauchen wir, um uns in den aneinandergekoppelten Feldern dieser Wortdefinitionen umherbewegen zu können. Das Namensfeld und das Linkfeld sorgen dafür, dass Definitionen zu einer linearen Liste zusammengekoppelt werden können, die dann vom Textinterpreter durchsucht werden kann. Wie das genau geschieht und wie das eine oder andere in den Speicher gesetzt wird, bewahren wir uns für den nächsten Artikelteil auf. Das Codefeld enthält die Adresse des Code-Interpreters für diese Definition und das Parameterfeld enthält alle Informationen, die diese Definition benötigt, um ihre Aufgabe auszuführen.

Die Suche nach der besten Kombination von Prozessorregistern, um Forth laufen lassen zu können, beginnt beim Wort EXECUTE im Text-Interpreter. Das Wort EXECUTE ruft ein Stückchen Maschinencode auf, um das betreffende Wort auszuführen. Hierbei müssen wir berücksichtigen, dass bei jedem Aufruf von EXECUTE die Adresse des Codefeldes (die CFA, Code Field Address) desjenigen Wortes, das ausgeführt werden soll, vom Text-Interpreter auf den Stack gelegt wird.

Es folgt der Code in Pseudo-Assembler-Notation:

EXECUTE: (CFA --)	Codefeld-Adresse steht auf dem Datenstack
MOV W,(SP)	Kopiere den obersten Datenstack-Eintrag, CFA, ins W-Reg.
INC SP	Verwerfe den Datenstack-Wert, mache einen Schritt vorbei
MOV PC,(W)	Kopiere die Adresse des Code-Interpreters in der Code Field Address in den Program Counter: Indirekter Sprung dahin

EXECUTE kopiert die CFA in das W-Register und springt – indirekt – über den Inhalt eben dieses W-Registers zum Code-Interpreter. Der Machinencode des Code-Interpreters wird nun ausgeführt. Da das W-Register selbst weiterhin auf die CFA des Forth-Wortes zeigt, das gerade ausgeführt wird, kann der Code-Interpreter Informationen darüber einholen, wie das Forth-Wort weiter zu behandeln ist. In unserer Forth-Implementation (FIG-Forth, siehe betreffende Literatur) liegt das Parameterfeld direkt hinter dem Codefeld. Wenn die Parameterinformation benötigt wird, holen wir diese von hier aus über den betreffenden Code-Interpreter ein.

Alle Code-Interpreter müssen ihren ausführenden Teil mit einem Stück Code beenden, der “NEXT” genannt wird. Der gibt die Kontrolle an den darüberliegenden Text-Interpreter zurück. Wird der Code-Interpreter von einem Hi-Level-Wort aufgerufen, so wird die Kontrolle an dieses Hi-Level-Wort zurückgegeben. NEXT setzt voraus, dass die Adresse des als nächstes auszuführenden Wortes im IP-Register aufbewahrt wird (der “Wo-bin-ich-geblieben”-Pointer). So kann der Text-Interpreter die Liste der im Parameterfeld einer Hi-Level-Definition gelagerten Adressen scannen.

Das Codestück für NEXT sieht in unserem Pseudo-Assembler so aus:

NEXT:	IP zeigt auf das als nächstes auszuführende Wort
MOV W,(IP)	Kopiere den Inhalt von IP, die CFA des als nächstes auszuführenden Wortes, ins W-Register.
INC IP	Lass IP auf das Wort NACH dem momentanen zeigen, um da schnurstracks weiter zu gehen.
MOV PC,(W)	Führe den Code-Interpreter aus, dessen Adresse jetzt im W-Register sitzt. Dieser Wert kommt aus dem Codefeld des gerade auszuführenden Wortes (indirekter Sprung).

Alle Worte in der Wortliste können durch EXECUTE ausgeführt werden, wenn ihre CFA auf dem Datenstack steht, oder durch NEXT, wenn die CFA, die sich in der Wortliste befindet, durch IP angezeigt wird. Anzumerken bleibt noch, dass es bei der Ausführung eines Forth-Wortes der Maschinencode des Code-Interpreters ist, der vom “Host-Computer” ausgeführt wird. NEXT und EXECUTE kriegen die Adresse dieses Code-Interpreters aus dem Codefeld der Definition des betreffenden Wortes zugewiesen.

Im nächsten Artikelteil werden wir weiter auf die Interna eingehen, und zwar auf das tatsächliche Abbilden der Register des AVR-Prozessors auf die Register der virtuellen Forth-Maschine, und natürlich auch auf den zugehörigen Machinencode.

