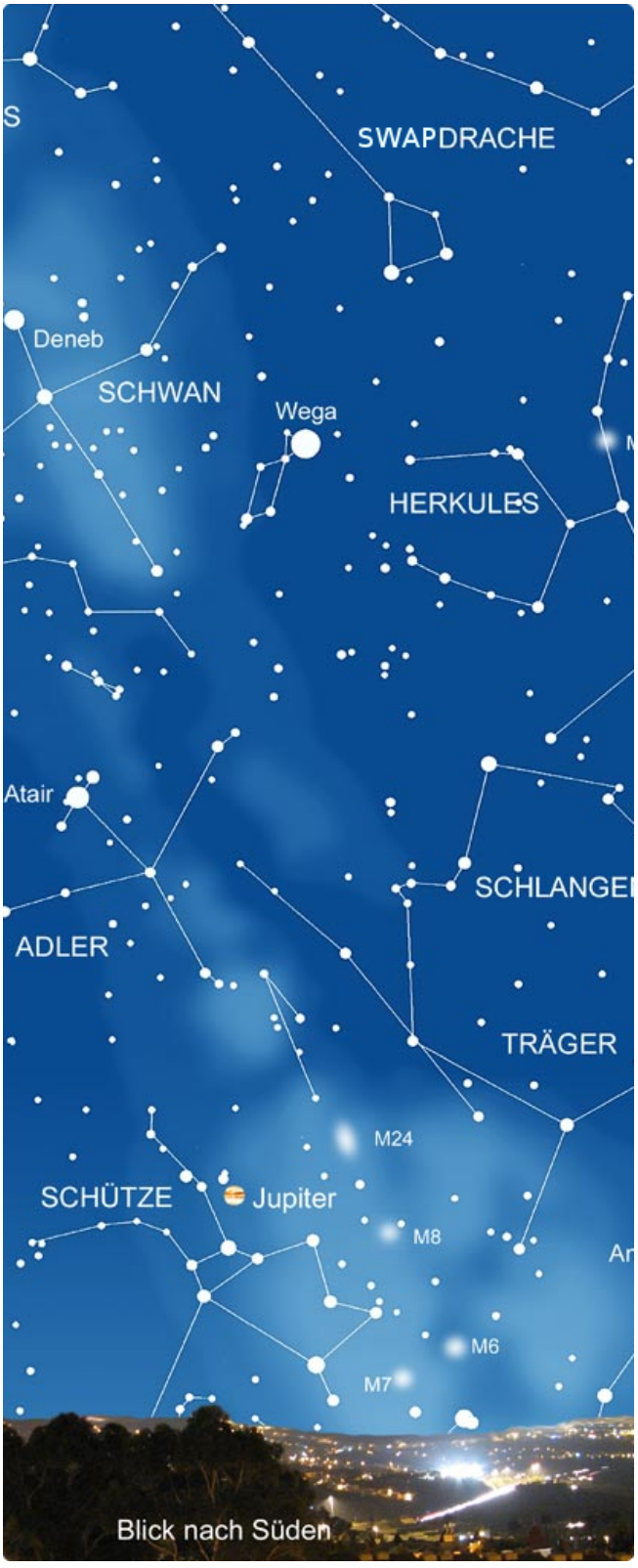
Vierte Dimension

2/2013

Das Forth–Magazin

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



In dieser Ausgabe:

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische Informatik

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

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

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

LEGO RCX–Verleih

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

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

Inhalt:

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

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

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

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

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

KIMA Echtzeitsysteme GmbH

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

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

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

Bergstraße 10 D–18057 Rostock
Tel.: +49 381 496800-0 Fax: +49 381 496800-29

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

Ingenieurbüro
Klaus Kohl–Schöpe

Tel.: **(0 82 66)–36 09 862**
Prof.–Hamp–Str. 5
D–87745 Eppishausen

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

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

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

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00€ + Porto u. Verpackung

Manuskripte und Rechte

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

Liebe Leser,

da Ulrich grad wenig freie Zeit hatte, erklärte er mir schnell, was zu tun ist, um die VD zu machen, half, das Versionsverwaltungssystem *fossil* (`http://www.fossil-scm.org`) zum Zugriff auf unser VD-Repository und die Umgebung für LaTeX unter Windows XP in Betrieb zu nehmen. Beiträge waren schon genug zusammengekommen, und so konnte es gleich losgehen, das Heft zusammenzubauen. Bernd hat dann geduldig alle Fehler, die ich im Satz dabei gemacht habe, wieder ausgebügelt, und pflegt still im Hintergrund die vier Umschlagseiten. Fred hat unermüdlich alle Tipp- und Rechtschreibfehler identifiziert. Druck und Versand haben Ewald und Andrea abgewickelt. Ihr seht, so ein Heft ist Teamarbeit. Ihr könnt mitmachen. Man lernt LaTeX und wie das alles so geht. Die Unterstützung im Verein ist exzellent!

Mein erstes Editorial in einer VD schrieb ich 1986[1]. Damals zogen die Mikroprozessoren in mein Berufsleben ein, und ich suchte nach einer Antwort auf die Frage: „Wie kann so ein Haufen Draht mit mir sprechen?“. Mit dem RSC-Forth von Rockwell im AIM65 konnt' ich das ergründen. Das erste `OK` und `.S` auf dem einzeiligen Display war dann eine aufregende Sache — Der Kasten redete mit mir! Inzwischen kann man sich seinen eigenen Forth–Prozessor machen — der STRIP32 von Willi Stricker zeigt, wie. DIY-(Do It Yourself)-Projekte überall auf der Welt zeigen, dass Forth lebt. Marc Malmrose benuzt es für Atmel-Komponenten in seinem Astrographen. Eine MCU wie der MSP340G2553 ist zusammen mit der 4E4TH-IDE von Dirk Brühl inzwischen wirklich „plug&play“ — USB einstecken, ein paar Klicks, schon ist Forth als Betriebssystem in die MCU geladen und über ein Terminal bedienbar, erweiterbar — und das für wahrlich kleines Geld — den PC mal vorausgesetzt. Man kann damit sehr einfach studieren, wie alles geht. Wie einfach das ist, zeigt Matthias Koch im LEDCOMM. An dieser Stelle sei noch mal auf den Mikrocontroller-Verleih der Forth-Gesellschaft hingewiesen: Hier könnt ihr alles ausleihen, was es zum Experimentieren so braucht, bis hin zum ARM. Auf der Ebene der PCs könnt ihr zusammen mit Hannes Teich und der Wave Engine in die Welt der Klangerzeugung eintauchen. Er benutzt gforth dazu, und lädt ein, mitzumachen.

Viele sind wir ja nicht hier im Forth-Vereinsleben, aber wir kommen gerne zusammen jedes Jahr. So zuletzt wieder in Garmisch-Partenkirchen im April des Jahres. Und als Nächstes? Vielleicht sehen wir uns ja bald auf der *Maker Faire* in Hannover (3. August 2013), auf der wir einen Stand haben.

Vielen Dank an alle Autoren und viel Vergnügen bei der Lektüre des Heftes,
Euer Michael

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
`http://fossil.forth-ev.de/vd-2013-02`

Die Forth–Gesellschaft e. V. wird durch ihr Direktorium vertreten:
Ulrich Hoffmann Kontakt: `Direktorium@Forth-ev.de`
Bernd Paysan
Ewald Rieger

[1] Vierte Dimension, April 1986, Vol2/1, S.3

ST Robotics und Forth

In Cambridge in die Schule zu gehen, scheint Spaß zu machen. Jedenfalls, wenn das alles so wahr wird, wie angekündigt: Schools Robotics Initiative - and Forth.

Wie es scheint, hat die Firma *ST Robotics* ausdrücklich Forth in die Initiative mit hineingepackt, und das schon lange. Nun sponsert sie ein Schulprogramm. Und dann ist da ja auch noch deren Roboterarm, den du über ein Hyperterminal fernsteuern kannst, derweil dir die Webcam im Browser zeigt, was passiert.

Schools Robotics Initiative: `http://forth-robotics.org/`
Die Firma dahinter: `http://www.strobotics.com/`
Und was man sehen kann: `http://www.strobotics.com/webcam.htm`

Viel Vergnügen. mk

ConnecTIng Wirelessly

Da komme ich aus dem Staunen nicht mehr raus! Wer mal sehen möchte, was da so alles geht auf einem Launchpad, klicke sich mal rein in die Welt der Bluetooth-apps in der *TI E2E Community* - z.B. zum Quadcopter von Trey German (der Mann heißt wirklich so :)

Und so kommt man dahin:
TI Home » TI E2E Community » Blogs » ConnecTIng Wirelessly » MCU Quadcopter BoosterPack blasts off – utilizing TI's CC2560 Bluetooth and CC4000 GPS solutions to take the design to new heights

mk

Buch: Applikation von Forth

Volker Pohlers hat das Buch „Applikation vom Forth“ aus dem Jahr 1987 eingescannt und als PDF zum Download bereitgestellt. Das Buch enthält Artikel von Egmont Woitzel und Ralf Neuthe und wurde herausgegeben von Prof. Hartmut Pfüller an der Wilhelm-Pieck-Universität Rostock. Herr Pfüller hat einer Veröffentlichung auf den Web-Seiten der Forth-Gesellschaft zugestimmt, das Buch steht nun jedem Interessierten unter `http://forth-ev.de/filemgmt/visit.php?lid=501` zum Download zur Verfügung. Vielen Dank an Volker Pohlers für den Scan des Buches.

Viel Spaß beim Lesen. mk

Reforth - Rethinking Forth

Reforth ist ein neues Forth-System von Sean Pringle. In Reforth versucht Sean einige (aus seiner Sicht) Probleme in klassischen Forth-Systemen zu lösen. Reforth ist noch in der Entwicklung, aber schon ein brauchbares System.

Worin unterscheidet sich Reforth von anderen Forth-Systemen?

- viele Kontrollstrukturen werden mit dem Wort END abgeschlossen (anstatt mit THEN, LOOP, AGAIN oder REPEAT)
- geschachtelte Wortdefinitionen (Sub-Words), welche auch zum Aufbau von Programmbibliotheken verwendet werden
- nur zwei feste lokale Variablen
- Wörter zum Ablegen von strukturierten Daten (Records), welche im Zusammenspiel mit Sub-Wörtern zu Objekten werden
- Null-terminierte Strings per String-Literale
- Posix-Reguläre Ausdrücke

Keiner dieser Funktionen ist wirklich neu oder revolutionär, aber Reforth bildet ein interessantes System, welches Beachtung verdient. Der Reforth-Compiler wandelt derzeit den Forth-Quellcode in C-Quellcode, welcher dann per *GNU CC* oder *CLANG* übersetzt wird.

Als Beispielanwendung findet sich ein Editor, welcher an den Unix “vi” Editor angelehnt ist und Syntax-Highlighing für verschiedene Programmiersprachen beherrscht.

Eine ausführliche Beschreibung der Besonderheiten von Reforth findet sich in englischer Sprache unter `http://aerosuidae.net/reforth/`, der Quelltext findet sich auf GitHub: `https://github.com/seanpringle/reforth`.

cas

MicroCore

Seit Anfang des Jahres entsteht an der Fachhochschule Nordwestschweiz in Windisch ein C-Compiler für uCore (MicroCore) auf der Basis von LCC. Das uCore-Backend ist inzwischen fertiggestellt. Zur Zeit wird an der Stackallokation auf der Basis der Forschungsarbeiten von Phil Koopman, Mark Shannon und Chris Bailey gearbeitet. Der LCC-Compiler erzeugt als Output Forthcode, der dann vom Forth-Crosscompiler in Objektcode umgesetzt wird. Als Testbed wird ein kleines FPGA-Board mit dem XP2-8 FPGA von Lattice benutzt, für das uCore mit 24 und 32 Bit Datenwortbreite generiert wurde. Nach Fertigstellung des C-Compilers werden kryptografische Bibliotheksfunktionen realisiert, die mit Spezialinstruktionen auf uCore flott gemacht werden sollen. Das Projekt wird von der Hasler-Stiftung (Schweiz) `www.haslerstiftung.ch` unterstützt. :) Klaus Schleisiek (Hamburg)

Forth-Gesellschaft e.V.
Ordentliche Mitgliederversammlung
21.04.2013

Carsten Strotmann

Sitzungsort: Forsthaus Graseck, Graseck 4, Garmisch–Partenkirchen.

Carsten Strotmann übernimmt es, das Protokoll zu führen.

19 stimmberechtigte Teilnehmer. Direktorium: Ewald Rieger, Ulrich Hoffmann, Bernd Paysan.

2013-04-21 Sun 09:14 **Wahl des Versammlungsleiters: Heinz Schnitter**

2013-04-21 Sun 09:18 **Ergänzungen zur Tagesordnung**

2013-04-21 Sun 09:18 **Bericht der Verwaltung Wirtschaftsjahr 2012**

- Mitglieder Ende 2012: 111
- Gute Zahlungsmoral in 2012.
- Leichter Mitgliederrückgang in 2012.
- Durchschnittsalter der Mitglieder: 55,8
- Forthbildung (Teilnahme an Messen etc) schlägt sich leider nicht in neuen Mitgliedern nieder.
- 2012: 2 Eintritte, 6 Austritte. 2013: 2 Austritte.
- Überprüfung der Gemeinnützigkeit 2008-2011: die Forth-Gesellschaft ist laut Bescheid vom 26.11.2012 von der Gewerbesteuer befreit, für weitere 5 Jahre gemeinnützig.
- Einnahmen ideeller Bereich: 2.116,00 €
- Ausgaben ideeller Bereich: -2.744,18 €
- Vermögensverwaltung: -166,18 €
- Einnahmen aus dem Zweckbetrieb: 2.214,26 €
- Ausgaben Zweckbetrieb: 3.069,67 €
- Einnahmen/Ausgaben: -1.649,77 € (Verlust)
- Vereinsvermögen 31.12.2012: 11.096,65 €
- Wirtschaftsplan 2013: 3.900 € Einnahmen, 6.650 € Ausgaben, Vermögen 31.12.2012: 8.346,00 € (Plan)

2013-04-21 Sun 09:42 **Kassenprüfer Gerald Wodni hat die Kasse geprüft,**

die Kassenführung ist korrekt gefunden worden, Gerald gibt die Empfehlung, den Vorstand zu entlasten.

2013-04-21 Sun 09:45 **Ulrich Hoffmann - Bericht von der Vierten Dimension und der Webpräsenz**

- 4 Ausgaben mit verteilten Editoren und Chefredakteuren.
- Umstellung des Repositorys auf Fossil.
- Webpräsenz: weitgehend problemlos, der Mailinglisten-Server ist wieder repariert und funktioniert, VD, Tagungs– und Mitglieder-Mailinglisten wurden neu eingerichtet, das Backup wurde repariert, besseres Backupsystem wird angedacht.

2013-04-21 Sun 09:50 **Bernd Paysan - Projekte:**

Chemnitz 2012, Augsburg 2012 Biesterwettbewerb, VCFe 2012 Benchmarkwettbewerb, LinuxTag 2012, Unfuck 2012, Chemnitz 2013, Augsburg 2013.

2013-04-21 Sun 09:56 **Entlastung des Vorstandes: einstimmig angenommen**

2013-04-21 Sun 09:58 **Verleih des Drachenpreises - neuer Drachenhüter ist Gerald Wodni**

2013-04-21 Sun 09:59 **Kaffeepause**

2013-04-21 Sun 10:31 **Wahl des Direktoriums**

- Frage, ob die Wahl zusammengefasst werden kann (alle Mitglieder des Direktoriums möchten wieder kandidieren).
- Egmont Woizel stellt den Antrag, dass das alte Direktorium wiedergewählt wird: Ulrich Hoffman, Bernd Paysan, Ewald Rieger.
- Bei der Wahl wird das Direktorium einstimmig angenommen.
- Die Gewählten nehmen die Wahl an.

2013-04-21 Sun 10:35 **Verschiedenes:**

Makerfaire

- Carsten organisiert über die Mailingliste.
- Meldungen für Standbesatzung: Ulrich Hoffmann, Michael Kalus, Gerald Wodni, ...
- Anmeldung 15. Juni

Forth-Tagung 2014 - 30 Jahre Jubiläum

Ort: Bad Voslau (Organisator: Gerald Wodni)

Editoren für VD 2013-2014

- Ulrich Hoffmann, Bernd Paysan , Carsten Strotmann -> Umfrage auf der Mitgliedermailingliste nach neuen Mitarbeitern.
- theForthnet -> eine neue wiederkehrende Kolumne über neue Pakete und Quellcode.
- Geplant: Je ein internationaler englischer Artikel pro Ausgabe.

LinuxTag 2013

- 22.-25.5.2013 Berlin - Teilnahme beschlossen.
- Blickfang Tricepts Roboter - GO gegen einen menschlichen Spieler.

neuer VM-Webserver (Netcup)

Neuer E-Mail-Alias "internet@forth-ev.de", um alle Aktiven rund um den Internetauftritt zu erreichen.

Projekt ARM-gForth (weitere Informationen kommen in der VD in den kommenden Ausgaben)

- Stellaris-Launchpad (Michael Kalus) - ARM-board
- gforth-ec (Bernd Paysan)
- Stellaris-Launchpad für den Mikrokontroller-Verleih (Carsten)
- gnublin (Carsten Strotmann)
- Rasberry Pi
- Beagle-Board
- CubieBoard (Carsten Strotmann)

2013-04-21 Sun 11:40 **Ende der Mitgliederversammlung**

Amforth und Python — SPI zum Raspberry Pi für einen Astrographen

Mark Malmros

In dem Projekt lasse ich gerade meinen Astrograph aufleben. Der braucht 9 (neun) Taktvorgaben für die Steuerung, und man muss 12–Bit–ADC in 3 Nibbles aus dem 755x242 CCD–Array (TC245) holen — und eine AD–Wandlung benötigt etwa 10 Mikrosekunden. Da muss man sich sputen. An dieser Stelle kam Amforth als Retter in der Not gerade recht, zusammen mit einer Lektüre darüber, wie das SPI-System arbeitet.