Vorwort:
Der Leserbrief des langjährigen Redakteurs der englischen Vereinszeitschrift "Forthwrite" kam uns (der Redaktion der VD; siehe Seite 12) erst zu einem Zeitpunkt zu Gesicht, als der zweite Teil des Artikels von Bernard Hodson in der Übersetzung von Fred Behringer schon vorlag. Der Fairness halber müsste man aber auch sagen, dass man das von Chris Jakeman erwähnte Buch "The Gene Machine" gelesen haben sollte, um sich ein abschließendes Urteil zu bilden. Wir haben es leider nicht auftreiben können. Chris hat es aber offensichtlich auch nicht gelesen. Wir bringen also auch den zweiten Teil des Artikels von Bernard Hodson. Fred hat auch diesmal wieder bereitwillig übersetzt (und sich mit dem Autor über einige dunkle Punkte "kurzgeschlossen"), auch wenn er vom Inhalt nicht ganz überzeugt war.
Ebenfalls der Fairness halber muss auch gesagt werden, dass der Redaktion „extrem kritische“ Leserbriefe zum ersten Teil der nachfolgend abgedruckten Arbeit vorliegen. Hier soll es aber den Absendern bewusst vorbehalten bleiben, sich direkt an Bernard Hodson zu wenden.

EINE NEUE ART DER COMPUTERPROGRAMMIERUNG (2)

von

BERNARD A. HODSON

bernard@genetix.ca

Übersetzt von Fred Behringer

Dieser Teil der Arbeit liefert ein paar Zusatzinformationen, die zum klareren Verständnis der verwendeten Begriffe beitragen sollen. Im Text wird zunächst einmal ein Anwendungs-Übersetzer diskutiert. Interessierte Personen, die sich diesen Übersetzer ansehen (oder die ihren eigenen Übersetzer entwikkeln) wollen, wenden sich bitte an den Autor unter bernard@genetix.ca . Im vorliegenden zweiten Teil der Arbeit werden auch einfache Beispiele näher beleuchtet.

DER ANWENDUNGS-ÜBERSETZER

Die Vorgehensweise hat unter anderem zum Ziel, von der verwendeten Landessprache unabhängig zu sein. Das wird über eine Sprachtabelle wie folgt erreicht.

Tabelle

```
10,'deflooping',5,'arith',7,'looping',10,'defineloop',8,'binarray',10,'storearray',8,'getarray',8,'putarray',4,'keep',6,'tofile','7,'addfree',7,'testneg',9,'stopclear',6,'clears',7,'readtxt',8,'testzero',6,'sqroot',4,'ceil',6,'screen',7,'isfalse',10,'readnumber',4,'sine',6,'cosine',7,'tangent',9,'cotangent',6,'arcsin',6,'arctan',3,'exp',4,'logn',10,'createfile',5,'skips',8,'openfile',8,'readunit',5,'cases',8,'makefile',11,'movetofixed',7,'storeit',8,'readfile',6,'record',6,'testeq',6,'testlt',6,'testle',6,'testgt',6,'testge',6,'testne',4,'crlf',6,'textin',7,'addtext',8,'notinuse',5,'addon',8,'textedit',6,'bypass',7,'textout',8,'setgraph',9,'rectangle',4,'quad',8,'triangle',6,'circle',3,'zzz',3,'pie',7,'notused',5,'reset',7,'bitmapp',7,'include',6,'getlib',4,'exit',10,'prntscreen',10,'readscreen',0
```

Diese Tabelle kann in jeder Sprache verfasst sein, die auf "Western" aufbaut, sie sollte sich aber auch an Sprachen wie Arabisch und Hebräisch anpassen lassen.

Der Anwendungs-Übersetzer sieht sich einfach die relative Position der einzelnen Sprachelemente in der jeweiligen Anwendung an und gibt eine damit verbundene relative Zahl aus (so könnte zum Beispiel looping die Zahl 2 liefern und getarray die Zahl 6). Wenn neue Sprachelemente hinzugefügt werden sollen, werden sie einfach an das Ende der Tabelle angehängt.

Beim Entwickeln eines Java-Laufzeit-Systems wurde eine Tabelle eingerichtet, die den Java-Bytecodes entspricht. An sich ist es im Genetix-Laufzeitsystem von Natur aus nicht nötig, für jeden Java-Bytecode ein Tabellenelement vorzusehen. Ähnliches lässt sich für Forth, APL, Fortran, Cobol, Algol usw. sagen. Es mag so aussehen, als ob die Anzahl der Sprachelemente auf 256 beschränkt ist. Diese Grenze lässt sich aber leicht überwinden.

Wie in Teil 1 beschrieben, ist jedes Sprachelement im Laufzeit-System mit einem Sprachelemente-String von Zahlen verknüpft. Wir werden das gleich etwas näher umreißen.

ENTWICKLUNGSSPRACHE

Die näheren Einzelheiten der Sprache sollen hier nicht wiedergegeben werden. Wir wollen uns mit einer zusammenfassenden Betrachtungsweise begnügen. Diese Sprachelemente werden beträchtlich erweitert. Wir werden kurz ein paar Anwendungen beschreiben, die die Sprache verwenden. Zuvor die Syntax, in alphabetischer Reihenfolge.

addon	hängt betrachtete Variable an Datenblockende an
addtext	fügt zusätzlichen Text hinzu
arc	zeichnet einen Bogen
arcsin	berechnet arcsin
arctan	berechnet arctan
arith	führt arithmetische Operationen aus
binarray	erzeugt ein Array mit Anfangswerten
bitmapp	erzeugt Bitmap-Image
cases	verzweigt zum angegebenen Auswahl-Fall

ceil	berechnet kleinste ganze Zahl größer/gleich
clear	löscht Bildschirm und fährt fort
cosine	berechnet cosin
cotangent	berechnet cotan
createfile	erzeugt auf der Festplatte eine neue Datei
defineloop	legt Parameter für eine benannte Schleife fest
deflooping	bearbeitet die benannte Schleife
exit	gibt Kontrolle an das Betriebssystem zurück
exp	Exponentiation
field	holt festgelegte Felder
getarray	holt Werte-Array
getlib	erzeugt Bibliotheksgruppe
include	bindet Bibliotheksgruppe ein
isfalse	prüft, ob Bedingung falsch
keep	legt Variable für spätere Bearb. in eine Liste
library	läutet Bibliotheksgruppe ein
logn	logn
looping	erhöht Startwert so lange, bis Endwert erreicht ist
makefile	erzeugt Datei-Struktur
movetofixed	kopiert Feld in einen festgelegten Bereich
openfile	existierende Datei öffnen
prntscreen	legt Text an bestimmte Stelle auf dem Bildschirm
putarray	fügt Wert in Array ein
readfile	holt Datenblock (record)
readnumber	liest Zahl
readscreen	liest Daten von bestimmter Bildschirm-Stelle
readtxt	liest Text
readunit	liest Ziffer
record	legt Felder an die eine oder andere Stelle
rectangle	zeichnet Rechteck
reset	zurück zum Text-Modus
screen	legt Daten auf Bildschirm
setgraph	setzt Grafik
sine	berechnet Sinus
sinewave	Sinewave-Zwischenerzeugung
skips	Abgabe der Kontrolle
sqroot	berechnet Quadratwurzel
stopclear	hält Ausführung an und löscht Bildschirm
storearray	speichert Wert in Array
storeit	legt momentanen Datenblock (record) ab
tangent	berechnet Tangens
testeq	prüft auf Gleichheit
testge	prüft auf größer-oder-gleich
testgt	prüft auf größer-als
testle	prüft auf kleiner-oder-gleich
testlt	prüft auf kleiner-als
testne	prüft auf Ungleichheit
testneg	prüft auf negativ
testzero	prüft auf null
textedit	bearbeitet Text
textin	liest Text ein
textout	überträgt Text in eine Datei
transf	überträgt Daten in Arbeitsbereich

Viele der zur Zeit verwendeten Sprachelemente sind auch in dem Demonstrations-Übersetzer enthalten, dessentwegen man sich an bernard@genetix.ca wenden kann. Viele zusätzliche Sprachelemente sind in der Entwicklung, und diese externe Sprache wird beträchtlich erweitert werden, je nachdem, wie groß der Bedarf ist, der sich bei den in Frage kommenden Benutzern herausstellt. Insbesondere wird die Einrichtung spezieller Vokabulare für spezielle Industriezweige erwartet, so dass die Mitarbeiter in den betreffenden Firmen ohne Rückfrage bei Programmierern verstehen können, was ihre Anwendungen tun (vergleiche die Arbeiten am Risk-Research-Institut der Universität von Waterloo, insbesondere die Arbeiten von Dr. Siddall über die EM-Sprache).

ANWEISUNGSZEILE IN ANWENDUNGEN

Die andere Funktion des Anwendungs-Übersetzers besteht darin, die Anweisungen (wenn möglich) soweit zu komprimieren, dass sie in Strings von einem Byte Länge passen. Dies wird wieder über eine relative Bezugszahl erreicht. Eine solche Anweisung könnte wie folgt aussehen:

Screen ^d, h and sum is^ d h sum

Hier sollen die editierten Werte der Variablen d und h und deren Summe auf den Bildschirm gelegt werden.

In komprimierter Form könnte das so aussehen: !3482

In diesem Fall wurde ! anstelle des Doppelpfeils 18, der relativen Position des Sprachelementes screen, verwendet, 3 würde das dritte Zeichen darstellen, 4,8,2 die vierte, achte und zweite beim Übersetzen der Anwendung gefundene Variable. Die Zeichen (Literals) werden an das Ende des komprimierten Strings gesetzt. Es könnte so aussehen, als ob das System auf diese Weise auf 256 Variablen beschränkt wäre. Das ist aber nicht der Fall, obwohl andererseits eine Untersuchung gezeigt hat, dass nur wenige Anwendungen mehr als 256 voneinander verschiedene Variablen enthalten.

Eine Anweisung, in welcher eine Übertragung der Kontrolle vorkommt, könnte wie folgt aussehen:

Testzero Toyota #a #8

was in komprimierter Form als %429 wiedergegeben sein könnte.

Hierbei wurde % für das sonnenförmige Sonderzeichen 15 verwendet, das testzero wiedergeben soll, 4 steht für die vierte im Anwendungs-Übersetzer gefundene Variable und 2 und 9 sind der zweite und der neunte gefundene Verzweigungspunkt.

Der Anwendungs-Übersetzer zeigt alle in den Anwendungs-Anweisungen eventuell auftretenden Fehler an und erzeugt als komprimierte Zeichenketten nur solche, die alle Regeln erfüllen.