Mehr an Hardware als an Software orientiert, passt Forth zu meinem Gehirn, so ist das nun mal! Ich habe mit einer Vielzahl von Embedded Forths gespielt — und auch Basteln mit FPC genossen. Einer meiner Favoriten war Frank Sargents pygmy Forth — und er schrieb ein 3–Anweisungs–Forth für den HC11, das viel Spaß gemacht hat — op–Codes in den Chip schieben mit einem DIY Assembler! Eintauchen in Python, C, Javascript etc. Aber ich vermisste ein bequemes Forth — dann, voila!, Amforth. Ich bin mehr ein Tüftler als ein ernsthafter Programmierer. Obwohl es auch Spaß macht, mal in den avr-Assembler zu springen — auch wenn ich mich darin manchmal etwas verloren fühle.

Bilddaten sammeln

Mein Projekt benötigte einige kritische Timing–Funktionen für die Datenerfassung, und high level system, um die Daten zu verarbeiten. Ich wollte einen ATMEGA328P für das Timing und die Akquisition verwenden, sowie einen Raspberry Pi mit Python und eine Vielzahl von Bildverarbeitungs–Modulen. Aber wie sollten die Daten schnell aus dem Atmega an Pi übermittelt werden? Als ich mich damit näher befasste, war ich wirklich zunächst entmutigt vom SPI wegen dessen Timing–Problemen. Und TWI (I2C) ist viel zu langsam.

Es gibt drei Python–C–Wrapper auf niedrigem Niveau für das Linux-spidev–Modul — aber wie kann man den Raspberry Pi als Slave konfigurieren? Ich fand einige Informationen darüber, wie man das Kernel-Modul umbaut und recompiliert. Speichern und Abrufen des SPI auf der forthigen Seite, und Herumspielen mit einem Python–SPI–Modul auf der Pi–Seite verhalf mir zum Verständnis des atmega–SPI–Systems, mehr als die Dokumente oder irgendwelche Beiträge es vermochten. Stell dir vor, das in C zu machen.

SPI zur Übertragung verwenden

Der einfachste Weg, um sich ein SPI–System vorzustellen, ist als *Drehtür*. Der Master macht das Drehen und bestimmt die Geschwindigkeit. Was an der einen Seite hinein geht, kommt an der anderen Seite heraus. Und es *dreht* auch mit Null–Bytes genauso. Mit diesem Sinnbild im Kopf gibt zwei Möglichkeiten, um die Aufgabe zu erfüllen.

Zum Einen können wir auf dem Raspberry Pi als Master–Seite den zugeteilten Puffer im py–spidev–Modul erweitern und das neu kompilieren. Da ich jedes serielle Register der 242 + 10 Pixel aus dem CCD in 12 Bit pro Durchgang lese, speichere ich das im SRAM (504 Bytes) und ziehe diesen Puffer zwischen den Durchgängen mit einer Python–Schleife (*streaming down*) herüber, mit einem Test für das Null-Byte zwischen den Durchgängen. Da man das SPI–Register in Assembler schnell anhalten kann, funktioniert dies gut, und die Latenz durch die Prüfung auf das Null–Byte auf der Master–Seite ist minimal - und es scheint, das Anhängen der resultierenden Python–Liste in ein Array ist ein simpler Kopierprozess. Mit diesem Ansatz können wir 380520 bytes in 755 Zyklen in unter 1,5 Sekunden aus der CCD transferieren. Wobei der 328P mit 20MHz getaktet ist, und der Pi SPI bei 3MHz läuft - das ist mehr als akzeptabel - für einen gekühlten Chip! Was es mit der Kühlung auf sich hat, ist weiter unten erklärt.

Andererseits: Da wir nur 12 Bits benötigen, aber in Bytes übertragen, kann man einen *bit shift* machen und das LSB nehmen. Dann einfach alles in den Puffer auf das Pi strömen lassen, wieder die Null-Bytes filtern, und die Bit-Verschiebung umkehren. Wenn man nichts in das SPDR Register tut, geht es zurück auf das Pi als Null-Byte. Und das Pi verfügt über 512 MB RAM, wovon das Meiste nichts tut, da ist Platz genug. Dieser Ansatz könnte die Auslese-Zeit unter eine Sekunde drücken, aber zu einem *Jitter* in der Taktung führen bei der Überprüfung des SPIF-Flags im atmega. Da bin ich mir nicht sicher, ob dies ein Problem für den CCD ist.

Ich habe auch ein Amforth das sich mit meinem Pi unterhält, nur über den UART, für eine automatische Nachführung des Teleskops (Autoguider). Dafür wird eine USB–Webcam, angeschlossen am Raspberry Pi, benutzt für die Verfolgung von Objekten am Himmel, und die Bildverarbeitung geschieht in Python, und liefert Korrekturen für die Stepper für den äquatorialen Antrieb des Teleskops. Da diese tasks im Vordergrund laufen, sind Interrupts im amforth ausgeschaltet.

Ich habe auch Pläne, dieses System dazu zu verwenden, Mosaike aus den CCD–Aufnahmen zu machen.

Etwas zu den Grundlagen der Bilderfassung mit dem Astrographen

Es gibt verschiedene physikalische Arten von CCD. Dieser hier hat zwei 2D–Arrays, eines ist dem Licht ausgesetzt, das andere nicht. Die vom Licht induzierten akkumulierten Ladungen aus dem Array werden sehr schnell in das maskierte, das *dunkle Array*, übertragen, Zeile für Zeile, das dauert etwa 250 Takte — das Datenblatt bezeichnet dies als *Frame–Transfer* — wobei *frame* das ganze 2D–Array meint. Es ist also im Wesentlichen ein elektronischer *Shutter*. Von dort wird dann Pixel für Pixel ausgelesen, 12 Bits pro Pixel für die Analog-Digital–Wandlung. Kühlen des CCD reduziert thermische Elektronen und verbessert damit das Signal–Rausch–Verhältnis.

Ungekühlt wäre der CCD schon in etwa 1 Sekunde oder weniger gesättigt, also überbelichtet bei Raumtemperatur und normalem Tageslicht. Bei etwa -30°C kann die CCD für mehr als 30 Minuten völliger Dunkelheit ausgesetzt werden, ohne allzu viele thermische Elektronen zu akkumulieren, so haben wir wenig Rauschen. Für die Abbildung sehr schwacher Sterne mit Langzeitbelichtungen ist niedriges thermisches Rauschen das Ziel. Und je schneller man das Bild aus dem Array heraus bekommt — sobald es belichtet ist — umso besser. Unter dem Strich gilt, wenn das Gerät abgekühlt ist, darf es einige Sekunden dauern, bis das Bild übertragen ist, ohne den Verlust des Signals durch Rauschen befürchten zu müssen. Wenn die Ladung ausgelesen wird, wird sie auch doppelt korreliert, vorher und nachher, gegen die Referenzspannung im Bauteil. So dass die 12–Bit–Analog–Digital–Umwandlung ziemlich konstant wird über die Bilder hin.

Ich baue dies, um quantitative photometrische stellare Magnituden von veränderlichen Sternen zu ermitteln, in der Hoffnung, ein hochwertiges Instrument zu haben, wenn dieses Projekt fertig ist.

In der Linkliste findet ihr ein Link zu einigen CCD–Aufnahmen, aufgenommen mit einem System ähnlich dem, was ich baue, mit dem TC245 CCD–Chip.

Da gibt es ein Bilder–Paar, das eine auf 16,1 begrenzte Magnitude hat, 30 Sekunden belichtet — ungekühlt!

Die Gates, oder vielmehr die Anordnung von Gates auf dem ladungsgekoppelten Bauelement (CCD) brauchen ungewöhnliche Spannungen - in diesem Fall -2,5 Volt für den *low* und 9,5 Volt für den *high* Pegel. Darum gibt es spezielle Takt-Treiber (DS0026), um das CCD zu schalten, daher die extra Treiberplatine. Zusätzlich brauche ich da ziemlich schnelle Anstiegszeiten, weswegen die Signale vom ATMEGA328P noch durch Schmitt–Trigger–Buffer in die Treiber gehen. Das ist auch ganz in der Nähe des CCD montiert, um die Leitungen von dort bis in den ATMEGA328P und das Pi so kurz wie möglich zu halten.

Im Abschnitt mit den Bildern ist das hoffentlich gut zu sehen.

Anmerkung: Wie dieser Beitrag zustande kam.

Am 13.03.2013 um 13:26 schrieb Mark Malmros in der *Amforth devel mailing list*:

> Danke für die Antwort, Mattias und Enoch.
>
> Das mit dem ZH:ZL–Register hat geklappt, und den Index in „unbenutztes“ SRAM zu verschieben, hat auch geholfen – es funktioniert! Und vielen Dank für den Hinweis auf das push–pop–Paar! (duh!) Diesen Hinweis kann ich brauchen, wird in einer anderen Schleife auch benötigt für ein weiteres Register.
>
> Und Enoch — ich schätze auch deine Antwort. Du scheinst einen scharfsinnigen Instinkt für meine Herausforderung zu haben. ...

(Neugierig geworden, um was es denn ginge, fragte ich nach, und Mark schilderte sein Projekt näher. Und wir kamen überein, dass ein forthig gesteuerter Astrograph unbedingt in die VD gehört. Ich bin gespannt, wie es damit weitergeht. Michael)

Mark lebt in Vermont, USA, am Lake Champlain — Chimney Point. Dort gebe es immer noch „eine sehr niedrige Lichtverschmutzung“.

Links

`http://astro.uchicago.edu/RAS/rchive/ccdimages.html` CCD-Aufnahmen
`https://github.com/doceme/py-spidev` Quellcode
`http://www.ti.com/product/tc245`
Datenblatt: 786- X 488-Pixel CCD Image Sensor (Rev. A)
`http://de.wikipedia.org/wiki/Serial_Peripheral_Interface` `http://en.wikipedia.org/wiki/Shutter_(photography)`
Shutter-Technik in der Photographie

Übersetzung: mk

Der Astrograph in Bildern

Abb. 1 zeigt das Raspberry Pi zusammen mit dem Amforth–Autoguider. Montiert auf der Mittelachse am Stativ ist ein ATMEGA328P–Steckbrett, das *shield*, auf einem Raspberry Pi huckepack, mit einem wifi–Adapter für SSH. Da ist auch das Darlington–Array für die Schrittmotor–Spulen mit einem 12V–Regler für deren Stromversorgung und einem 5V–Regler für das Pi. Der ATMEGA328P wird aus dem 3,3V–Bus des Pi mitversorgt, wodurch auf Pegelwandler verzichtet werden kann.

Python PIL und Numpy verarbeiten die Bilder von einer USB–Webcam auf einem kleinen Leit-Teleskop, um die Bahnkorrekturen zu machen.

Das Pi ist zudem als ISP (internet service provider) konfiguriert, durch seine GPIO–Pins.

Dank an Gordon Henderson, der den gepatchten avrdude für das Raspberry Pi beigetragen und compiliert hat; siehe

`http://project-downloads.drogon.net/files/`

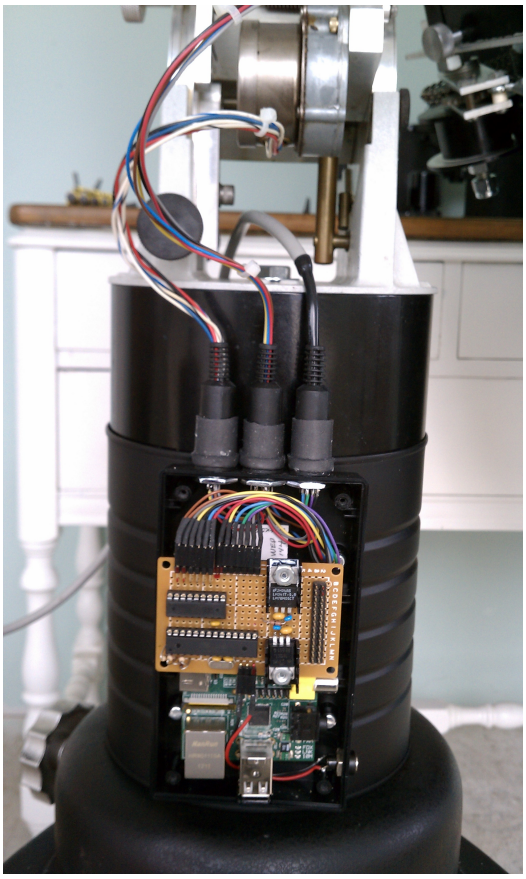


Abbildung 1: Raspberry Pi plus Amforth-Autoguider

Abb. 2 zeigt das CCD–Hardware–Gehäuse. Es ist noch unvollendet; sobald die Software komplett und getestet ist, bis auf die CCD–Chip–Anschlüsse herunter, wird das Gehäuse eloxiert werden und montiert. Der CCD sitzt da auf einer Kühlrippe mit einem zweistufigen thermoelektrischen Chip (Peltier–Vorrichtung) mit einer kanalisierten Rückenplatte für zirkulierende Kühlflüssigkeit. Das Gehäuse wird noch hermetisch abgedichtet, um Beschlagen beim Laufen des Geräts bei etwa -30 C° zu reduzieren, was mit einem I2C–Temperatur–Chip überprüfbar sein wird. Die Treiberplatine wird auf der Seite des Gehäuses montiert.

Und in **Abb. 3** ist der 10″–F5–Newtonian–Astrograph als Ganzes zu sehen, das 10″-Teleskop, und die noch unvollendete CCD–Kamera in einem 2″-Okularauszug. Es wird auch Schläuche haben, die aus dem CCD-Gehäuse herausführen, für das zirkulierende Kühlmittel des thermoelektrischen Kühlers hinter dem CCD. Das schwere Stativ stammt von einer TV–Kamera aus den frühen Tagen des Fernsehens.

Abbildung 3: Der 10″–F5–Newtonian–Astrograph

Zum Listing

Gezeigt wird ein Assembler–Code–Snippet (Textdatei) für die Amforth–Pi–SPI–Verbindung. Diese ist stark kommentiert. Eine einfache Python-Funktion zur Überprüfung der Schleife befindet sich am Ende des Assemblercodes. Die Abkürzungen in den Kommentaren — SRG, A0, A1, ADC usw. — beziehen sich auf die CCD–Taktsignale.

Listing

```
1   ; serial_fetch.asm  "line@" >> "serial@"
2   ; compiled into Amforth 5.0 m328p
3   ; here were attempt to push 252 x 2 bytes of data by spi with one start/stop call
4   ; this code compiles as of 3/12/2013 m.malmros@gmail.com
5   ; "serial@" does one complete serial register transfer and moves 252 words as 504
6   ; bytes into the Pi
7   ; uses the Z registers to index into and out of SRAM starting at $0300 and
8   ; going up plus $01F8 (504) bytes
9   ; "serial@" should be integrated into an outer loop that steps each horizontal
```

Abbildung 2: CCD–Gehäuse–Hardware

```
; line into the serial register gates
; this can either be binned or unbinned
; gate clocking - esp. ADC on time - needs further adjustments
; note "serial@"
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
VE_SERIALFETCH:
    .dw $ff07  ; allocate bytes for the ".db" ?? (serial@ = 7)
    .db "serial@",0
    .dw VE_HEAD
    .set VE_HEAD = VE_SERIALFETCH
XT_SERIALFETCH:
    .dw PFA_SERIALFETCH
PFA_SERIALFETCH:
    ldi temp0, $FC ; load register with loop count of 252
    ldi temp1, 0b0011_1111 ; set up pins 5 thru 0 PORTC for output
    out DDRC , temp1
    ; set up indirect counter(x) for SRAM transfers
    clr r30 ; Clear Z low byte
    ldi r31,$03 ; Set Z high byte to $03 (hence we start at $0300 in SRAM)
    ldi temp1, 0b1111_0000 ; AND bit to mask pins 3 thru 0
    mov temp4, temp1        ; temp4 is r14 and thus ldi can't be used
PFA_SERIALLOOP:
    ldi temp1, 0b0000_0101
    out PORTC, temp1    ; A1 goes low ADC returns to hi stopping conversion
    ldi temp1, 0b0010_0101
    out PORTC, temp1    ; SRG1 goes hi
    in temp6, PIND      ; read nibble
    AND temp6, temp4    ; mask low nibble
    ldi temp1, 0b0000_0011
    out PORTC, temp1    ; A1 goes hi, A0 goes low, SRG1 goes low
    SWAP temp6          ; shift bits to the right
    ldi temp1, 0b0001_0011
    out PORTC, temp1    ; SRG2 goes hi
    in temp7, PIND      ; read nibble
    AND temp7, temp4    ; mask low nibble
    ldi temp1, 0b0000_0111
```

```
46      out PORTC, temp1    ;A0 goes hi, SRG2 goes low
47      ldi temp1, 0b0000_1111
48      out PORTC, temp1    ;SRG3 goes hi
49      add temp6, temp7    ; temp6 has LSB as result so temp7 can be trashed over
50      ldi temp1, 0b0000_0111
51      in temp7, PIND      ; temp7 has MSB
52      AND temp7, temp4    ; mask low nibble
53      out PORTC, temp1    ; SRG3 goes low
54      ldi temp1, 0b0000_0110
55      SWAP temp7          ; shift bits to the right 4 places !
56      out PORTC, temp1    ; ADC goes low starting conversion
57  ; ADC conversion may take up to 10 microseconds, so nops need to be filled in here
58  ;;;;;;;;;;;;;;;;;;;;;;;;  move  our two bytes (from temp6 and temp7) into SRAM
59      st Z+,temp6 ; Store temp6 (low byte in data space loc. $0100(X post inc)
60      st Z+,temp7 ; Store temp7 (high byte in data space loc. $0101(X post inc)
61  ; 504 bytes ($1F8) are stored in SRAM from $0100 to $02F8
62  ;;;;;;;;;;;;;;;;;;;;;;;;;
63      dec temp0
64      brne PFA_SERIALLOOP
65  ;;;;;;;;;;;;;;;;;;;;;;;;;; SPI XFER SRAM OUT 504 BYTES ;;;;;;;;;;;;;;;;;;;;;;;;;;
66      ldi temp3, $03  ; high ... ZH
67  ;   "pull" serial line from SRAM to the pi
68      ldi temp1 , $FF
69      out SPDR, temp1 ; load SPI data register with $ FF as start sig to pi python script
70      ldi temp1, (1<<SPE) ; or use $40 for turning on SPI system as slave
71      out SPCR, temp1 ;
72      SPI_STREAM:
73          in r16, SPSR    ; 1 cycle - check status register for interrupt flag
74          sbrs r16, SPIF  ; 1 cycle to skip -  if bit is set skips the next instr
75          rjmp SPI_STREAM ; ( loop to check bit)
76          ld temp6,-Z     ; 2 + 2 cycles when skipped to -- load register
77                          ; with last byte stored in SRAM - LIFO
78          out SPDR, temp6 ; 1 cycle - load SPI data register with our byte
79          cp r31, temp3   ; 1 cycle -- checks hi byte (r31) of Z register
80          brsh SPI_STREAM ; 1 or 2 cycles, brsh-branch if same or higher (unsigned)
81      clr temp1       ; turn off the SPI system
82      out SPCR, temp1
83      out SPDR, temp1 ; ensure that the SPI data register is clear (null)
84      jmp_ DO_NEXT
85  ;;;;;;;;;;;;;;;;;;;;;;;;;; TIMING ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
86  ; timing - this loop can consume 9(?) cycles to load data when SPIF is detected
87  ; assume about 1 microsecond per loop (@ 20Mhz)
88  ; 10 bits per micro (10 clocks cycles driven by the Pi) = mhs of 10 Mhz
89  ; if clk = 4Mhz (4M bps ?) , shifting 8 bits takes 2 micro seconds
90  ; 2 micro seconds per byte x 504 = 1008 micro or 1 ms per line
91  ; 755 lines = 755 ms or 3/4 second at 2 Mhz would take 1.5 seconds
92  ;;;;;;;;;;;;;;;;;;;;;;;; python function to test ;;;;;;;;;;;;;;;;;;
93  ; py-spidev
94  ; adjust MAXBUFFER in spidev.o and re-compile
95  ; import spidev
96  ; a= spidev.SpiDev(0,0)
97  ; a.mode = 0
98  ; a.max_speed_hz = 3000000
99  ; image_array = []
100 ; def spiloop():
101 ;     global image_array
102 ;     while len(image_array) < 755 :
103 ;         abyte = a.readbytes(1)[0]
104 ;         if abyte == 255 :
105 ;            image_array.append(a.readbytes(504))
```