Damit sind die Übersetzeraktivitäten im Wesentlichen erfasst, und es erscheint jetzt angebracht, ein triviales Beispiel mit einigen einfachen Anwendungen zu untersuchen.

Genetix –neue Art der Programmierung...

Jedes Sprachelement ist im Laufzeitsystem mit einem kleinen String von numerischen Codes verknüpft, die sich auf die internen Funktionen beziehen, die aufgerufen werden müssen, um die in Frage stehende Funktion auszuführen. Diese internen Elemente sind ihrerseits wieder numerische Codes, die entweder andere interne Elemente oder aber Segmente des virtuellen Prozessors aufrufen. Die hierarchische Struktur der internen Elemente führt zu einem sehr kompakten Gesamtsystem.

Es folgen ein paar der momentan verwendeten internen Elemente:

apprun	startet den Anwendungslauf
branch	bestimmt, wohin verzweigt werden soll
clear	löscht den Bildschirm
cleara	setzt a auf null
clearb	setzt die arithmetische Stelle b auf null
clearr	setzt Ergebnisbereich auf null
close	schließt die in Bearbeitung befindliche Datei
convert	wandelt Dezimalzahl in internes Format um
create	erzeugt Datei
cursorx	handhabt den Bildschirm-Cursor
def1	Hilfsfunktion
def1a	Hilfsfunktion
def2	wird in define verwendet
definet	wird bei der Erzeugung von Schleifen verwendet
exits	kehrt zum Betriebssystem zurück
edit	bearbeitet numerische Daten
getar	holt arithmetische Variable
getbinarr	bei Initialisierung eines binären Arrays verwendet
getbinary	holt Binärwerte für Schleifenkontrolle
getconst	holt benötigte Konstante
getdigit	einzelne Ziffer ins Binärformat
getloops	richtet Schleifensprünge ein
getnumber	wandelt Ziffern in interne Darstellung um
getputarray	wird in Array-Operationen verwendet
gettext	sucht nach verlangtem Textblock
offsett	holt den Variablen-Offset
opens	öffnet Datenblock (record)
varfind	sucht nach Variablennamen
variable	holt Variable
varput	speichert Variable ab
varputa	speichert arithmetische Werte ab

WORLD

```
batip01
        \Diese Anwendung legt eine einfache Nachricht auf den
         Bildschirm\
screen ^hello world out there^ ^are you happy?^
exit
zend
```

Die "Hallo-Welt-Anwendung" ist dumm und trivial. Sie begegnet einem aber immer wieder. Sie scheint der Heilige Gral für neue Systeme zu sein. Also wollen wir sie auch hier bringen.

Man sieht, dass hier genau wie in Java eine 'magische' Zeichenkette auftritt: Alle Anwendungen beginnen mit batip01. Die 01 gibt die Versionsnummer wieder.

In allen Anwendungen steht als letzte Zeile die Anweisung zend.

Die Anweisung exit kann überall in der Anwendung stehen.

Kommentare beginnen und enden mit \

Abgesehen von den zwei Literalen (Zeichen), würde sich als komprimierte Version !01"@ ergeben. ! repräsentiert das Sprachelement screen, " wird als Abschlusselement einer Anweisung verwendet, und @ steht für die Anweisung exit. 0 und 1 sind in Binärdarstellung gegeben.

CELC Temperaturumwandlung

```
batip01
arith fahr = 68
screen ^Fahrenheit      Celcius^
#2 arith celcius = fahr - 32
arith celcius = celcius / 9 * 5
screen fahr ^           ^ celcius
arith fahr = fahr + 18
looping 68 18 248 #1 #2
#1 exit
zend
```

In diesem Fall wäre fahr die Variable 0, celcius die Variable 1, #2 wäre der Verzweigungspunkt 0, und #1 der Verzweigungspunkt 1. Die Konstanten 32,9,5,18,68,248 werden, je nachdem, was sie sein sollen, ins Binär- oder Gleitkomma-Format umgewandelt. Ein vorangesetztes Zeichen zeigt den geforderten Typ an.

Das eben genannte Beispiel würde sich dem Laufzeitsystem in komprimierter Form mit etwa 32 Bytes präsentieren. Hinzu kämen noch die Zeichen (Literale) und die Konstanten.

Binäre Konstanten verwenden die jeweils erforderliche Anzahl von Bits, Gleitkomma-Konstanten benötigen mindestens vier Bytes, drei für Länge, Vorzeichen und Exponent. (Die Länge wird für das Format "einfache Länge" auf 32 Bytes erweitert.) Man kann also, absolut genommen, Zahlen im Größenordnungsbereich 10 hoch +/- 128 darstellen.

Der sich tatsächlich ergebende komprimierte Bytecode kann auf meinem Drucker oder meinem Bildschirm nicht ausgegeben werden. Eine verbale Beschreibung des Bytestroms würde den Leser aber nur verwirren.