Ledcomm — den Glühwürmchen nachempfundene Kommunikation zwischen zwei Leuchtdioden

Matthias Koch

Ledcomm verwandelt Leuchtdioden in serielle Schnittstellen, die sich wie Glühwürmchen zublinkern und so verstehen. Dabei wird die gleiche Leuchtdiode abwechselnd sowohl zum Leuchten als auch zum Beobachten und Empfangen eingesetzt. Dies ist möglich, da Leuchtdioden gleichzeitig auch Photodioden sind - die bei roten und gelben Leuchtdioden besonders für die eigene Wellenlänge empfindlich sind. Bei grünen und blauen Leuchtdioden unterscheidet sich das Emissions- und Absorptionsspektrum, so dass diese ihre eigene Farbe nicht sehen können. In diesem Artikel soll ein Algorithmus vorgestellt werden, der diesen Effekt zur Kommunikation ausnutzt, gewürzt mit einigen Erklärungen und Experimenten. Viel Spaß !

Material und Methoden

- Zwei Launchpads mit MSP430G2553
- Zwei gleiche superhelle rote Leuchtdioden im klaren Gehäuse. Ich benutze die *LED 5-4500 RT* von Reichelt, eine sehr helle rote Gallium-Aluminium-Arsenid-Leuchtdiode von Kingbright in klarem 5 mm Plastikgehäuse, kleinem Abstrahlwinkel, 4500 mcd bei 660 nm. Auch ein Paar superhelle gelbe Leuchtdioden kann funktionieren, mit infraroten, grünen und blauen Leuchtdioden hat es bislang nicht geklappt.
- Zwei Vorwiderstände um 100 Ohm, optional für die Vorsichtigen.

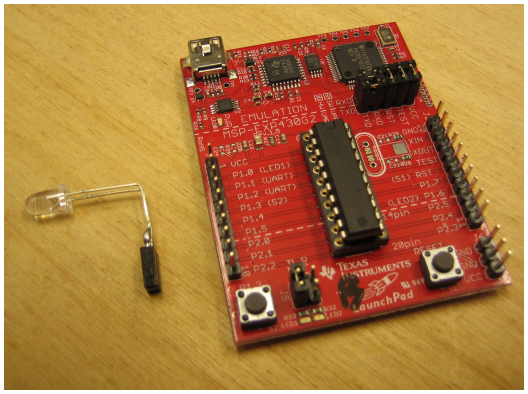


Abbildung 1: Die Ausrüstung: 2x (1 rote LED, 1 LaunchPad)

Schneller Sprung ins Wasser

Die beiden MSP430G2553 werden mit *mecrisp-2553-with-basisdefinitions-launchpad.hex* gebrannt, diese Datei ist unter [1] zu finden. Anschließend wird je ein Terminal geöffnet und der hier abgedruckte Ledcomm-Quelltext eingetippt oder nachgeladen [2]. Wo die hellen roten LEDs angeschlossen werden, ist im Quelltext einstellbar, momentan ist P2.0 die Anode und P2.1 die Kathode. Wichtig sind kurze Leitungen und saubere Kontakte! Die LEDs können auch direkt an die Pins vom Launchpad geklemmt werden.

Das ist alles. Jetzt kann mit `ledcomm` ein Chat-Programm gestartet werden. Die frisch angeschlossene Leuchtdiode leuchtet (blinkt ganz, ganz schnell) und wartet auf eine Verbindung. Zum allerersten Ausprobieren sollten die beiden superhellen Leuchtdioden Kopf an Kopf gehalten werden, so dass sie direkt in ihr Gegenüber scheinen. Im Terminal wird dann eine erkannte Verbindung gemeldet, eingetippte Buchstaben werden auf die Gegenseite übertragen. Die kleinen, auf dem Lauchpad aufgelöteten LEDs zeigen das Laufen des Chatprogrammes in grün und eine bestehende optische Verbindung in rot an. Werden die beiden ineinanderscheinenden Leuchtdioden voneinander entfernt, reißt in einem bestimmten Abstand die Verbindung wieder ab, was im Terminal kundgetan wird.

Eine Antistatikmatte darunter oder ein Finger an den Kontakten entlädt die winzige Kapazität und verhindert das Gelingen. Sind vielleicht Anode und Kathode vertauscht ? Umgekehrt leuchtet die LED nämlich auch, aber viel schwächer. Wenn es jedoch in einem dämmrigen Raum mit kurzen Leitungen, sauberen Kontakten und bei Sichtkontakt „Kopf an Kopf, Auge in Auge“ trotzdem nicht geht, dann sind die LEDs nicht geeignet.

Vom Innenleben der Leuchtdioden

Eine Leuchtdiode leuchtet - das ist ihr wohlbekanntes und oft genutztes Talent. Doch sie ist auch eine Photodiode, deren Photostrom mit einem Operationsverstärker oder einem empfindlichen Multimeter gemessen werden kann. Für die ganz einfache Helligkeitsmessung an einem Microcontroller gibt es aber noch einen Trick: In Sperrrichtung bilden die unterschiedlich dotierten Hälften des Kristalls in der Leuchtdiode eine kleine Kapazität, die aufgeladen werden kann, und die vom Photostrom entladen wird. Um nun mit einer Leuchtdiode an einem Mikrocontroller Helligkeiten messen zu können, müssen Anode und Kathode an digitale Pins angeschlossen werden, so dass die Leuchtdiode je nach Polung leuchtet oder in Sperrrichtung geladen wird. Wird nun mit geladener Sperrschichtkapazität einer der beiden Pins auf Eingang geschaltet, so wird nach einiger Zeit der eingelesene logische Zustand wechseln - genau dann, wenn die sinkende

Spannung an der Sperrschichtkapazität, die ja durch den Photostrom entladen wird, die Hystereseschwelle des Einganges erreicht. Je heller es ist, desto schneller findet die Entladung statt. Und so kann über die Zeit, die zum Erreichen der Hystereseschwelle benötigt wird, die Helligkeit bestimmt werden.

Die Photoströme sind winzig, die Sperrschichtkapazität auch, so dass hier auf kurze Anschlüsse (geringe zusätzliche Kapazität) und saubere Kontakte geachtet werden muss. Da die Eingänge intern mit Feldeffekttransistoren beschaltet sind, sind die Leckströme durch den MSP430 so klein, dass sie die Messung nicht stören, aber ein Finger oder eine Antistatikmatte können die Messung schon sehr beinflussen. Die Entladezeiten sind natürlich von Leuchtdiode zu Leuchtdiode unterschiedlich, aber liegen größenordnungsmäßig zwischen 50 μs im Sonnenschein und mehreren Millisekunden bei Dämmerung.

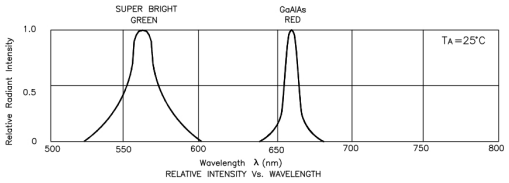


Abbildung 2: Das Spektrum der superhellen roten LED

Der Algorithmus und wie er funktioniert

Ledcomm wird vom Timer in immer gleichen Zeitabständen aufgerufen - der Basiszeit. In so einer Basiszeit kann die Leuchtdiode entweder leuchten, oder aber zu Beginn kurz umgekehrt geladen werden und während der Basiszeit ihre Sperrschichtkapazität durch den Photostrom entladen lassen, um hell oder dunkel unterscheiden zu können. Aus diesen beiden Elementen - *Strahlen* und *Lauschen* - ist alles weitere zusammengesetzt.

Zu Beginn, ohne bestehende Verbindung, folgen auf 8 Basiszeiten *Strahlen* 32 Basiszeiten *Lauschen*. Da keine Pausen dazwischen sind, ist von außen ein 8 Basiszeiten langer Lichtpuls zu sehen. Für jede Basiszeit *Lauschen* ergibt sich ein Bit - hell oder dunkel - das in einen Lichtmusterregister hingeschoben wird. Da die einzelne Leuchtdiode sich selbst nicht sehen kann, zeigen alle Bits Dunkelheit, zumindest solange die Sonne nicht direkt auf die Schaltung scheint.

8x *Strahlen*, 32x *Lauschen*, 8x *Strahlen*, 32x *Lauschen*, ... So geht es immer weiter.

Während das *Lauschen* läuft, sucht der Algorithmus nach eintreffenden Lichtpulsen von der Gegenstelle. Dazu wird nach jedem *Lauschen* das Muster Hell-Hell-Hell-Dunkel-Dunkel gesucht, was der Flanke eines eintreffenden Pulses entspricht. Wird diese erkannt, wird gezählt, wie viele Basiszeiten lang Helligkeit zu beobachten war - die Pulslänge kodiert die übertragenen Daten - und es wird sofort ein Puls zurückgeschickt, ohne auf den Ablauf der 32 Basiszeiten zu warten.

Wenn die beiden Leuchtdioden sich also sehen, wird das Muster schneller, und ein Beobachter von außen würde sehen:

8 Basiszeiten lang Strahlen der 1. LED,
2x Dunkel,
8x Strahlen der 2. LED,
2x Dunkel,
8x Strahlen der 1. LED...

Damit sind die beiden Kommunikationspartner synchronisiert und könnten beginnen, Daten zu übertragen. Da die beiden Stellen jedoch in Bewegung sein könnten oder der Schatten eines Fingers eine Hell-Dunkel-Flanke erzeugen könnte, wird in dem Algorithmus gewartet, bis 18 aufeinanderfolgende einkommende Pulse korrekt detektiert worden sind, bevor der Verbindungsanfang sicher erkannt ist und mit dem Senden von Daten begonnen wird. Wenn an einer Stelle jedoch nach dem 32. *Lauschen* immer noch keine Flanke erkannt worden ist, dann gilt die Verbindung als abgerissen.

Dem aufmerksamen Leser ist an dieser Stelle bestimmt aufgefallen, dass es für den Verbindungsaufbau auch genügen würde, wenn eine der Stellen bis zum ersten eintreffenden Puls immerfort lauschend dunkel bleibt und erst dann *strahlend reagiert*.

Jetzt sind alle Grundlagen gelegt...

...und es können Daten über die Pulslängen übertragen werden.

Eins aber noch: Würden wir an dieser Stelle die Länge der eingetroffenen, beobachteten Pulse protokollieren, fiele etwas auf: Ein Puls, der mit acht Basiszeiten Dauer gesendet worden ist, wird oft mit 8 Lauschbits Helligkeit erkannt, aber manchmal auch mit 9. Dies kommt daher, dass die beiden Taktquellen nicht perfekt synchron laufen - es kann also passieren, dass eine Basiszeit Strahlen auf zwei Basiszeiten Lauschen verteilt wird und bei genügender Helligkeit in beiden Lauschtakten erkannt wird. Falls der eine Takt etwas schneller oder langsamer als der andere läuft, könnten auch mal 7 oder 10 detektiert werden.

Die Datenübertragung an sich ist nun keine Schwierigkeit mehr:

- 3 bis 6 Basiszeiten lang hell werden als logische *Eins* erkannt, die als 4x Strahlen gesendet wird
- 7 bis 10 Basiszeiten lang hell werden als logische *Null* erkannt, die als 8x Strahlen und in Ruhe gesendet wird
- mehr als 11 Basiszeiten lang hell werden als *Übertrag* erkannt, welcher als 12x Strahlen gesendet wird.

(Natürlich wird nicht ein beliebig langer Puls als Übertrag erkannt - wird an passender zeitlicher Stelle geblendet, so würde nach Ablauf der hellen 32 Basiszeiten *Lauschen* ohne die Erkennung von Hell-Hell-Hell-Dunkel-Dunkel ein Verbindungsabriss erkannt werden.)

```
8 (9) 8 (9)   8 (8) 8 (9) 8 (8) 8 (8) 8 (7) 8 (9) 8 (8) 8 (8)
<- Sync ->    <-- Beide Datenempfangsregister mit je 16 logischen Nullen

8 (8) 8 (9) 8 (8) 8 (8) 8 (10) 8 (9) 8 (8) 8 (8)
leeren und auf stabile Verbindung warten     --> (Verbindungsanfang)

8 (9) 8 (8) 4 (8) 8 (9) 8 (4) 4 (5) 12 (9) 8 (13) 8 (Timeout beim Lauschen)
                          <-Ankommend %110=6->
Ruhe-0...   <--Ausgehend %1001=9  -->              (Verbindungsende)
```

Abbildung 3: Beispiel einer kleinen, aber kompletten Ledcomm-Verbindung

In Ruhe und bei der Synchronisation werden Nullen übertragen, die in ein Datenempfangsregister hineingeschoben werden. Da für die Synchronisation 18 Pulse abgewartet werden, ist das Datenempfangsregister zu Beginn stets null. Solange keine Daten zu senden sind, werden im Ruhezustand Null-Pulse abgestrahlt. Kommen Daten, werden sie mit der führenden Eins zuerst übertragen und mit einem Übertrags-Puls abgeschlossen. Beim Eintreffen eines Übertragspulses wird der aktuelle Inhalt des Datenempfangsregisters gelesen und anschließend gelöscht. Auf diese Weise benötigen kleine Datenwerte weniger Zeit zur Übertragung, als wenn stets alle führenden Nullen mit übertragen werden müssten.

`42 = %101010`

wird als (4) (8) (4) (8) (4) (8) (12) übertragen.

`34 = %100010`

führt zu (4) (8) (8) (8) (4) (8) (12),

`06  = %110`

führt zu (4) (4) (8) (12). Und eine `0` wird als (12) übertragen - schließlich ist ja zu Beginn stets eine Null im Datenempfangsregister.

Um all dies einmal zusammen zu zeigen, zeigt **Abb. 2** ein Beispiel einer kleinen, aber kompletten Ledcomm-Verbindung, wobei die Zahlen in Klammern die beobachteten Pulslängen, die Zahlen ohne Klammern die gesendeten Pulslängen von einer Seite darstellen.

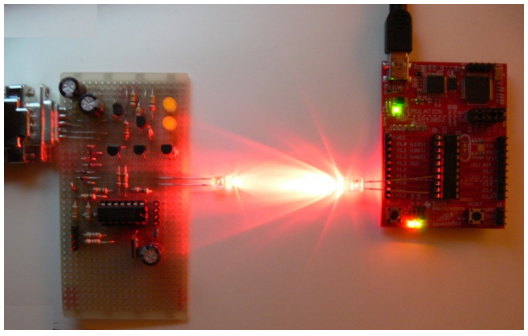


Abbildung 4: LEDCOMM bei der Arbeit. Rechts LED am LaunchPad, links gleiche LED an Platine des Autors.

Experimente und Erklärungen für einige davon

Die ersten Versuche lassen sich am besten in einem dämmrigen Raum durchführen. Das Bestimmen der Reichweite ist bestimmt schon geschehen, eine Sammellinse kann die Reichweite noch einmal beträchtlich erhöhen. Natürlich funktioniert es auch mit einem Spiegel - interessant ist aber auch die Reflektion an weißem Papier, weil so die Kommunikation leicht mit einem an ein Oszilloskop angeschlossenen Fototransistor (mit Spannungsquelle und Widerstand) beobachtet werden kann. Auch Lichtleiter können verwendet werden - Messingröhrchen, um einen Kern gewickelte Alufolie, Plexiglasplatten, Glasfasern... Anschließend kann versucht werden, Ledcomm zu blenden - ob es gelingt, hängt von der Farbe des Lichtes (die Photonenenergie muss die Bandlücke des LED-Kristalls überwinden) und der Helligkeit ab, auch in hellen Räumen funktioniert Ledcomm überraschend gut, direktes Sonnenlicht allerdings schickt die Verbindung in den Garten.