FACT Berechnung der Fakultät

```
batip01
#5 arith up = 1
arith start = 1
screen ^enter number less than 30  ^
readnumber number
arith number1 = number
testneg number #1 #a
#a testzero number #1 #2
#1 exit
#2 arith number = number + 1
screen ^entered factorial for ^ number1
#4 arith start = start * up
arith up = up + 1
arith number1 = number1 - 1
testzero number1 #3 #4
#3 screen start
skips #5
exit
zend
```

Diese Anwendung fordert den Benutzer auf, eine Zahl n einzugeben, und liefert dann als Ergebnis die Fakultät n! dieser Zahl. Sie ist vor allem ein Maß für die Genauigkeit von arithmetischen Multiplikationen.

Die Variablen 0,1,2,3 sind: up, start, number, number1
Die Verzweigungspunkte 0,1,2,3,4,5 sind: #5,#1,#a,#2,#4,#3

Diese Anwendung würde 65 Bytes erzeugen, wobei noch die Literale (Zeichen) und Konstanten hinzukämen.

BITMAP

Diese experimentelle Anwendung soll hier nicht besprochen werden. Sie erzeugt ein JPEG-Bild.

Als Beispiel eines Sprachelementes betrachten wir zcalc, das mit der Anweisung arith verknüpft ist. Überarbeiten wir das, um Überflüssiges zu beseitigen, so erhalten wir:

```
zcalc:   zichvind,0,128,addcode1,1,zddcode1,c7,2,addcode1,1,
zgetar   zddmsgline,v,0,zddcode1,st5,2,code1query,253,4,16,
zddv,r,32   zstoreno,addcode1,1,endline,zddcode1,st6,2,
addcode1,1,zgetar   zddmsgline,v32,0,zactcalc,
code1query,253,6,clrst6,156,addcode1,1 endline
```

zcalc ist eines der längeren Sprachelemente, und bei addcode1, zddcode1, usw. handelt es sich eigentlich um Zahlen. Eine solche Zahl gibt dem Laufzeitsystem exakte Auskunft darüber, was zur Ausführung einer arithmetischen Operation benötigt wird. Der Hauptteil der Aktivitäten bezieht sich direkt auf Laufzeit-Module, zgetar, zstoreno und zactcalc jedoch sprechen andere interne Elemente im hierachischen Verbindungsnetz an, während endline ein Kontrollwort ist.

Die Anzahl der Module im Laufzeitsystem ist recht klein und kann bis jetzt mit einer Zahl von 0-65535 erfasst werden. Die prinzipiellen Kategorien des Zahlensystems sehen wie folgt aus:

```
;33280  msgs
;33792  keys
;34304  functions
;34816  adding
;35328  subing
;35840  multing
;36352  diving
;36864  addctl
;37376  subctl
;38400  zdd ops
;38912  zero2
;39424  zdi ops
;39936  zid ops
;40448  zii ops
;40960  zichmove
;41472  offquery
;41984  matchis
;42496  matching
```

Die im Message-Modul (msgs) zur Zeit verwendeten 5 Bestandteile sind:

```
;msgs 33280
appnmsg = 33280
invalidmsg = 33281
msglinemsg = 33282
codesmsg = 33283
```

Eines der Kopiermodule andererseits hat 12 Bestandteile. Die ersten 5 sind:

```
;39936 ;   zid im Laufzeitsystem
zidtestno = 39941
zidst1 = 39968
zidst2 = 39969
zidst3 = 39970
zidst4 = 39971
```

Die ersten sieben eines anderen Moduls sind:

```
;40448 zii im Laufzeitsystem
ziist1 = 40480
ziist2 = 40481
ziist4 = 40483
ziist5 = 40484
ziist12 = 40491
ziicode1 = 40501
ziic6ind = 40523
```

In diesem Beispiel führt die zii-Basis von 40448 das Laufzeitsystem zu jenem Modul, das Daten von der einen indirekt definierten Speicherstelle zu einer anderen indirekt definierten Speicherstelle kopiert. Die Differenz zwischen der Basis und dem verwendeten Element-Code legt die in dieser Operation zu verwendende Variable fest. So bezieht sich beispielsweise

40481 auf die 33-te Variable in einer Tabelle von Variablennamen (st2), wohingegen 40501 den 53-ten Variablennamen angibt (Code1).

NUTZEN UND VERWENDUNG DES REVIDIERTEN SYSTEMS

Es drängt sich natürlich die Frage auf, was uns diese Neuordnung des herkömmlichen Begriffsgebäudes bringt. Es gibt eine ganze Reihe von Vorteilen, die jetzt aufgezählt werden sollen.

Als 'Vision' hat man die Vorstellung, dass ein einziges Programm für alle Anwendungsmöglichkeiten verwendet werden kann. Das heißt nicht, den gesamten Code in jedem einzelnen Fall zu verwenden (es wäre z.B. zwecklos, auf einer Smart-Card Grafik einzusetzen), es heißt aber, für jede Plattform Modul-Modelle zur Verfügung zu haben, die schon auf anderen Plattformen getestet wurden und die auf jeden Fall so klein sind, dass das Austesten trivial wäre.