Die Basiszeit und damit die Geschwindigkeit der Übertragung wird an dieser Stelle festgelegt:
`1953 $0172 !`
Ein kleinerer Wert macht die Kommunikation schneller, aber auch kurzreichweitiger und manchmal weniger stabil, wogegen ein größerer Wert alles verlangsamt, größere Reichweiten zulässt, aber dafür auch empfindlicher für störendes Umgebungslicht macht. 1935/8MHz = 8/32768Hz = 244 μs als Basiszeit haben sich als guter Kompromiss bewährt. Natürlich muss der Wert auf beiden Seiten verändert werden.

Nach den ersten Erfolgen lohnt es sich, mit unterschiedlichen Leuchtdiodenpärchen zu experimentieren, je nach Kristallmaterial und Bauform sind durchaus Überraschungen möglich. Werden die Leuchtdioden verkehrt herum angeschlossen, glimmen sie nur schwach - zu sehen ist die umgekehrte Spannung, die kurz zum Laden der Sperrschichtkapazizät angelegt wird.

Interessant ist, dass bei geschicktem Blenden die Verbindungsanzeige auf einer Seite alleine ausgehen kann. Der Algorithmus sucht nach Hell-Dunkel-Flanken, die in einem bestimmten zeitlichen Rhythmus eintreffen. Wenn nun eine Seite geschickt geblendet wird, wird auf der Seite nichts mehr empfangen, aber es werden weiterhin Pulse zur Synchronisierungssuche ausgesandt. Die kommen an, und gaukeln auf der nicht geblendeten Seite, die sich damit synchronisiert, eine laufende Verbindung vor.

In sich selbst zurückspiegeln führt übrigens zu keiner Reaktion - die Leuchtdiode kann nur abwechselnd leuchten und beobachten, so kann sie sich selbst nicht sehen. Gleiches Problem ergäbe sich, wenn beide exakt synchron laufen würden. Da aber die Taktfrequenz aus dem internen Oszillator nicht perfekt ist, laufen beide stets nach einer kleinen Weile ein wenig zeitlich auseinander und können sich dann sehen.

Um einmal eine Helligkeitsmessung im Detail mitzuverfolgen, kann ein Oszilloskop direkt an eine der Leuchtdioden angeschlossen werden, aber dann wird die Verbindung abreißen, da die Kapazität normaler Tastköpfe im Vergleich zur Sperrschichtkapazizät von einigen Picofarad groß ist und die Helligkeitsmessung aus der Sicht des Mikrocontrollers durcheinanderbringt.

Links

[1] `http://mecrisp.sourceforge.net/`
[2] `http://www.forth-ev.de/filemgmt/singlefile.php?lid=490` enthält das komplette Paket um mit zwei LaunchPads das *LEDCOMM* aufzubauen. Umgebung: Linux. Im Paket enthalten sind:

README - enthält das Mecrisp-Handbuch und das Glossar für den Kern.

forth-mecrisp-2553.hex - Die Grundvariante für den G2553.

basisdefinitions-launchpad.txt - Die zusätzlichen Definitionen für das Launchpad.

README-Launchpad - Die Beschreibung dazu.

disassembler.txt - Für die ganz Neugierigen, die einmal tiefer hineinschauen möchten.

[3] `http://www.forth-ev.de/filemgmt/singlefile.php?lid=502` enthält ebenfalls das komplettePaket. Umgebung: Windows XP.

Literatur

MITSUBISHI ELECTRIC RESEARCH LABORATORIES
`http://www.merl.com`
Very Low-Cost Sensing and Communication Using Bidirectional LEDs
Paul Dietz, William Yerazunis, Darren Leigh
TR2003-35 July 2003
Hinterlegt bei
`http://www.forth-ev.de/filemgmt/singlefile.php?lid=492`

Quelltext

```
1   compiletoflash
2
3   : Sendedatenholen ( -- Daten true | false ) ?key dup if key tuck emit then ;
4   : Datenempfangen  ( Daten -- ) emit ;
5   : Verbindungsanfang ( -- ) ." (Up) "      1 p1out cbis! ; \ Verbindungsanzeige
6   : Verbindungsende   ( -- ) ." (Down) " cr 1 p1out cbic! ; \ an roter SMD-LED
7
8   1 constant Anode
9   2 constant Kathode
10
11  : Strahle ( -- )
12    Anode   P2OUT cbis!
13    Kathode P2OUT cbic!
14    Anode Kathode or P2DIR cbis!
15  ;
16
17  : Lauschen-Vorbereitung ( -- )
18    Anode   P2OUT cbic!            \ Sperrschichtkapazität
19    Kathode P2OUT cbis!            \ durch Verpolung laden
20    Anode Kathode or P2DIR cbis!
21    begin Kathode P2IN cbit@ until \ Warten, bis die Kathode geladen ist
22    Kathode P2DIR cbic!
23  ;
24
25  : Lauschen-Nachbearbeitung ( -- Flag )
26    Kathode P2IN cbit@ not         \ Ist die Kathode entladen, ist es hell
27  ;
28
29  18 constant Synchrondauer
30
31  8 variable Strahlzaehler
32  0 variable Verbindungsdauer
33  0 variable Lauschzaehler
34  0 variable Lichtmuster
35  0 variable Sendedaten
36  0 variable Datenregister
37
38  : msb? ( x -- x Flag ) dup $8000 and 0<> ;
39
```

```
40  : HolePuls ( -- )
41    8 Strahlzaehler ! \ Null-Puls, wird auch im Ruhezustand gesendet.
42
43    \ Verbindungsdauer prüfen, beginne erst zu Senden,
44    \ wenn die Verbindung auch sicher besteht. Belasse es im Falle einer
45    \ gerade laufenden Synchronisation dabei, Ruhezustandspulse abzugeben.
46
47    Verbindungsdauer @ Synchrondauer =
48    if
49
50    Sendedaten @ ?dup if  \ An bestehender Übertragung weiterarbeiten
51                          msb? if 4 Strahlzaehler ! then \ Eins-Puls erforderlich ?
52                          shl \ Wurde gerade die Merk-Eins herausrotiert ?
53                          dup 0= if 12 Strahlzaehler ! then  \ Übertragungspuls !
54                          Sendedaten !
55
56                        else  \ Neue Daten holen und vorbereiten
57                          Sendedatenholen
58                          if
59                            \ Bei neuen Daten <> 0 wird die führende Eins gesendet
60                            ?dup if  ( Daten-zum-Senden )
61                                   4 Strahlzaehler !
62
63                                   \ Füge im LSB eine Merk-Eins an und rotiere
64                                   \ die führende Daten-Eins durch das MSB hinaus
65                                   msb? if \ Ist das MSB schon die führende Eins ?
66                                          shl 1 or \ Nur Merk-Eins reinrotieren
67                                        else       \ die nicht übertragen wird
68                                          shl 1 or \ Eine Merk-Eins reinrotieren
69                                          begin    \ die nicht übertragen wird
70                                            msb?   \ Schieben, bis die führende
71                                            swap shl swap \ Daten-1 herauspurzelt
72                                          until
73                                        then
74                                                 \ Fertig geschobene Datenbits
75                                   Sendedaten !  \ zum weiteren Senden bereitlegen
76
77                                 \ Für eine Daten-Null genügt ein Übertragungspuls
78                                 else 12 Strahlzaehler !
79                                 then
80                          then
81                        then
82    then ;
83
84  : Bitmustererkennung ( -- )
85    Verbindungsdauer @ Synchrondauer <>
86    if \ Verbindung besteht erst nach 18 mal Zublinkern stabil.
87      1 Verbindungsdauer +!
88      Verbindungsdauer @ Synchrondauer = if Verbindungsanfang then
89    then
90
91    Lichtmuster @
92        %1111111111100 and  \ Übertragungspuls gültig mit 11 bis 14 Basiszeiten
93    dup %1111111111100 = if drop Datenregister @ Datenempfangen
94                                 0 Datenregister ! \ Datenregister löschen, da die
95                           else    \ Daten mit variabler Länge übertragen werden
96
97            %111111100 and    \ Null-Puls wird mit 7-10 Basiszeiten erkannt
98            %111111100 = if   Datenregister @ shl      Datenregister !
99                              else Datenregister @ shl 1 or Datenregister !
```

```
100                       then \ Eins-Puls wird mit 3-6 Basiszeiten erkannt.
101
102                       then
103   ;
104
105   : Taktlauscher-init ( -- )
106     0 Sendedaten !     \ Keine Daten zum Herausrotieren und Abstrahlen !
107     Verbindungsdauer @ Synchrondauer = if Verbindungsende then
108     0 Verbindungsdauer !
109
110     \ Für einen hellen Taktlauscher
111     HolePuls Strahle
112
113     \ Für einen dunkelen Taktlauscher
114     \ 0 Strahlzaehler !  1 Lauschzaehler !  Lauschen-Vorbereitung
115   ;
116
117   : Taktlauscher ( -- )
118     Strahlzaehler @ ?dup
119     if \ Strahlzähler ist nicht Null
120       -1 Strahlzaehler +!
121       1- if
122            Strahle
123          else \ Gerade Null geworden
124            32 Lauschzaehler !
125            0 Lichtmuster !
126            Lauschen-Vorbereitung
127          then
128     else
129       Lauschen-Nachbearbeitung
130       1 and Lichtmuster @ shl or
131       dup Lichtmuster ! ( Neues-Lichtmuster )
132       %11111 and   \ Suche nach der Hell-Dunkel-Flanke
133       %11100 = if  \ am Ende eines Pulses
134                  Bitmustererkennung
135                  HolePuls
136                  Strahle
137                else \ Noch ist keine Flanke eingetroffen
138                  -1 Lauschzaehler +!
139                  Lauschzaehler @ if Lauschen-Vorbereitung
140                                  else Taktlauscher-init then
141                then
142     then ;
143
144   : ledcomm ( -- )
145     255 p2dir c! \ Ausgänge für die Ledcomm-LED und die Verbindungsanzeige
146     65  p1dir c!
147     64  p1out c!
148
149     Taktlauscher-init
150     ['] Taktlauscher irq-timera0 !
151
152     $10  $0162 ! \ Capture/compare Interrupt Enable
153     1953 $0172 ! \ Periodendauer einstellen, 1953 / 8 MHz = 244 us
154     $210 $0160 ! \ SMCLK als Taktquelle, aufwärts zählen
155
156     eint begin again \ Interrupts aktivieren, in Endlosschleife warten ;
157
158   compiletoram
```

Wave Engine (6)

Hannes Teich

In der letzten Folge wurde mit einer neuen (dritten) Version der Wave Engine begonnen. Diesmal wird eine erste kleine Melodie erzeugt und in eine abspielbare Wave-Datei geschrieben. Dabei dürften die Defizite deutlich werden, die es künftig zu beseitigen gilt. Wie versprochen, ist alles herunterladbar und unter Gforth lauffähig.

Das Anlegen der Frequenztabellen wurde schon besprochen. Heute kommen der Sinus-Generator und der Wave-Generator dazu. Während Letzterer so ziemlich seine endgültige Form aufweist - er wurde von Version 2 übernommen - ist der Sinus-Generator lächerlich übersichtlich, gemessen daran, was er künftig noch alles zu bewältigen haben wird: Klangfarben durch Obertöne, Ab- und Nachklingen, weicher Toneinsatz und einiges mehr.

Die Partitur - also die Notentabelle - die sinnvollerweise in Version 2 forthfrei war, ist für diesmal der Einfachheit halber forthig gestaltet, mit `CREATE` und Kommas. Die Melodie ist einstimmig und besteht aus lauter gleich langen Tönen und Pausen. Eine Schleife arbeitet die Melodie ab, eine weitere ist für den einzelnen Sinus-Ton zuständig, der mit 44100 Stützpunkten pro Sekunde berechnet wird. Wer bei reinen Sinustönen den Eindruck hat, das Instrument sei leicht verstimmt, der möge von Sinus auf Rechteck umschalten. Rechtecktöne weisen die volle Palette an ungeradzahligen Harmonischen auf und lassen die Tonhöhe besser erkennen. Bei Beginn und Abbruch der Töne ist jeweils ein Knack zu hören, selbst beim nahtlosen Übergang von einem Ton zum nächsten. Der Grund ist in **Abb.1** zu sehen. Das zu umgehen wird die Aufgabe der schon früher beschriebenen Staccato-Funktion sein.

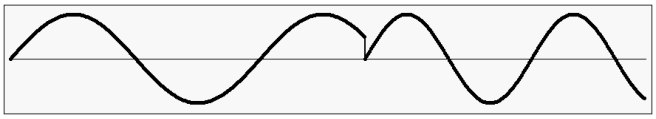


Abbildung 1: Harter Tonübergang mit hörbarem Knack

Anmerkungen zu den Listings

Die vorliegende Version 3-01 ist in fünf Dateien untergebracht. In die Haupt-Routine (*WAVER3-01.fs*) werden eingegliedert: die Frequenztabelle (*freqtab-1.fs*), der Wave-Generator (*wavgen.fs*), die Partitur (*partit.fs*) und der Sinus-Generator (*singen.fs*), die sich alle beim Laden am Bildschirm melden. Fehlt eine, beschwert sich Gforth entsprechend. Der Name der Ausgabe-Datei im Wave-Format ist mit *WE-001.wav* voreingestellt. Die hier nur teilweise abgedruckten Listings sind ziemlich selbsterklärend. Hinweis: *freqtab-1.fs* unterscheidet sich von *freqtab.fs* aus der letzten Folge nur kosmetisch; und vom Wave-Generator *wavgen.fs* wird hier nur ein Ausschnitt abgedruckt.

Die **Abb.2** erklärt exemplarisch den Aufbau einer WAV-Datei und ist in die kommentierten Listings eingebettet, weil dort ja auch ersichtlich ist, wie es gemacht wird. Und am Ende des Listings zeigt **Abb.3** wie das Programm aufgerufen wird.

Alle beteiligten Dateien stecken im Ordner *Wave-Engine-Ver3-01*, siehe unten in *WEpage*. Damit bleibt mir bis zum nächsten Mal nur noch „Fröhliches Exprimentieren!“ zu wünschen.

Referenzen

In diesen VD-Heften wurde bisher über die Wave Engine berichtet: 2/2011 *Top-One-Partitur*, 3/2011, 1/2012, 2/2012, 4/2012, 1/2013.
Beachten Sie bitte den Dateibereich der Website der Forth-Gesellschaft unter
`http://www.forthev.de/filemgmt/viewcat.php?cid=54`
oder alternativ
`http://tinyurl.com/WEpage`
sowie die Website des Autors unter
`http://www.stocket.de/WE`

Listing: waver3-01.fs

```
1   \ ====== 1 ======= 2 ======= 3 ======= 4 ======= 5 ======= 6 ======= 7 ==|
2   \ WAVER3-01.fs - last edit: 03-jun-2013 12:00 -jgt
3   \               ---------------------------------------
4   \              |                                       |
5   \              |       W a v e   E n g i n e           |   ============
6   \              |                                       |   Gforth 0.7.0
7   \              |            Version 3-01               |   ============
8   \              |_______________________________________|
9   \
10  \ Präambel:  Nachdem die komplexe, aber recht ordenlich laufende zweite
11  \ Version des Synthesizers "Wave Engine" durch ein Linux-Update instabil
```

```
12  \ geworden war, reifte der Entschluss, eine dritte Version (3-xx) neu zu
13  \ beginnen und dabei allen Code zugänglich zu machen.
14  \
15  \ Die hier vorligende Version 3-01 kann noch nicht viel, aber sie ist
16  \ imstande eine Partitur (Zahlentabelle) in eine abspielbare Wave-Datei
17  \ zu wandeln. Die Melodie ist einstimmig und verwendet nur gleich lange
18  \ reine Sinustöne (alternativ Rechtecktöne) und Pausen.
19  \
20  \ Den Sinustönen fehlen jegliche Obertöne. Die Rechtecktöne beinhalten
21  \ alle ungeradzahligen Harmonischen, nach oben hin abfallend. Künftig
22  \ sollen Klänge aus bis zu 16 Harmonischen zusammengesetzt werden.
23  \
24  \ Frei gewählt werden kann: ein-/zweikanalig, Stereoklang ein/aus,
25  \ Sinus/Rechteck, Lautstärke, Geschwindigkeit und Tonart-Verschiebung.
26  \ Kommentartext kann vor oder nach den Daten der Wave-Datei stehen.
27
28        true  constant  mono          \ false: 2 Kanäle (mit/ohne Stereo)
29       false  constant  stereo        \ true: Raumklang (mono = false)
30       false  constant  square        \ true: Rechteck statt Sinus
31          70  constant  volume        \ Amplitude (70 okay, max 128)
32        6000  constant  durat         \ Tonlänge (6000 okay)
33        true  constant  prelist       \ true: Text vor (statt nach) Daten
34
35     s" WAVER3-01"      2constant  version   \ Versionsbezeichnung
36     s" WE-001.wav"     2constant  wavefile  \ Name der Ausgabedatei
37     s" freqtab-1.fs"   included             \ Frequenztabelle
38     s" wavgen.fs"      included             \ Wave-Generator
39     s" partit.fs"      included             \ Partitur
40     s" singen.fs"      included  cr cr      \ Sinus-Generator
41  \ ----------------------------------------------------------------------
42
43  \ Wave-Daten an dataChunk anhängen
44  : append-data  ( --)
45         30000 nulls                         \ Pause vorab
46               bytecnt @
47            16 align-nulls                   \ Pausenende justieren
48               waves                        \ Wave-Daten schreiben
49               bytecnt @ 4 -                \ 4 Pos. nach rechts
50            16 align-nulls                  \ 0...15 Null-Bytes
51               bytecnt @ dataend ! ;        \ Chunk-Ende merken
52
53  \ Text an listChunk anhängen
54  : append-text  ( --)
55               s"   [ "              $>buf
56               s" Wave Engine 3.01"  $>buf
57               s" edit 130602-1420"  $>buf   \ editiert: ...
58               s" comp "             $>buf   \ compiliert: ...
59               timestamp                     \ aktuelles Datum
60               s"  ]"                $>buf   \ Text-Ende
61                                             \ Leerzeichen anhängen
62               bytecnt @  4 -                \ 4 Pos. nach rechts
63            16 align-spaces                  \ 0...15 Leerzeichen
64               bytecnt @ listend ! ;         \ Chunk-Ende merken
65
66  : init  ( --) ;
67
68  \ ----------------------------------------------------------------------
69
70  \                    H A U P T - R O U T I N E
71
```

```
72  : build-wavefile  ( --)
73                      init
74                      create-outfile          \ Wave-Datei anlegen
75                      build-waveChunk         \ erste 12 Bytes
76                      build-formatChunk       \ weitere 24 Bytes
77        prelist IF
78                      build-listChunk         \ listChunk ohne Text
79                        append-text           \ Text anhängen
80                      build-dataChunk         \ dataChunk ohne Daten
81                        append-data           \ Wave-Daten anhängen
82                ELSE
83                      build-dataChunk         \ dataChunk ohne Daten
84                        append-data           \ Wave-Daten anhängen
85                      build-listChunk         \ listChunk ohne Text
86                        append-text           \ Text anhängen
87                ENDIF
88                      patch-waveChunkSize     \ Länge über alles
89                      patch-dataChunkSize     \ Länge dataChunk
90                      patch-listChunkSize     \ Länge listChunk
91                      fillup                  \ Füllbytes
92                      close-outfile ;         \ Wave-Datei schließen
93
94  \                             S T A R T
95  : go  build-wavefile
96              ."  fertig! " .s f.s  cr ;      \ "<0> <0>" = Stacks okay
97
98      version  type  ."   >>> Mit »go« starten! <<< " cr cr
99
100 \ =======[ Ende von WAVER3-01.fs ]=======================================
```

Listing: wavgen.fs

```
1   \ ====== 1 ======= 2 ======= 3 ======= 4 ======= 5 ======= 6 ======= 7 ==|
2   \ wavgen.fs - last edit: 03-jun-2013 12:00 -jgt
3     cr ." included: wavgen.fs
4
5   \                       W A V E - G E N E R A T O R
6
7               variable   tmp         \ n>buf, w>buf, c>buf temporär
8               variable   bufcnt      \ Pufferzähler
9               variable   bytecnt     \ Bytezähler über alles
10              variable   xcount      \ Zähler für Punktausgabe
11              variable   databeg     \ dataChunk Beginn
12              variable   dataend     \ dataChunk Ende
13              variable   listbeg     \ listChunk Beginn
14              variable   listend     \ listChunk Ende
15            0 value      wavefileID  \ Identifikationsnr der Wave-Datei
16        22050 constant   bufsize     \ Größe des Datenpuffers
17              create     wbuffer bufsize allot
18
19  \ Wave-Datei anlegen.
20  : create-outfile  ( --)
21              wavefile  R/W  BIN               ( c-addr 2 1)
22              CREATE-FILE ( fileid ior) drop    ( ior=0 for success)
23              to wavefileID ;
24
25  \ Daten in Wave-Datei schreiben.
26  : >file  ( addr len)  wavefileID  WRITE-FILE ( ior) drop ;
27
28  \ Schreibpuffer leeren und in Datei schreiben.
```

```
29  \ FLUSH-FILE leert auch den unsichtbaren Zwischenspeicher.
30  : bufflush  ( --)
31              wbuffer bufcnt @ >file
32              0 bufcnt !
33              wavefileID FLUSH-FILE drop ;
34
35  \ Schreibpuffer ("wbuffer") laden und ggf. in die Wave-Datei übertragen.
36  \ So lange die Daten nicht in den Puffer passen, wird der Puffer randvoll
37  \ gefüllt und sodann in die Wave-Datei entleert.
38  \ Dann wird der Datenrest in den Puffer geladen.
39  \ (Am Programmende wird der Puffer mit bufflush geleert.)
40  : $>buf  ( addr len --)
41          BEGIN   bufsize bufcnt @ -  ( free) >r
42                  dup r@ >=
43          WHILE   over wbuffer bufcnt @ + r@ move \ transfer
44                  swap r@ + swap  r> -
45                  wbuffer bufsize >file
46                  0 bufcnt !
47          REPEAT  r> drop  dup
48          IF      >r wbuffer bufcnt @ + r@ move   \ transfer
49                  bufcnt @ r@ + bufcnt !
50                  bytecnt @ r> + bytecnt !
51          ELSE    2drop
52          THEN ;
53
54  \ 4-byte-Wert in den Schreibpuffer schreiben.
55  : n>buf  ( u --)  tmp !                         \ bytes counted in bufcnt
56              tmp 4 ( addr len) $>buf ;           \ schreiben
57
58  \ Wie n>buf, aber ohne bytecnt zu verändern (für Patches).
59  : n>buf|  ( u --)                               \ bytecnt erhalten
60              bytecnt @ >r n>buf r> bytecnt ! ;
61
62  \ 2-byte-Wert in den Schreibpuffer schreiben.
63  : w>buf  ( u --)  tmp !                         \ byte counted in bufcnt
64              tmp 2 ( addr len) $>buf             \  schreiben
65  \ Fortschritt anzeigen:
66              1 xcount +!
67              xcount @ 100000 >= IF               \ alle 10000 Frames
68              0 xcount ! ." ."  THEN ;            \  einen Punkt ausgeben
69
70  \ 1-byte-Wert in den Schreibpuffer schreiben.
71  : c>buf  ( u --)  tmp !                         \ byte counted in bufcnt
72              tmp 1 ( addr len) $>buf ;           \  schreiben
73
74  \ Einen Wert 0...99 als zwei ASCII-Dezimalziffern schreiben.
75  \ Beispiel: 73 --> ($20 + 3) * $100 + ($20 + 7) = $2300 + $27 = $2327
76  \ Wegen "little endian" wird das Ergebnis $2327 als $27 $23 abgelegt.
77  : >ascii  ( u -- uw)
78                  100 mod  dup 10 mod  '0' + 256 *
79                          swap 10 /  '0' + +  w>buf ;
80
81  \ Aktuelles Datum und Uhrzeit schreiben (Format: yymmdd-hhmmss).
82  : timestamp  ( --)
83          TIME&DATE  ( sec min hour day month year)  100 mod
84          >ascii ( yy)   >ascii ( mm)   >ascii ( dd)
85          [char] -  c>buf
86          >ascii ( hh)   >ascii ( mm)   drop ( ss) ;
87
88  \ Gegebene Anzahl oder ASCII-Nullen oder Leerzeichen schreiben.
```

```
89  : nulls   ( n --)  0 ?DO    0 c>buf  LOOP ;
90  : blanks  ( n --)  0 ?DO  $20 c>buf  LOOP ;
91
92  \ Pufferinhalt durch Nullen bzw. Leerzeichen so verlängern, dass der
93  \ Pufferzähler (cnt) modulo n = 0 ergibt.
94  : align-nulls   ( cnt n --)  dup >r  mod r@ swap - r> mod nulls ;
95  : align-spaces  ( cnt n --)  dup >r  mod r@ swap - r> mod blanks ;
96
97  \ Das "Resource Interchange File Format" ("RIFF") besteht aus mehreren
98  \ Abschnitten ("Chunks"), von denen hier nur vier Verwendung finden:
99  \ Wave, Format, Data und List.
100 \ Die ersten 44 Bytes sind vom Aufbau her festgelegt, nur die mit (*)
101 \ gekennzeichneten Werte sind veränderlich. waveChunkSize ist die Länge
102 \ über alles, aber ohne die ersten 8 Bytes (hier: 52-8=44). Auch die
103 \ Werte für Mono/Stereo sowie für Samples und Bytes pro Sekunde sind
104 \ veränderlich. Werden Daten (je 2 byte bei Mono, 4 byte bei Stereo)
105 \ eingefügt, so ändern sich die Längenangaben entsprechend. Gleiches
106 \ gilt für Kommentartext. waveChunk und formatChunk stehen immer am
107 \ Anfang, dataChunk und listChunk können ggf. vertauscht werden.
108 \ ----------------------------[ Chunk 1 ]----------------------------------
109 \   0    00    52494646    'RIFF'                        groupID
110 \   4    04    54000000    44     (*)                    waveChunkSize
111 \   8    08    57415645    "WAVE"                        riffType
112 \ ----------------------------[ Chunk 2 ]----------------------------------
113 \  12    0C    666D7420    'fmt '                        formatChunkID
114 \  16    10    10000000    16                            formatChunkSize
115 \  20    14    0100        1          no compression     wFormatTag
116 \  22    16        0200    2      (*) mono=1, stereo=2   wChannels
117 \  24    18    44AC0000    44100  (*) draft: 11250       dwSamplesPerSec
118 \  28    1C    10B10200    176400 (*) draft: 44100       dwAvgBytesPerSec
119 \  32    20    0400        4                             wBlockAlign
120 \  34    22        1000    16                            wBitsPerSample
121 \ ----------------------------[ Chunk 3 ]----------------------------------
122 \  36    24    64617461    'data'                        dataChunkID
123 \  40    28    00000000    0      (*)                    dataChunkSize
124 \ (44)  (3C)                (noch keine Daten)            (2 byte)
125 \ ----------------------------[ Chunk 4 ]----------------------------------
126 \  44    3C    4C495354    'LIST'                        listChunkID
127 \  48    30    28000000    0      (*)                    listChunkSize
128 \ (52)  (34)                (noch kein Text)
129 \ ------------------------------------------------------------------------
130
131 \ Hier werden die Chunks aufgebaut. Die Daten im dataChunk werden durch
132 \ den Sinus-Generator per w>buf geschrieben, wodurch sich der Bytezähler
133 \ bufcnt entsprechend erhöht und listChunk vor sich herschiebt.
134 \ ----------------------------[ Chunk 1 ]----------------------------------
135 : build-waveChunk  ( --)
136                       s" RIFF"  $>buf      \ groupID
137                              0  n>buf      \ waveChunkSize (patch)
138                       s" WAVE"  $>buf ;    \ riffType
139
140 \ ----------------------------[ Chunk 2 ]----------------------------------
141 : build-formatChunk  ( --)
142                       s" fmt "  $>buf      \ formatChunkID
143                             16  n>buf      \ formatChunkSize
144                              1  w>buf      \ wFormatTag (no compr)
145
146           mono  IF           1  w>buf      \ wChannels (mono)
147         ( draft) 0   IF  11025  n>buf      \ dwSamplesPerSec
148                          22050  n>buf      \ dwAvgBytesPerSec
```

```
149                     ELSE   44100  n>buf      \ dwSamplesPerSec
150                            88200  n>buf      \ dwAvgBytesPerSec
151                     ENDIF
152                 ELSE           2  w>buf      \ wChannels (stereo)
153         ( draft) 0  IF     11025  n>buf      \ dwSamplesPerSec
154                            44100  n>buf      \ dwAvgBytesPerSec
155                     ELSE   44100  n>buf      \ dwSamplesPerSec
156                           176400  n>buf      \ dwAvgBytesPerSec
157                     ENDIF
158                 ENDIF          4  w>buf      \ wBlockAlign
159                               16  w>buf ;    \ wBitsPerSample
160
161   \ ----------------------------[ Chunk 3 ]---------------------------------
162   : build-dataChunk  ( --)
163                         s" data"  $>buf      \ dataChunkID
164                                 0  n>buf      \ dataChunkSize (patch)
165                       bytecnt @  databeg !  \ Position merken
166                                  ( ... ) ;  \ hier folgen Wave-Daten
167
168   \ ----------------------------[ Chunk 4 ]---------------------------------
169   : build-listChunk  ( --)
170                         s" LIST"  $>buf      \ listChunkID
171                                 0  n>buf      \ listChunkSize (patch)
172                       bytecnt @  listbeg !  \ Position merken
173                                  ( ... ) ;  \ hier folgt Text
174
175   \ ------------------------------------------------------------------------
176
177   \ Hier werden die Chunk-Längen überschrieben (gepatcht).
178   \ Auf Pos. 4 steht die Länge "über alles" (nach den ersten 8 byte).
179   \ Die Länge des formatChunk ist unveränderlich = 16.
180   \ Die Reihenfolge von dataChunk und listChunk ist wählbar.
181   \ Auf Pos. 40 steht die Länge der Wave-Daten bzw. des Textes.
182   \ Die Position des letzen Chunk (Text bzw. Wave-Daten) ist veränderlich.
183
184   \ Länge des waveChunk (über alles) setzen
185   : patch-waveChunkSize  ( --)
186               bufflush
187               4. waveFileID REPOSITION-FILE ( ior) drop
188               bytecnt @ 8 - n>buf| ;
189
190   \ Länge des dataChunk (Wave) setzen
191   : patch-dataChunkSize  ( --)
192               bufflush
193               databeg @ 4 - s>d waveFileID REPOSITION-FILE ( ior) drop
194               dataend @ databeg @ - n>buf| ;
195
196   \ Länge des listChunk (Text) setzen
197   : patch-listChunkSize  ( --)
198               bufflush
199               listbeg @ 4 - s>d waveFileID REPOSITION-FILE ( ior) drop
200               listend @ listbeg @ - n>buf| ;
201
202
203   \ Datei mit Nullen beenden
204   : fillup  ( --)
205               bufflush
206               bytecnt @ s>d waveFileID REPOSITION-FILE ( ior) drop
207               bytecnt @  16 align-nulls
208               bufflush ;
```

```
209
210  \ Wave-Datei schließen
211  : close-outfile  ( --)
212              wavefileID  CLOSE-FILE ( ior) drop ;
213
214  \ =======[ Ende von wavgen.fs ]=========================================
```

```
00000000  52 49 46 46 7C 00 00 00 57 41 56 45 66 6D 74 20  RIFF....WAVEfmt
00000010  10 00 00 00 01 00 02 00 44 AC 00 00 10 B1 02 00  ................
00000020  04 00 10 00 64 61 74 61 18 00 00 00 00 00 00 00  ....data........
00000030  31 32 33 34 35 36 37 38 00 00 00 00 00 00 00 00  12345678
00000040  00 00 00 00 4C 49 53 54 38 00 00 00 20 20 5B 20      LIST....  [
00000050  57 61 76 65 20 45 6E 67 69 6E 65 20 33 2E 30 31  Wave Engine 3.01
00000060  65 64 69 74 20 31 33 30 35 33 30 2D 32 31 31 30  edit 130530-2110
00000070  63 6F 6D 70 20 31 33 30 35 33 31 2D 31 32 35 37  comp 130531-1257
00000080  20 5D 20 20 00 00 00 00 00 00 00 00 00 00 00 00   ]  ............
00000090
```

Diese Wave-Datei hat eine Länge von $84 = 132 byte. (Die letzten 12 Null-Bytes gehören nicht dazu.) Die Längenangaben der Chunks sind immer um 8 byte kürzer als der Chunk selbst. Auf Position $4 steht die Gesamtlänge der Wave-Datei ($7C), auf Position $10 die Länge des Format-Chunk ($10), auf Position $28 die Länge des Data-Chunk ($18) und auf Position $48 die Länge des List-Chunk. Der Übersichtlichkeit halber gibt es hier mit "12345678" nur 8 Pseudo-Wavedaten-Bytes. Diesen voran gehen 4 Null-Bytes, stellvertretend für etwa 10000 Null-Bytes, die für eine Pause am Anfang eines Musikstücks sorgen. Am Ende der Daten wird mit Leerzeichen aufgefüllt, um die Position des nachfolgenden List-Chunk festzulegen, aus Gründen guter Lesbarkeit im Hex-Format. Da die Reihenfolge von Data- und List-Chunk frei gewählt werden kann, wird im List-Chunk ebenso verfahren, aber mit nachfolgenden Leerzeichen, die dem verfassten Text automatisch zugesetzt werden (hier zweimal $20). Am Ende der Wave-Datei werden (aus kosmetischen Gründen) Null-Bytes angehängt, die nicht mitgezählt werden.