Der Funktionsumfang kann je nach Bedarf erweitert oder reduziert werden. Damit bietet sich eine viel größere Chance, auf Kundenwünsche eingehen zu können. Insbesondere wird das wichtig, wenn ein und dieselbe Software auf vielen Plattformen verwendet werden soll, beispielsweise bei der Übertragung von Software von einem Wirtscomputer auf eine Smart-Card, wobei nur ein kleiner Teil der verfügbaren Module auf der Smart-Card benötigt wird.

In einer früheren Version hatte sich gezeigt, dass man mit GENETIX alles tun konnte, was mit Java möglich war, und das in einer sehr viel kompakteren Form, ohne den komplizierten Ballast, den alle Java-Laufzeitsysteme mit sich herumschleppen. Und bei dem jetzt bestehenden numerischen System gilt das sogar in noch stärkerem Maße.

Was Java betrifft, ist die Situation sogar noch ein ganzes Stück schlimmer, als eben angedeutet. Bei der numerischen Technik braucht sich das Laufzeitsystem nicht um solche Dinge wie Ausnahmefälle (Exceptions) zu kümmern, oder um Methoden, die denen der Anwendung untergeordnet sind, oder um Multi-Threading, Dinge, die geeignet sind, die Java-Anwendungen aufzublähen.

Ein und dieselbe Software auf allen Plattformen zu haben, heißt, dass jede Plattform sowohl Sender wie auch Empfänger sein kann. Besonders wichtig ist das auf dem Gebiet der Nachrichtenübertragung, wo es im Falle einer Fehlfunktion des einen Systems erforderlich sein kann, dass ein anderes System übernimmt. In gewisser Weise spiegelt das die im Internet herrschende Situation wider.

ACKNOWLEDGEMENT

I wish to express my thanks to Fred Behringer for his patience in translating my papers, and for his suggestions to clarify certain obscure points. *Bernard A. Hodson*

volksForth

Ein Projekt

Carsten Strotmann

volksFORTH 3.81 für den IBM-PC und Kompatible wurde aus dem volksFORTH 3.80 für CP/M Rechner entwickelt. Es wurden wenige Änderungen am Kern des Systems vorgenommen. Diese beziehen sich hauptsächlich auf Stellen, die in der Version 3.80 unelegant waren und für die inzwischen bessere Lösungen in der Forth-Community erarbeitet worden waren.

Grundlegend überarbeitet wurde das File-Interface. Auf der Benutzerebene stehen die gleichen Worte wie im volksFORTH 3.80 für den ATARI und CP/M zur Verfügung. Die darunterliegende Implementation wurde jedoch grundlegend geändert, so dass jetzt endlich in Forth auch sequentielle Files, die nicht die starre BLOCKstruktur haben, manipuliert werden können. Damit ist es endlich möglich, auch volksFORTH für kleine Hilfsprogramme zu verwenden, die mit anderen Programmen erstellte Files "bearbeiten" und durch den Befehl SAVESYSTEM als "standalone"-Programm abgespeichert wurden.

Besonders weitreichende Möglichkeiten erschließen sich dadurch, dass beim Aufruf von volksFORTH auf der Betriebssystemebene noch eine ganze Kommandozeile mit übergeben werden kann, die dann unmittelbar nach dem Booten von Forth ausgeführt wird. Durch die Systemvariable RETURN_CODE kann nach Verlassen des Forthprogramms ein Wert an MSDOS zurückgegeben werden, der mit dem Batch-Befehl ERRORLEVEL abgefragt werden kann.

Darüberhinaus ist es auch möglich, mit dem Befehl MSDOS aus dem Forth heraus eine weitere COMMAND.COM shell aufzurufen und später mit EXIT wieder ins Forth zurückzukehren, wobei der Bildschirm, der zum Zeitpunkt des Aufrufs bestand, wiederhergestellt wird. Selbstverständlich kann neben MSDOS selber auch jedes andere beliebige Anwendungsprogramm aufgerufen werden - auch eine weitere Inkarnation des Forth-Systems - so dass sich mit diesen Möglichkeiten die Begrenzungen, die in dem beschränkten Adressraum von 64k liegen, überwinden lassen. Auch komplizierte Overlaystrukturen sind nicht mehr notwendig. Es werden einfach aus einem zentralen "Verwaltungsprogramm" heraus spezielle Forth-Anwendungsprogramme aufgerufen.

Die Distributionsdiskette des volksFORTH 3.81 für den IBM-PC und Kompatible enthält folgende Files:

PKXARC.COM	"Auspack"-Programm für die .ARC Files
INSTALL. BAT	Installiert volksFORTH auf Disketten oder in Subdirectory

READ.ME	Installationshinweis
KERNEL.COM	Der Kern des volksFORTH
FORTH1.ARC	Kompaktierte Sourcefiles für das VOLKS4TH.COM
FORTH2.ARC	Kompaktierte Sourcefiles weiterer Hilfsprogramme
MINIMAL.COM	Ein Forthprogramm zur Anpassung an "Fastkompatible"
VOLKS4TH.COM	Die durch INCLUDE VOLKS4TH.SYS erzeugte Arbeitsumgebung

Auf dieser Diskette ist der Sourcecode mit dem weitverbreiteten Hilfsprogramm PKXARC in den Files FORTH1.ARC und FORTH2.ARC "abgepackt".