Abbildung 2: Eine Demo-Wave-Datei

Listing: singen.fs

```
1   \ ====== 1 ======= 2 ======= 3 ======= 4 ======= 5 ======= 6 ======= 7 ==|
2   \ singen.fs - last edit: 03-jun-2013 12:00 -jgt
3     cr ." included: singen.fs"
4
5   \                      S I N U S - G E N E R A T O R
6
7           variable   timer                    \ Tonlängenzähler
8           fvariable  phase                    \ Sinusphase
9
10  \ Neuen (einzelnen) Ton abarbeiten
11  : sound  ( r-step --)
12          durat  timer !                      \ Tonlänge laden
13          0e  phase f!                        \ Phase ab Null
14          BEGIN
15             phase f@ fsin                    \ ### Sinus ###
16    square   IF f0>=
17                IF .6e ELSE -.6e ENDIF        \ Rechtecksignal (square = 1)
18             ENDIF
19             volume s>f f*                    \ Amplitude erhöhen
20             f>s dup w>buf                    \ in Wave-Datei schreiben
21    mono     0= IF
22    stereo         IF negate ENDIF            \ Raumklang
23                   w>buf                      \ in 2. Kanal schreiben
24                ELSE drop ENDIF
25             fdup phase f@ f+  phase f!       \ Phase um step erhöhen
```

```
26              timer @  1-  dup  timer !  0=     \ Tonlängenzähler 1-
27           UNTIL                                \ Tonende?
28           fdrop ;                              \ Phasenschritt löschen
29
30    \ Musikstück abarbeiten
31    : waves  ( --)
32           tunelen 0                            \ Länge der Musik
33           DO
34              tune i cells + @                  \ Ton#
35              dup    0 >                        \ Ton oder Pause?
36              IF     1-                         \ Ton# --> Tabellenoffset
37                     transpo +                  \ Tonhöhe-Verschiebung
38                     fadr12 f@                  \ Phasenschritt (Tonhöhe)
39              ELSE   drop 0e                    \ Pause, kein Schritt
40              ENDIF  sound                      \ Ton erzeugen
41           LOOP ;
42
43    \ =======[ Ende von singen.fs ]=============================================
```

Listing: partit.fs

```
1     \ ====== 1 ======= 2 ======= 3 ======= 4 ======= 5 ======= 6 ======= 7 ==|
2     \ partit.fs - last edit: 03-jun-2013 12:00 -jgt
3       cr ." included: partit.fs"
4     \                            P A R T I T U R
5
6     \ Hier werden die Daten für den Sinus-Generator bereitgestellt.
7     \ Die Melodie ist einstimmig, die Tondauer einheitlich.
8     \ Der Tonumfang: knapp 3 Oktaven (G bis e'') plus 1 Oktavverschiebung.
9     \ Der Tonumfang wird durch transpo = 12 um 1 Oktave nach oben verschoben.
10
11          12  value  transpo        \ 12 = eine Oktave höher (C-Dur)
12
13    create tune  here    ( "Quinto" )
14          37 ,  0 ,   44 ,  0 ,   37 ,  0 ,   44 ,  0 ,
15          37 ,  0 ,   44 ,  0 ,   37 ,  0 ,   44 ,  0 ,
16          37 , 61 ,   44 , 63 ,   37 , 65 ,   44 , 61 ,
17          37 , 56 ,   44 , 58 ,   37 , 60 ,   44 , 61 ,
18          37 ,  0 ,   44 ,  0 ,   37 ,  0 ,   44 ,  0 ,
19          37 ,  0 ,   44 ,  0 ,   37 ,  0 ,   44 ,  0 ,
20          37 , 61 ,   44 , 63 ,   37 , 65 ,   44 , 61 ,
21          37 , 56 ,   44 , 58 ,   37 , 60 ,   44 , 61 ,
22          37 ,  0 ,   44 ,  0 ,   37 ,  0 ,   44 ,  0 ,
23          37 ,  0 ,   44 ,  0 ,   37 ,  0 ,   44 ,  0 ,
24          32 , 60 ,   44 , 60 ,   32 , 61 ,   44 , 63 ,
25          32 , 56 ,   44 , 58 ,   32 , 60 ,   44 , 61 ,
26          37 , 63 ,   44 , 65 ,   37 , 56 ,   44 , 65 ,
27          37 , 56 ,   44 ,  0 ,   37 ,  0 ,   44 ,  0 ,
28          32 , 60 ,   44 , 60 ,   32 , 61 ,   44 , 63 ,
29          32 , 65 ,   44 , 63 ,   32 , 60 ,   44 , 61 ,
30          37 , 63 ,   44 , 65 ,   37 , 56 ,   44 , 65 ,
31          37 , 56 ,   44 ,  0 ,   37 ,  0 ,   44 ,  0 ,
32          37 , 61 ,   44 , 63 ,   37 , 65 ,   44 , 61 ,
33          37 , 56 ,   44 , 58 ,   37 , 60 ,   44 , 61 ,
34          37 ,  0 ,   44 ,  0 ,   37 ,  0 ,   44 ,  0 ,
35          37 ,  0 ,   44 ,  0 ,   37 ,  0 ,   44 ,  0 ,
36           0 ,  0 ,   49 ,  0 ,   44 , 44 ,   45 ,  0 ,
37          44 ,  0 ,    0 ,  0 ,   48 ,  0 ,   49 ,  0 ,
38                here swap - 4 / value tunelen
39
```

```
40  \ ------------------------------------------------------------------------
41  \                  F R E Q U E N Z - Ü B E R S I C H T
42  \
43  \  Name  Ton#  Frequenz  Winkelschritt
44  \ ----------------------------------
45  \  G      32   97.99886   0.01396247
46  \  Gis    33   103.8262   0.01479272
47  \  A      34   110.0000   0.01567234
48  \  Ais    35   116.5409   0.01660427
49  \  H      36   123.4708   0.01759161
50  \ ----------------------------------
51  \  c      37   130.8128   0.01863766
52  \  cis    38   138.5913   0.01974592
53  \  d      39   146.8324   0.02092007
54  \  dis    40   155.5635   0.02216404
55  \  e      41   164.8138   0.02348198
56  \  f      42   174.6141   0.02487830
57  \  fis    43   184.9972   0.02635764
58  \  g      44   195.9977   0.02792494
59  \  gis    45   207.6523   0.02958545
60  \  a      46   220.0000   0.03134469
61  \  ais    47   233.0819   0.03320854
62  \  h      48   246.9417   0.03518322
63  \ ----------------------------------
64  \  c'     49   261.6256   0.03727533
65  \  cis'   50   277.1826   0.03949183
66  \  d'     51   293.6648   0.04184014
67  \  dis'   52   311.1270   0.04432808
68  \  e'     53   329.6276   0.04696397
69  \  f'     54   349.2282   0.04975659
70  \  fis'   55   369.9944   0.05271527
71  \  g'     56   391.9954   0.05584989
72  \  gis'   57   415.3047   0.05917089
73  \  a'     58   440.0000   0.06268938  <-- Kammerton a' = 440 Hz
74  \  ais'   59   466.1638   0.06641708
75  \  h'     60   493.8833   0.07036645
76  \ ----------------------------------
77  \  c''    61   523.2511   0.07455065
78  \  cis''  62   554.3653   0.07898367
79  \  d''    63   587.3295   0.08368028
80  \  dis''  64   622.2540   0.08865617
81  \  e''    65   659.2551   0.09392794
82  \  f''    66   698.4565   0.09951318
83  \  fis''  67   739.9888   0.1054305
84  \  g''    68   783.9909   0.1116998
85  \  gis''  69   830.6094   0.1183418
86  \  a''    70   880.0000   0.1253788
87  \  ais''  71   932.3275   0.1328342
88  \  h''    72   987.7666   0.1407329
89  \ ----------------------------------
90  \  c'''   73   1046.502   0.1491013
91  \  cis''' 74   1108.731   0.1579673
92  \  d'''   75   1174.659   0.1673606
93  \  dis''' 76   1244.508   0.1773123
94  \  e'''   77   1318.510   0.1878559
95  \  Pause   0
96
97  : test  ( Ton# --)  dup 0=
98              IF      ." Pause"
99              ELSE    1- 440e freq12  fdup f. 2pi f*  44100e f/  f.
```

```
100                ENDIF ;
101
102    \ Die Tonnummer wird hier um 1 zum Tabellenoffset verringert.
103    \ Nummern von 1 aufwärts, denn Null ist für die Pause reserviert.
104
105    \ =======[ Ende von partit.fs ]=========================================
```

```
jgt@jgt-M2NS-NVM:~/Forth/Wave Engine$ gforth WAVER3-01.fs

included: freqtab-1.fs
included: wavgen.fs
included: partit.fs
included: singen.fs

>>> Mit »go« starten! <<<

Gforth 0.7.0, Copyright (C) 1995-2008 Free Software Foundation, Inc.
Gforth comes with ABSOLUTELY NO WARRANTY; for details type `license'
Type `bye' to exit
go ..... fertig! <0> <0>  ok
```

Abbildung 3: Aufruf-Protokoll

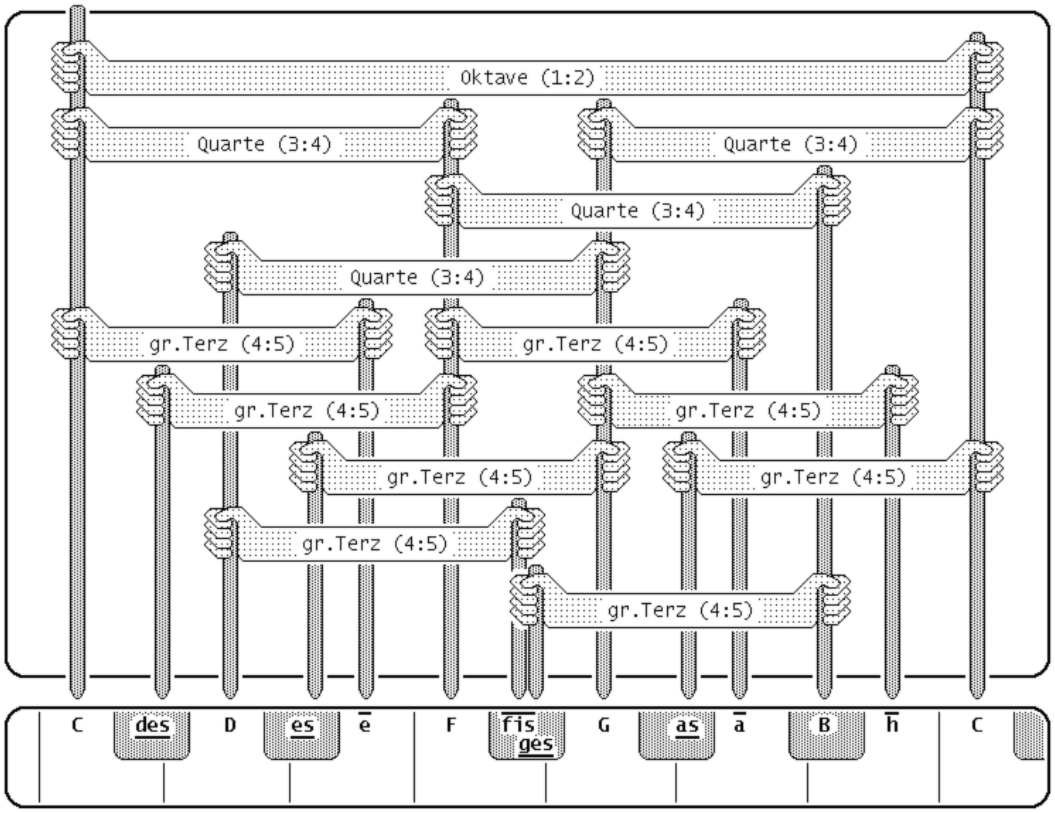


Abbildung 4: Rück- und Vorgriff auf die vom Autor propagierte reine Stimmung, die sich auf Schwingungsverhältnisse stützt statt auf lineare Aufteilung der Oktave.

Der STRIP32–Forth–Prozessor

Willi Stricker

Auf den 16–Bit STRIP folgt logischerweise eine 32–Bit–Version *STRIP32*. Dessen Konzept ist nahezu identisch mit dem der 16–Bit–Version *STRIP16*. Im Wesentlichen besteht der Unterschied darin, dass hierbei alle Datenpfade eine Breite von 32 Bit haben. Das gilt insbesondere für alle Stack–Elemente von Parameter- und Returnstack, aber auch für den Memory–Datenbus, an den alle Speicherbausteine und Input/Output–Elemente, jeweils intern oder extern, angeschlossen sind.

Auch der Memory–Adressbus bekommt, wie bei den meisten 32–Bit–Prozessoren, die gleiche Bitzahl wie der Datenbus. Damit ergibt sich ein adressierbarer Speicherbereich von 2^{32}, also etwa 4 Giga–Byte. Die Bitzahl der Stack–Pointer ist, wie bei der 16–Bit–Version, jeweils abhängig von der Anzahl der bereitgestellten Stackelemente, theoretisch maximal auch 32 Bit. Bei deren Bearbeitung werden beim Schreiben überzählige Bits ignoriert, beim Lesen durch Nullen ersetzt.

Der STRIP32 erhält den gleichen Basis–Befehlssatz wie der STRIP16 (Minimal–Befehlssatz). Aufgrund der Erweiterung auf 32 Bit ergeben sich jedoch zwei Besonderheiten.

- Die Sprungbefehle (Branches) werden hardwareseitig verändert.
- Die Memory–Schnittstelle wird erweitert (zusätzliche Befehle).

Die STRIP32–Sprung–Befehle `BRANCH` und `?BRANCH`

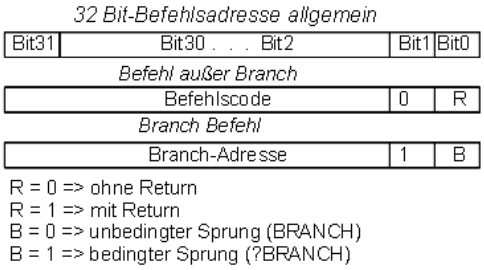


Abbildung 1: Schema der Instruktion

Beim 16–Bit–System wurde bereits das bei 16–Bit–Code–Adressen unbenutzte Bit0 (LSB) als *Return–Bit* benutzt. Im 32–Bit–System ist bei allen Befehls–Adressen zusätzlich das Bit1 immer null und damit unbenutzt, es kann deswegen ebenfalls zusätzliche Informationen enthalten, und wird nun als *Branch–Bit* benutzt. Im Fall, dass Bit1 eine 1 enthält, werden die oberen 30 Bits als Sprung–Adresse (bit0=bit1=0) eingesetzt. Damit benötigt der Sprung nur eine Zelle Speicherplatz, einen Speicherzugriff und nur einen Takt–Zyklus. Es wird also sowohl Speicherplatz als auch Ablaufzeit eingespart!

Während das Bit0 wie beim 16–Bit–System als *Return–Bit* dient, wird es beim Sprung zum *Bedingungs–Bit*. Auf die Möglichkeit *Sprung mit Return* wird hierbei verzichtet, da er nur sehr selten benutzt wird.

32–Bit–Memory–Schnittstelle

Der STRIP32–Kernel bearbeitet ausschließlich 32–Bit–Daten! Der externe Adressraum ist aber wie bei fast allen Prozessoren in Bytes organisiert, so dass die Notwendigkeit besteht, auch auf 8–Bit- oder 16–Bit–Daten (Byte bzw. Word) zugreifen zu können. Das gilt insbesondere für Input/Output–Elemente. Dazu wird hardwareseitig zwischen Kernel und externer Peripherie ein *Memory–Interface* vorgesehen, das vom Kernel zwei zusätzliche Steuerleitungen bekommt, die Word–Line *W* und Byte–Line *B*.

```
 8-Bit Byte Zugriff: B = 1, W = 0
16-Bit Word Zugriff: B = 0, W = 1
32-Bit Cell Zugriff: B = 0, W = 0
```

Für den Adress–Zugriff gilt, dass der 8–Bit–Operand auf alle Adressen zugreifen kann, der 16–Bit–Operand nur auf gerade Adressen (Bit0 = 0), und der 32 Bit–Operand nur auf durch 4 teilbare Adressen (Bit0 und Bit1 = 0). Für die Zugriffe *fetch* und *store* auf Bytes und Words sind dann spezielle Befehle nötig.

```
C@    ( adr -- 0.0.0.b0 )
C!    ( bx.bx.bx.b0 adr -- )
W@    ( adr -- 0.0.b1.b0 )
W!    ( bx.bx.b1.b0 adr -- )
  b3.b2.b1.b0 = 32Bit Wert, 4 Bytes
  0 = Byte ist gleich null
  bx = beliebiges Byte (unwirksam)
  adr = 32-Bit-Adresse
```

Zusätzlich wurden zwei Sonderbefehle für die Byte- und Word–Manipulation vorgesehen.

```
CSWAP ( b3.b2.b1.b0 -- b3.b2.b0.b1 )
WSWAP ( b3.b2.b1.b0 -- b1.b0.b3.b2 )
```

Hinweis: Die Befehle `W@`, `W!`, `CSWAP` und `WSWAP` sind keine Standard–Forth–Befehle!

Die Anwendung der speziellen Swap–Befehle soll an einem Beispiel gezeigt werden. Über ein Byte–Port, also eine Serielle Schnittstelle, mit der Adresse `BYPO`, werden nacheinander 4 Bytes erwartet, die zu einem 32–Bit–Operanden zusammengefasst werden sollen. Sie treffen in der Reihenfolge Byte0, Byte1, Byte2, Byte3 ein.

```
BYPO C@  \ Byte0 einlesen an Stelle 0
BYPO C@  \ Byte1 einlesen an Stelle 0
CSWAP    \ Byte1 an Selle 1 "swappen"
OR       \ Byte0 und 1 zusammenfassen an 0 und 1
BYPO C@  \ Byte2 einlesen an Stelle 0
```

```
BYPO C@  \ Byte3 einlesen an Stelle 0
CSWAP    \ Byte3 an Stelle 1 "swappen"
OR       \ Byte2 und 3 zusammenfassen an 0 und 1
WSWAP    \ Bytes 0 und 1 nach 2 und 3 "swappen"
OR       \ alle zusammenfassen
```

Link

Der STRIP–Forth–Prozessor (16-Bit) wurde ursprünglich in einem FPGA der Firma Actel auf einem Eval-Kit programmiert (Typ APA 075), und ist beschrieben worden im Beitrag: Willi Stricker, „Von der virtuellen Forth–Maschine zum realen Forth–Prozessor“, Vierte Dimension 2010, Heft Nr.4, S.19-22. `http://www.forth-ev.de/filemgmt_data/files/4d2010-04.pdf`

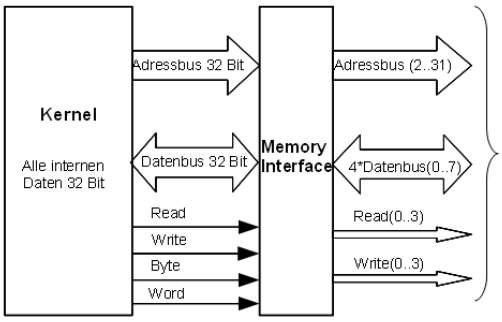


Abbildung 2: Blockschaltbild

Nachfolgend sei auch die Diskussion wiedergegeben, die sich rund um die Frage nach dem minimalen Befehlssatz für eine Forth–Maschine ergeben hatte. Fred hat im Folgenden das Pro und Kontra kurz zusammen gefasset.

Einsparung von `(lit)` als Primitive durch Fassung als High-Level-Wort

Fred Behringer

In [1] habe ich u.a. behauptet, dass man aus Willi Strickers Liste von Primitives [2] das Wort `(lit)` (aus Turbo-Forth (TF)) auch als High-Level-Wort fassen kann. Mein Vorschlag enthielt den Festwert 2 . Ich wollte als Werkzeug nur den üblichen Colon-Compiler zulassen, so wie in TF enthalten, und im Übrigen kein Forth-Wort, das nicht über den Colon-Compiler erzeugt wurde. Willi sagte in [3] dazu: „Die Definition `: LIT R@ @ R> 2 + >R ;` (aus [1]) funktioniert zwar im Prinzip, benutzt jedoch für die Literale 2 sich selbst. Das geht also nicht!“

(... was ganz [1] im Schlepp hat.) Warum denn nicht? Den expliziten Wert 2 umgehe ich durch:

```
: (zwei) r@ ;
: zwei ( -- 2 ) (zwei) (zwei) swap - ;
```

und baue ihn in meinen urspünglichen Vorschlag ein zu:

```
: (lit) ( -- n )
   r@ @ r>  (zwei) (zwei) swap -   + >r ;
```

Eine Zirkeldefinition kann ich dabei nicht erkennen. Von Bernd Paysan erhielt ich nach dem Verfassen dieses Vorschlags die folgende sehr interessante E-Mail-Mitteilung:

> Es gibt noch eine kürzere Lösung, Konstanten einzubinden ohne ein richtiges (lit):
>
> ```
> : (const) ( -- n )  r> @ ;
> : cell ( -- 2 ) (const) [ 2 , ] ;
> ```
>
> Lösung: `(const)` beendet einfach das aufrufende Wort, und benötigt deshalb keine Arithmetik.
>
> Die Begründung, warum man `(lit)` im Befehlssatz eines Forth-Prozessors nicht weglassen darf, auch wenn es ohne geht, ist aber die: `(lit)` ist eines der häufigsten Wörter, es ist performancekritisch.

Bernds Mail hat mich sehr gefreut. Sie vervollständigt meine Sicht zum Thema „Möglichst wenige Primitives und sonst nur High-Level-Worte“. Der Zusatzaufwand interessiert mich dabei weniger.

[1] Behringer, Fred: Kontrollstrukturen als Colon-Definitionen und ohne die Immediate-Eigenschaft? Vierte Dimension 4/2011, S.35-40.

[2] Stricker, Willi: Minimaler Basis-Befehlssatz für ein Forth-System. Vierte Dimension 3/2009, S.15-17.

[3] Stricker, Willi: Kommentar zum Artikel „Kontrollstrukturen als Colon-Definitionen und ohne die Immediate-Eigenschaft“ in der VD 4/2011 von Fred Behringer. Vierte Dimension 1/2012, S.6.

Interaktives Programmieren in der 4E4TH–IDE

Dirk Brühl, Michael Kalus

Forth auf MCUs erlaubt einen komplett anderen Programmierstil als C oder Assembler. Beim interaktiven Programmieren (*iP*) sendet man relativ kurze Stücke Quellcode hinunter an die Ziel–MCU, die Forth auf der CPU automatisch compiliert, und diese kurzen Stücke können dann gleich ausgeführt werden. Damit kann man alles testen, die chip–internen Peripherals der MCU genauso wie alle außen zusätzlich angeschlossenen Komponenten. Das ist etwas, das kein Simulator so kann, weil die echten Eigenschaften des Einbaus dabei zum Tragen kommen. Solch ein Vorgehen ist unerlässlich, will man ein Produkt wirklich auf „Herz und Nieren" durchchecken, alle Hardwarekomponenten und Analoges darin eingeschlossen. Die Teste laufen über eine konventionelle serielle Schnittstelle (UART) und benötigen lediglich zwei Leitungen, RXD und TXD - von der Masse GND mal abgesehen.

Es wird also nicht, um das deutlich herauszustellen, Code „am Stück" auf einem PC erzeugt, dort auch im RAM compiliert, möglicherweise durch einen Simulator gegeben, und dann endlich als binary auf die Ziel–MCU gebracht (eng.: *flash target*), um dann zu debuggen, und diesen Zyklus erneut zu durchlaufen. Beim iP geht man da einen viel direkteren Weg, und baut den Code sukzessive um Tests herum auf. Erst ganz am Schluss, wenn das Programm steht, wird es am Stück erneut auf die MCU geladen. Also erst dann als fertiges Abbild des Programmspeichers (*image*).

Diese Arbeitsweise wird von keiner der gängigen (mir bekannten) IDEs unterstützt. Und die diversen Werkzeuge an Text–Editoren, Terminal–Emulatoren und flashing tools sind alle doch sehr umständlich zu bedienen und halten einen unnötig auf. Auch Dirk Brühl war es leid, damit hantieren zu müssen, und er hat sich in 2012 daran gemacht, eine IDE zu konzipieren, die das iP abbildet. Benutzt man diese 4e4th–IDE zum ersten Mal, und hat man noch die alten Konzepte der edit–compile–flash–Zyklen im Kopf, ist man erst einmal verdattert. Es dauert aber nicht lange, und dann begreift man, was da so anders läuft, und es wird klar, dass einem diese 4e4th-IDE sehr entgegenkommt, und man beginnt, flüssiger und flüssiger damit umzugehen.

Die Eigenschaften

Die 4e4th–IDE V1.0 integriert die in der folgenden Tabelle aufgelisteten Grundfunktionen des iP.

- Smart Terminal
- Einfacher iP–Session–Editor
- Target–Unterstützung
- Autosave von Quellcode

Diese vier sind in einem GUI zusammengefasst (**Abb. 1**). Zentral ist die Eingabezeile und das Multifunktionsfenster. Über sehr wenige Schaltflächen wird das Ganze bedient. Da gibt es den START/STOP–Knopf (*button*), um sich mit der Ziel-MCU (*target*) zu verbinden, oder um das Target abzumelden, bevor man es abstöpselt. Dann ist oben ein dominierender breiter Balken da, mit dem zwischen Terminal- und Edit–Mode hin- und hergeschaltet wird.[1] Und eine Zeile mit einigen wenigen Menüeinträgen, damit wird das Target unterstützt. Das ist es eigentlich auch schon. Das iP passiert also vom Konzept her auf der Eingabezeile und im Session–Fenster.

Die Eingabezeile

Da wir es ja mit einem sehr kleinen Forth auf einer sehr kleinen Maschine zu tun haben, wenn wir eine MCU iP'en, gibt es im Zielsystem klassischerweise auch nur den kleinen TIB (*terminal input buffer*) im RAM, also gerade mal eine Zeile von 80 Zeichen, mit sehr begrenzter Editierfunktion — Backspace. Das wird durch die 4e4th–IDE nachgebildet. Auch dort ist der TIB gegeben. Und von da aus wird Zeichen für Zeichen an die MCU gesendet, sobald man etwas eintippt. Die LEDs oben links im Fenster zeigen dies an. Da ist auch zu sehen, ob man verbunden ist, und ob das Target wieder empfangsbereit oder noch beschäftigt ist.

Das Terminal–Fenster

Der komplette Zeichenstrom zum Target hin und von diesem zurück (echo) wird im Terminal–Fenster mitgeschrieben, und automatisch im *TermLog* festgehalten (*key logger* und *emit logger*). Das Fenster gibt also Einblick in das, was sich gerade im Target tut. Die Übertragung geschieht tatsächlich zeichenweise, es wird wirklich jeder einzelne Tastendruck vom Target empfangen und durch ein Echo quittiert. So besteht die volle Kontrolle darüber, was das Target jeweils mit dem Zeichen macht.

Das iP–Session–Fenster

Besonders an der 4e4th–IDE ist nun, dass parallel zum *TermLog* ein weiteres Log gefahren wird. Darin wird allerdings nicht der komplette Zeichenstrom, der hin und her geht, geloggt, sondern nur der Anteil, der an die MCU gesendet worden ist. Das wurde englisch *iP session log* getauft, kurz *Session*. Durch Klick auf den Balken oben kann die Ansicht der Logs gewechselt werden. Und dann lässt sich das Session–Log auch gleich editieren. Und ab

[1] Das *Code Composer Studio v5* macht es übrigens ganz ähnlich. Dort wird auch zwischen dem Debuger und dem Editor, welche sich überdecken, hin- und hergeschaltet. Dort nennt man es „umschalten der Perspektive".

der Änderung kann sofort nochmal an das Target übertragen werden (*compile*) — zeitsparenderweise nur der korrigierte und der noch nicht compilierte Teil. Auf diese Weise wächst im Session–Log nach und nach das komplette ausgetestete fertige Programm heran.

Quellcode an das Forth im Target schicken

Dieser Vorgang wird für gewöhnlich *Download* genannt. Forth verarbeitet ja Ascii–Zeichenketten. Aus diesem Eingabestrom compiliert das Forth im Target selbst den ausführbaren Code. Das ist anders als in C. Natürlich kann es sein, dass in einer Datei der Quellcode Fehler enthält und Forth ein Wort nicht kennt. Gewöhnliche Terminal–Emulatoren reagieren nicht darauf, wenn Forth eine Fehlermeldung macht. Das smarte Terminal hingegen hält dann an. Und bietet gleich die Fehlerstelle aus dem Quellcode zur Korrektur an. Danach kann der Vorgang wiederholt werden. Sei es, dass weiter hinzucompiliert wird, oder dass man den bisherigen Inhalt aus dem Target–Flash löscht (`WIPE`), und den Download von Neuem beginnt . Und es gibt zwei alternative Wege, den Quellcode zum Target zu senden. Aus der Zwischenablage (*clipboard*) heraus, oder aus einer markierten Stelle einer Quelldatei.

Das Terminal ist insofern auf Forth gut abgestimmt, als es auch zeilenweise aus dem Clipboard senden kann, was beim Debuggen des Codes hilfreich sein kann. Es kann aus jeder Textdatei Quellcode entnommen werden, und über die Zwischenablage zum Download werden. Zum einen können auf diese Weise Quellcode–Bausteine aus anderen Dateien als die im Projekt schon integrierten sofort ausprobiert werden. Zum anderen wird damit ein alternativer Programmierstil unterstützt, bei dem in einem externen Editor der eigenen Wahl Forth-Module geschrieben werden, die dann in die Zwischenablage kopiert und ans Target gesendet werden.

Freundlicherweise sammelt das SmartTerminal ja auch diese zusammengetragenen Schnipsel alle im Session–Log, wodurch dann schon die Zusammenfassung entsteht, von all dem, was man gemacht hat. Bereinigt man das, was dort gesichert wurde, von den offensichtlichen Irrtümern, die man während des iP gemacht hat, ist damit dann der Forth-Quellcode der Applikation fertig.

An dieser Stelle sollte erwähnt werden, wie die 4e4th–IDE Projekte verwaltet. Alle Projekt–Dateien liegen im 4e4th–IDE–Verzeichnis. Nimmt man Forth–Quellcode aus externen Dateien dazu, werden diese zuerst in das Projektverzeichnis kopiert, damit alles beisammenbleibt was mit dem Projekt zu tun hat. Der Download erfolgt immer aus diesen Projekt–Dateien heraus. Werden Änderungen an solchen Dateien vorgenommen, landen diese Änderungen in der Projekt–Datei. Die ursprüngliche Quelle bleibt unverändert. Die 4e4th–IDE kann dort nicht von sich aus hinschreiben. Das ist ein beachtenswerter Unterschied zu allgemeinen Text–Editoren. Dadurch bleibt alles innerhalb des Projektes. Nur durch gezielten Eingriff des Benutzers kann auch eine Datei an anderer Stelle in Kopie hinterlegt werden. (*Project: Save as file …*)

Die Wortliste

Auf der 4e4th–IDE wird eine Liste aller Worte, die beim iP hinzugekommen sind, mitgeführt. Diese Liste ist als Stapel organisiert, das jüngste Wort liegt oben auf. Man kann Worte einzeln markieren, und dann mit einem Klick im Target ausführen lassen. Und den Quellcode der Definitionen nachsehen. Dieser wird in den zentralen Bildschirm geschrieben, kann dort auch editiert werden, und — markiert — erneut zum Target geschickt werden. Ist die Anwendung fertig, genügt ein Klick auf *App!*, um daraus im Target die Applikation fertigzustellen, die dann nach reset gebootet wird.

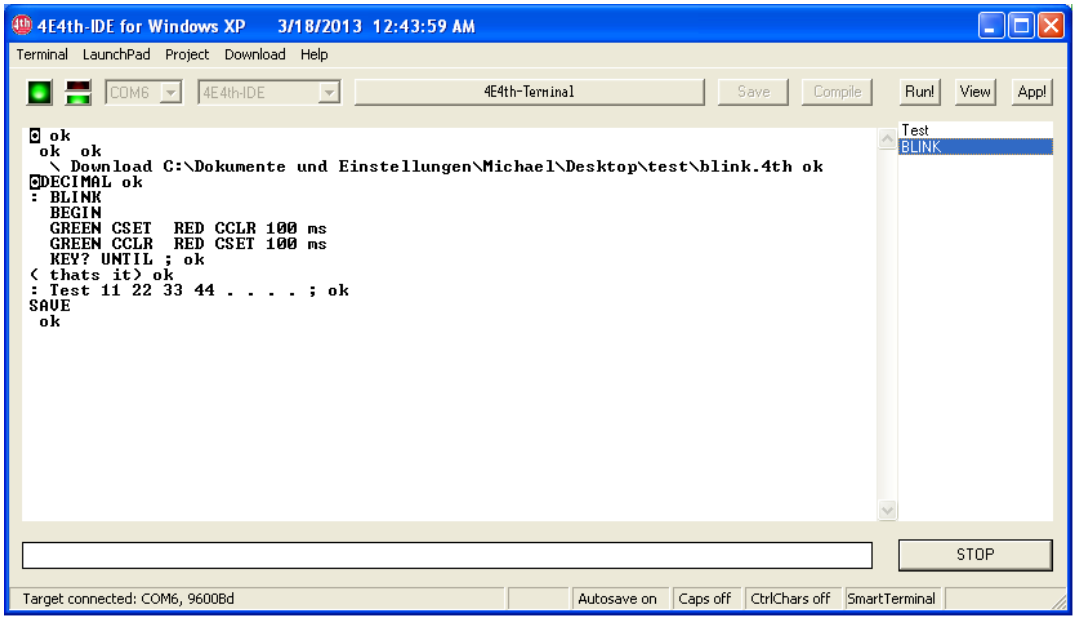


Abbildung 1: Screenshot der Forth–IDE (Windows XP)

Die 4e4th–IDE unterstützt damit den Wortschatz, den das 4e4th bietet. `WORDS` und `SAVE` und `' <word> APP !` sind dort ja schon vorhanden. Die IDE nutzt diese Eigenschaften des 4e4th aus. Obschon die IDE auch für andere Forths benutzt werden kann, sind solche ganz speziellen Eigenschaften natürlich genau auf das 4e4th zugeschnitten.

Die Unterstützung des Targets (Target support)

Zugeschnitten wurde die 4e4th–IDE erst einmal für die LaunchPads von Texas Instruments (TI), speziell für die MCU MSP430G2553. Das Flash dieser MCU kann, ohne fremde Werkzeuge benutzen zu müssen, komplett gelöscht und mit einem frischen Forthkern ausgestattet werden. Vier Forth–Varianten stehen derzeit zur Auswahl.

Die IDE installiert auch den Treiber für die LaunchPads, falls diese auf dem PC noch fehlen. Und erstellt eine Liste aller angeschlossenen LaunchPads — ja, es ist möglich, mit mehreren *gleichzeitig* zu arbeiten! Und sie kann die verschiedenen Speicherbereiche — RAM, Flash, Infoflash — aus dem Target auslesen und in eine Datei sichern (image).

Das *SmartTerminal* kennt die Namen der Peripherie–Register der MCU und ersetzt diese durch deren Adresse während des Downloads, sodass der knappe Platz in der MCU nicht von all diesen vielen Konstanten verbraucht wird, der Programmierer aber die Namen aus dem Manual der MCU verwenden kann.

Zum Debuggen kann die Anzeige aller Steuercodes aus dem Ascii–Zeichensatz hinzugeschaltet werden. Und es ist möglich, alle Eingaben in Großbuchstaben zu wandeln, bevor sie an das Target gesendet werden.

Sicherung der Arbeit

Damit auch wirklich nichts verloren geht von der wertvollen Arbeit, wird alles automatisch geloggt, der Programmierer muss sich darum nicht kümmern. Man kann sofort in der letzten Session weitermachen, ohne irgendetwas laden zu müssen. Das Terminal–Log zeichnet alles auf, das von Target kommt, also auch die Echos der Eingabe. Das Session–Log registriert alle Eingaben des Programmierers, von der Eingabezeile ebenso wie von den Downloads. Jeder Wechsel vom Terminal in den Editor, und zurück, hinterlegt eine Kopie des bisher Erreichten mit Zeitstempel im Backup-Verzeichnis. Zwischenstände kann man zudem mit einem einfachen Klick auf Save in eine zeitgestempelte Datei sichern. Zusätzlich zum Loggen wird automatisch nach jedem Download das *Autosave* ausgeführt. Im Backup–Ordner befinden sich daher nach und nach eine Menge Dateien mit Zeitstempel hinter dem Dateinamen. Wenn ein Projekt abgeschlossen ist, befindet sich der gesamte Quellcode ja im SavedFile. Sichere das an eine Stelle deiner Wahl. Danach kann der Backup–Ordner geleert werden, davon wird nun nichts mehr benötigt.

Hilfen

Eine Schritt–für–Schritt–Anleitung gibt es im Internet. Wer mit Forth noch nicht vertraut ist, arbeitet am besten Leo Brodies *Starting Forth* durch. Beim 4e4th wurde darauf geachtet, damit kompatibel zu sein. Das 4e4th und so auch die IDE dazu sind simpel gehalten und sollen vor allem den Einstieg in Forth auf MCUs ermöglichen. Daher ist alles so „gerade aus“ wie möglich gehalten. Im repos des 4e4th gibt es das Glossar zu den Forth-Worten. Sie verhalten sich weitgehend ANS–Forth–konform, also kann man auch im Standard nachlesen. Im Wiki der Forth-Gesellschaft gibt es inzwischen Beispielprogramme, und eine Seite, in der grundlegende Experimente mit der MCU aufgeführt sind. Und die Forth Communitiy in den Foren erwies sich bisher als sehr hilfsbereit - wofür ich mich an dieser Stelle auch einmal herzlich bedanken möchte.