Mit dem Programm INSTALL.BAT muss nun zuerst der Sourcecode "ausgepackt" werden. INSTALL erwartet einen Parameter, der das Ziel der Auspackoperation angibt.

Mit INSTALL A: oder INSTALL B: wird auf zwei Disketten eine Diskettenversion erzeugt. Mit INSTALL d:\path wird die Installation in das durch "d:\path" angegebene Subdirectory vorgenommen.

Für die Diskettenversion ist zu beachten, dass die erste Diskette ca. 340k Byte Platz beansprucht, so dass bei 360k Laufwerken das Betriebssystem gleichzeitig auf dieser Diskette keinen Platz mehr hat. Es gibt für den praktischen Betrieb in einer reinen Diskettenumgebung keine Erfahrung, so dass ich nur hoffen kann, dass es funktioniert.

Hinweis zum Copyright

Die Programme und die zugehörigen Quelltexte können frei verwendet werden. Das beinhaltet die Weitergabe und Nutzung der Programme und gilt selbstverständlich auch für Applikationen, die auf volksFORTH aufgebaut sind.

Das Handbuch unterliegt dem Copyright;
(c) 1985 - 1988 Klaus Schleisiek, Ulrich Hoffmann, Bernd Pennemann, Georg Rehfeld und Dietrich Weineck.

Wenn die Distributionsdiskette "ausgepackt" ist, so finden sich folgende Files:

KERNEL.COM	Der volksFORTH Kern, der von KERNEL.SCR erzeugt wurde
MINIMAL.COM	Minimalsystem für die Anpassung an "Fastkompatible"
VOLKS4TH.COM	Normale Softwareentwicklungsumgebung, mit VOLKS4TH.SYS erzeugt
ANSI.VID	Videodisplaytreiber auf der Basis des ANSI.SYS MS-DOS Treibers
BIOS.VID	Videodisplaytreiber durch BIOS-call $10
MULTI.VID	Multitasking Videodisplaytreiber durch BIOS-call $10
MINIMAL.SYS	Loadfile, um aus KERNEL.COM MINIMAL.COM zu erzeugen
VOLKS4TH.SYS	Loadfile, um aus KERNEL.COM VOLKS4TH.COM zu erzeugen
GRAPHIC.PRN	Druckerinterface für den IBM-Graphic Printer
M130I.PRN	Druckerinterface für den M130i
NEC8023.PRN	Druckerinterface für den NEC 8023a
DISKS.CFG	Konfigurationsfile für die Speicheraufteilung
SYSTEM.CFG	Konfigurationsfile für die Floppylaufwerke
ASM.SCR	8088/86 Postfix-Assembler
BLOCKING.SCR	Hilfsprogramm, um physikalische Blöcke in Files zu übertragen
DOUBLE.SCR	Definitionen für doppeltgenaue (32 bit) Zahlen
EDITOR.SCR	Full Screen Editor
EXTEND.SCR	Allgemeine Systemerweiterungen
F83ASM.SCR	Der Assembler aus F83
INSTALL.SCR	Definition der Befehlstasten des Editors
KERNEL.SCR	Sourcecode des volksFORTH Kerns
PRIMED.SCR	Primitivsteditor zur Anpassung an "Fastkompatible" Rechner
SEE.SCR	Der automatische Decompiler
SERIAL.SCR	Die serielle Schnittstelle
TASKER.SCR	Der Multitasker
TIMER.SCR	Der Timer im IBM-PC
TOOLS.SCR	Der manuelle Decompiler, der Tracer und DUMP-Utility

Zuerst sollte versucht werden, das Programm VOLKS4TH. COM zu starten. Meldet sich das System auf dem Bildschirm mit einer Statuszeile in der 25ten Zeile, so ist der Rechner kompatibel genug für den Multitasking Videotreiber MULTI.VID. Sollte jedoch keine Meldung auf dem Bildschirm erscheinen, so handelt es sich leider nur um einen "Fastkompatiblen" Rechner und es ist etwas Arbeit erforderlich, das volksFORTH anzupassen.

Der nächste Versuch, MINIMAL.COM zu starten, sollte eigentlich erfolgreich sein. Wenn das nicht der Fall ist, dann handelt es sich noch nicht einmal um ein ordentliches MS-DOS Betriebssystem, da von dieser FORTH-Version keinerlei Routinen im ROM des Rechners direkt - unter Umgehung des MS-DOS - benutzt werden.

MINIMAL listet nach dem Starten eine kurze Beschreibung des Primitivsteditors, der nun dazu benutzt werden kann, das Loadfile VOLKS4TH.SYS so zu verändern, daß damit ein System mit weniger anspruchsvollem Videodisplaytreiber erzeugt werden kann.

Voraussetzung dazu ist, daß durch das Systemfile CONFIG. SYS (siehe MS-DOS Handbuch) der Devicetreiber ANSI.SYS beim Booten des Systems installiert wurde. Existiert auch dieser nicht, so ist die Benutzung des Full-Screen Editors in der ausgelieferten Form nicht möglich. Ist ANSI.SYS ins System inte-

griert, so kann mit der Anpassung des Forthsystems fortgefahren werden. Nach dem Starten von MINIMAL.COM ist folgendes einzugeben:

USE VOLKS4TH.SYS 1 LIST

Danach kann durch das NEW Kommando (siehe Beschreibung des Primitivsteditors) im File VOLKS4TH.SYS die Zeile, die "include multi.vid" enthält, durch "include ansi.vid" ersetzt werden. Dann muss auch noch im File EDITOR.SCR auf Screen 1 eine entsprechende Änderung vorgenommen werden, indem die Ladeanweisung für das Multitasking Display "auskommentiert" wird und entsprechend die Ladeanweisung für das ANSI-Display aktiviert wird, dadurch, dass in der ersten Spalte der "\" (siehe: Interpreter-Worte) umgesetzt wird.

Danach wird mit BYE ins Betriebssystem zurückgekehrt. Die Befehlszeile

KERNEL INCLUDE VOLKS4TH.SYS

erstellt dann ein neues File VOLKS4TH.COM, das nun die ANSI.SYS Steuersequenzen für die Cursorsteuerung benutzt. Dies dauert einige Zeit, da insgesamt ca. 15kByte Objectcode compiliert werden müssen.

Nach dem Ende des Ladevorgangs meldet sich das System mit einem Piep und es sollte auch in der 25. Zeile eine inverse Statuszeile angezeigt werden.

Nun kann noch mit Eingabe von INCLUDE DISKS.CFG die aktuelle Speicherkapazität der Diskettenlaufwerke festgelegt werden. Dies ist jedoch nur dann notwendig, wenn im DIRECT-Modus (siehe: Fileinterface) auf die Disketten zugegriffen werden soll. Danach evtl. mit SAVESYSTEM VOLKS4TH.COM die Änderungen permanent auf die Disk zurückschreiben.

Damit ist die Anpassung an den Rechner beendet.

Der Videodisplaytreiber BIOS.VID benutzt - wie auch MULTI. VID - den BIOS-call $10. Da jedoch keine Rücksicht auf Multitaskingbetrieb genommen wird (nur eine Task kann den Bildschirm steuern), ist die Bildschirmausgabe schneller.

Als letztes bleibt dann noch die Anpassung an den Drucker vorzunehmen. Dafür gibt es im System bereits die drei Files

GRAPHIC.PRN M130I.PRN NEC8023.PRN.

Sollten Sie einen Drucker haben, der noch andere Steuersequenzen benötigt, so müssen Sie sich für Ihren Drucker auch ein entsprechendes .PRN File schaffen. Als Vorbild kann dabei jedes der drei vordefinierten Files dienen; es sind lediglich die jeweils aktuellen Werte für die Steuersequenzen an passender Stelle einzutragen - dies erfordert jedoch schon einige Vertrautheit mit Forth. Sollten Sie dazu nicht in der Lage sein, so hilft Ihnen vielleicht ein alter Forth-Hase in ihrer Nähe, mit dem Sie sicher die Forth Gesellschaft e.V. in Verbindung bringen kann. Bitte denken Sie auch daran, neue Druckertreiber an die Forth Gesellschaft einzusenden, damit diese in das volksFORTH-Paket aufgenommen werden können.

Der passende Druckertreiber muss dann in dem Loadfile VOLKS4TH.SYS so eingetragen werden, wie das beim Videodisplaytreiber beschrieben wurde.

Hinweise:

Aus dem Handbuch zum volksForth stehen noch diverse Seiten zum Abdruck bereit. Wenn Sie, liebe Leser, diese Seiten gerne in der Vierten Dimsnion lesen wollen, dann melden Sie das bitte der neuen Redaktion (vd@forth-ev.de).

Ich selbst wiederhole meine Anfrage nach einem volksForth für den PDA unter WindowsCE. Ist eine Adaption des volksForth an WindowsCE (keine DOS-Box verfügbar) überhaupt denkbar? Wenn volksForth auf dem PDA, der für mich ein zukunftsweisendes Werkzeug ist, keine Zukunft hat – welches PD-Forth ist dann für diese kleinen Rechner empfehlenswert?

Friederich Prinz

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

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

Forth-Hilfe für Ratsuchende

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

Spezielle Fachgebiete

FORTHchips
(FRP 1600, RTX, Novix)

Klaus Schleisiek-Kern
Tel.: (**0 40) - 37 50 08 03** (g)

KI, Object Oriented Forth,
Sicherheitskritische
Systeme

Ulrich Hoffmann
Tel.: (**0 43 51**) **- 71 22 17** (p)
Fax: **- 71 22 16**

Forth-Vertrieb
vlksFORTH
ultraFORTH
RTX / FG / Super8
KK-FORTH

Ingenieurbüro **Klaus Kohl**
Tel.: (**0 70 44) - 90 87 89** (p)
forth@designin.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.

3;4 / 2005