Dank und Ausblick

Die 4e4th-IDE wuchs in 2012 soweit heran, dass damit in einen noch geschlossenen Beta–Test gegangen werden konnte. Ausgesuchte wohlwollende Forthler aus Deutschland, der Türkei, Australien und den USA halfen beim Debuggen und gaben Anregungen und auch Hilfestellung beim Umgang mit Windows. All diesen lieben Menschen sei herzlich gedankt. Die jüngste Version ist nun für den offenen Beta–Test freigegeben, und wir freuen uns auf bug reports und weitere konstruktive Hinweise.
dirk.bruehl@usa.net
mik.kalus@gmail.com

Links

Das 4e4th–Projekt:
`http://www.forth-ev.de/repos/4e4th/`
Wähle dort den release/4e4th-a43 aus.
Die 4e4th–IDE:
`www.4e4th.eu`
Sowie:
`http://www.ti.com/ww/en/launchpad/overview_head.html`
`http://www.forth-ev.de/wiki/doku.php/projects:4e4th:start`
`http://www.forth-ev.de/wiki/doku.php/projects:4e4th:4e4th:start:msp430g2553_experimente`

.S und ok — Eine Revision

Michael Kalus

Neulich tauchte die Frage auf, wie man das, was auf dem Datenstack liegt, „richtig“ wiedergeben kann. Natürlich darf der Inhalt des Stacks bei dieser Inspektion nicht verändert werden (non-destructive). Die Recherche ergab, dass .S (dot-s sprich „print stack“) zwar je nach Hersteller und Implementation etwas anders aussieht, aber die Unterschiede doch nicht so gewaltig sind.

dpans Forth .S

Im Standard heißt es dazu:

> 15.6.1.0220 .S
> dot-s TOOLS
>
> (--)
>
> Copy and display the values currently on the data stack. The format of the display is implementation-dependent.
>
> .S may be implemented using pictured numeric output words. Consequently, its use may corrupt the transient region identified by #>.

Da ist also weder das Aussehen noch eine Reihenfolge festgelegt. Also schauen wir mal, wie das in den gängigen Forths gelöst worden ist.

Beispiele

gforth (Mac OSX)

```
11 -22 33 .s <3> 11 -22 33 ok
```

gforth stellt die Werte in der Reihenfolge dar, in der sie eingegeben worden sind, von links nach rechts, der oberste Wert also ganz rechts, vorzeichenrichtig in der jeweils gewählten Zahlenbasis. Zusätzlich wird in spitzen Klammern angegeben, wie viele Werte auf dem Stack sind. Es werden nur die obersten 9 Werte wiedergegeben, aber in den Klammern steht immer die tatsächliche Anzahl. Das ok ist schlicht gehalten.

win32forth (Windows)

```
11 -22 33 .s [3] 11 -22 33 ok...
```

win32forth macht es auch so. Hier steht die Anzahl in eckigen Klammern. Und es gibt nach jedem ok noch einen Hinweis darauf, was auf dem Stack verblieben ist. Das ok wird von sovielen Punkten gefolgt, wie Werte auf dem Stack liegen.

VFX Forth (Windows IA32)

```
11 -22 33 .s
DATA STACK
     top
              33 0000:0021
                -22 FFFF:FFEA
                 11 0000:000B
ok-3
```

Im VFX wird üppiger informiert. Da wird der Stapel auch als Stapel angezeigt. Und es wird darauf hingewiesen, dass es sich um die Wiedergabe des Datenstacks handelt, und wo „oben“() ist. Das ist für Dokumente, die auch andere als der Programmierer selbst später noch lesen und verstehen sollen, hilfreich. Die Werte werden vorzeichenrichtig in Dezimal und daneben noch mal absolut Hexadezimal dargestellt, in der Wortbreite der verwendeten Maschine. So ist der Streit, ob man grad mehr die Adressen oder den Zahlenwert haben will, im VFX so gelöst, dass man immer beides hat. Dem ok folgt auch hier die Anzahl der Stackelemente, als Decimalzahl.

SwiftX MSP430 EVALUATION (Windows)

```
11 -22 33 .s
11 -22 33 <-Top  ok
```

Im SwiftX ist man wieder sparsamer. Die Werte erscheinen in einer neuen Zeile, wieder in der Eingaberichtung. Für alle Fälle zeigt ein Pfeil und das Wort Top auf das obere Ende des Stapels. Das ok ist auch hier schlicht.

4e4th (MSP430)

```
11 -22 33 .s <3> 11 -22 33 $0Aok
```

Im 4e4th sieht es so aus wie im gforth. Aber dem ok wird immer die Zahlenbasis vorangestellt, in der sich das System gerade befindet; die Darstellung dieser Zahlenbasis erfolgt immer hexadezimal, symbolisiert durch das $-Präfix.

Mecrisp (MSP430)

```
11 -22 33 .s Stack: [3 ] 11 , 65514 , 33 *>
ok.
```

Es sei gleich verraten, was das *> ist: „...Du weißt doch, ein Stapel im Windstoß wird durch die Gegend gepustet, da fliegen die Zettel und Elemente herum. Das *> ist mein Briefbeschwerer.“ Dort, wo der auftaucht, ist das „Oben“ des Stapels. Der Zeilenumbruch vor dem ok dient dazu, im Falle vieler .S zur Fehlersuche ein bisschen Übersichtlichkeit zu bewahren.

Außerdem fand der Erfinder es in den Anfangstagen von Mecrisp nützlich zu sehen, wo genau die Stackausgabe

zu Ende ist, da dies manchmal die letzte Ausgabe vor einem chaotischen Absturz gewesen ist. Jetzt ist Mecrisp für den MSP430 stabil, doch der Briefbeschwerer hat sich gehalten. Bei .S im Mecrisp-MSP430 wird übrigens die Zahl der Elemente immer dezimal angegeben, während die Stackelemente in der aktuellen Basis gezeigt werden, *unsigned*, wie man im Beispiel sieht.

AmForth

```
11 -22 33 .s <3> 33 -22 11 ok
```

Nur im amforth ist die Reihenfolge anders herum. Da wird die Politik „was ich zuerst lese, ist oben auf dem Stack“ verfolgt. Das ist also so, wie es bei wiederholtem DOT auch wäre. Das ok bleibt auch hier schlicht. Eine ältere Version des .S gab den Stapel auch als zeilenweise Liste aus, top zuerst, bottom zuletzt, und zu jedem Element auch dessen Adresse im Stapel.

Starting Forth

Es gibt also unterschiedliche Meinungen dazu, und für verschiedene Zwecke die eine oder andere Abwandlung. Das Schlusswort hat Meister Leo Brodie. In seinem *Starting Forth*, Kapitel 2, schrieb er den inzwischen klassischen „praktischen Hinweis“ zur stackerhaltenden Stackwiedergabe.

> Anfänger, die gerade lernen, wie man mit Zahlen auf dem Stack nützlicherweise hantiert, machen oft einige DOTs, um zu sehen, was auf dem Stack war. Das Problem mit den DOTs ist aber, dass diese die Zahlen nicht auf dem Stack belassen, um damit weiterzumachen.
>
> Das Forth-Wort .S druckt alle Werte, die gerade auf dem Stack sind, *nicht-destruktiv*, also ohne sie zu entfernen. Lass uns das testen, zuerst ohne was auf dem Stack.
>
> ```
> .S <0>  ok
> ```
>
> Wie du siehst, zeigt diese Version des .S mindestens eine Zahl. Das ist die Anzahl der Elemente auf dem Stack.
>
> Nun versuchen wir es mit Zahlen auf dem Stack:
>
> ```
> 1 2 3 .S <3> 1 2 3  ok
>
> ROT .S <3> 2 3 1  ok
> ```

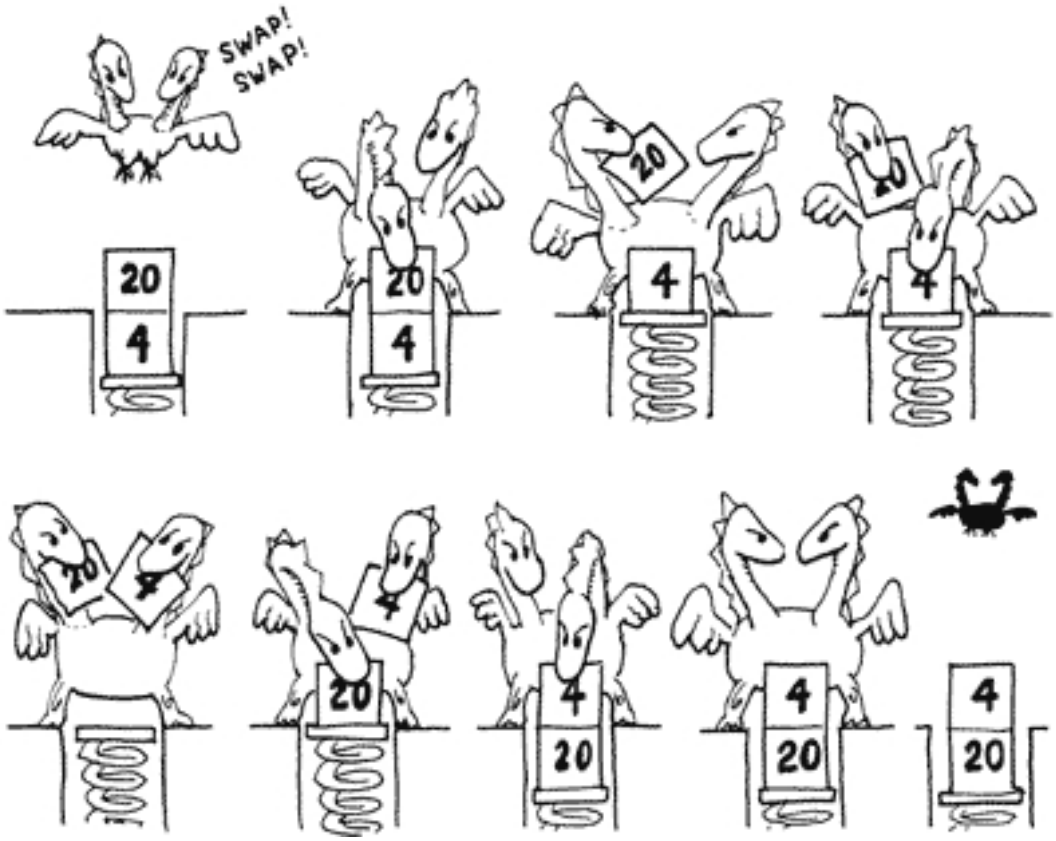


Abbildung 1: Der Stack bei Brodie

Link

http://www.forth.com/starting-forth/sf2/sf2.html

Bericht von der Forth–Tagung im April 2013

Michael Kalus

Bei meiner Anreise am Donnerstag schien tagsüber noch die Sonne und ermöglichte eine wunderbare klare Sicht auf die Alpen und die noch schneebedeckten Gipfel, die schon vom Allgäu aus zu sehen waren. Angekommen dann in Garmisch-Partenkirchen an der Olympia-Sprungschanze folgte der abenteuerliche Weg hinauf zum Tagungsort, dem Forsthaus Graseck. Ich kannte das ja noch nicht, war 2002, als die Forth-Gesellschaft hier schon einmal tagte, nicht dabei. Mit einer klitzekleinen Kabinenbahn für nur 4 Personen, *oder* Gepäck, ging's hinauf auf die Alm über der Klamm. Unterwegs blickt man nach rechts in die Schwindel erregende Tiefe hinunter in die Partnach-Klamm, und links steil hinauf in die Kalksteinschichten des Alpen-Aufschlusses, den die Partnach mit der Zeit in den Berg gefräst hat.

Abbildung 1: Die kleine Kabinenbahn.

Eingecheckt, und nach dem Abendessen traf man sich dann im Foyer, oder wer mochte, auch draußen auf der Terrasse, um die letzten Sonnenstrahlen noch zu genießen. Als die Dämmerung einsetzte, begann Erich schon mit dem ersten Workshop: EMACS —- die älteste IDE überhaupt. Danach klang der Abend aus mit verschiedenen Basteleien, auch Hardware wurde noch präpariert für den nächsten Tag, und Bernd sorgte dafür, dass auch das Internet dort oben auf dem Berg gut funktionierte.

Am nächsten Morgen, dem Freitag, nach dem Frühstück zunächst der Ausflug auf den Wank. Vorgesehen war eigentlich ein Zugspitz-Besuch, aber wegen dichter Wolken um den Gipfel herum war das im wahrsten Sinne des Wortes aussichtslos. Auch oben auf dem Wank noch Nieselregen und der Kopf in den Wolken. Aber die Fahrt mit der Wank–Bahn hinauf zum Gipfel–Gasthof, und ein Spaziergang dort oben, sowie die Fachsimpelei in der gemütlichen anregenden Atmosphäre des Lokals, das war den Ausflug dann doch wert. Zurück im Tagungshotel, gab es bayerisch deftiges Mittagessen. Um 15:00 Uhr dann Tagungsbeginn, zu dem auch vier Gäste aus den Niederlanden begrüßt werden konnten. Wir waren 24 Teilnehmer.

Abbildung 2: Beweismittel... :-)

In Ullis Beitrag „Forth in der Javascript-Welt“ ging es schon um die Frage des minimalen Forth-Kerns und darum, wie man möglichst viel eines Forth-Systems in Forth selbst schreiben kann. Auch die Frage der Sicherheit von Forth-Systemen wurde bedacht. Zu den Einzelheiten dieses Vortrages, und allen folgenden, siehe Bernds Zusammenstellung im Forth-Wiki. Nach einer Pause mit Kaffee und Kuchen ging es weiter in dem vollgepackten Tagungsraum, urgemütlich dort oben auf dem Berg. Der Ausflug in die Partnachklamm unterhalb des Tagungsortes musste leider abgesetzt werden, da wegen der Kühle in diesem Jahr noch zu viele Eiszapfen und Eiswände vom Winter den Weg durch die Klamm versperrten.

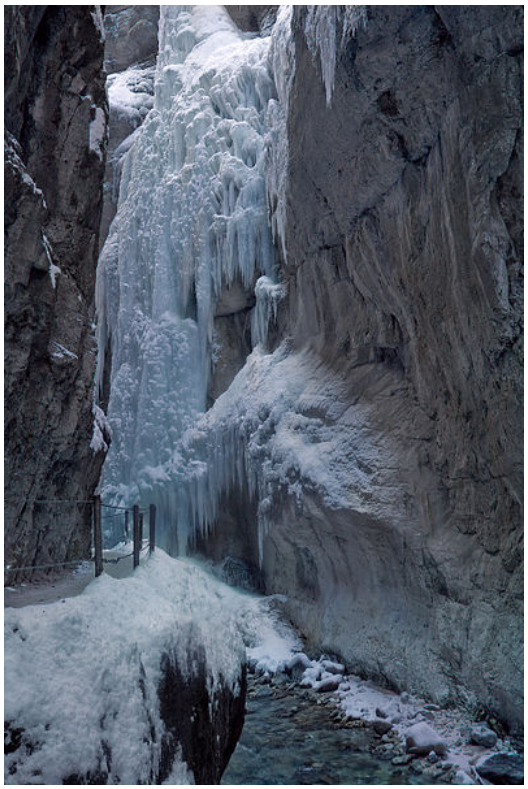


Abbildung 3: Eis in der Partnachklamm.

Beim Treffen wurde mir wieder sehr deutlich, dass die Forthtagung auch immer ein Update auf den neusten Stand der Rechnertechnik ist. Die Größe der Rechner, von ganzen Türmen, die man so mitschleppte in den früheren Jahren, bis heute zum handlichen Laptop, und den noch handlicheren Tablets oder Smartphones, ist doch erheblich geschrumpft.

Bernd berichtete über Net2O Application Layer, zeigte, dass sich was getan hat bei der Entwicklung des neuen Internets. Sein Peer–to–Peer–System ist im Fossil Repository mit aktuellem gforth realisiert. Einige Probleme, die Bernd im letzten Jahr mit diesem Netz noch hatte, insbesondere bei der Vermittlung und einigen Flaschenhälsen darin, sind inzwischen behoben worden. Da hat er gute Lösungen gefunden.

Hans berichtete dann über einen embedded Webserver, geschrieben in Forth. Dies macht auch kleine Geräte internetfähig. Gerald schloss sich dem an und zeigte FAT - Forth einfach ins Web bringen. Demonstriert wurde dies an einer Steuerung einer Lichtanlage für das DMX512–Protokoll. Nach anschließender Diskussion ging es zum Abendessen, und nach lang ausgebreitetem Tafeln traf man entspannt wieder bei den Workshops ein. Heinz erläuterte den Stand des Open Network Forth (ONF) heute, nach gut 30 Jahren Inbetriebnahme des Systems. Gegen 22:00 Uhr endete die Tagung und die üblichen Verdächtigen werkelten noch bis in die frühen Morgenstunden.

Am Samstag früh hatte es geschneit! Puderzucker auf der Alm und den Gipfeln ringsherum. Die Sicht immer noch sehr diesig. Für die Tagungsteilnehmer kein Problem, konzentriert auf die Sache ging es weiter mit dem nächsten Beitrag zum STRIP32, einem Forth–Prozessor eigener Machart von Willi. Er schilderte den weiteren Fortschritt im Design beim Wechsel vom 16- auf den 32-Bit-Prozessor. Wobei mich persönlich die Bits 0 und 1 interessierten, da er diese geschickt für schnelle Returns und Branches verwendet hatte.

Anschließend berichtet Erich über das Forth für den Parallax–Propeller–Chip und seine Schwierigkeiten, damit diese sehr eigenartige Architektur in den Griff zu bekommen. Nach der Kaffeepause stellte Albert aus den Niederlanden das No-Forth[1] vor, und Beethoven, eine musikalische Umsetzung von Beethovens Noten im Microprozessor mit einer eigenen Notation dafür. Leon, ebenfalls aus den Niederlanden angereist, berichtete danach über seinen GA144-Simulator. Den hat er in VECTOR geschrieben, und als IDE für den Umgang mit dem GA144 ausgelegt - ein beeindruckendes Werk. Motiviert war er durch die doch problematische Benutzung des ColorForth, das zwar zum Chip mitgeliefert wird, aber mit dem die Analyse des durchaus interessanten GA144 doch schwierig sei.

Zum Mittagessen diesmal eine deftige Brotzeit mit Wurst, Schinken, Käsesorten. Anschließend viel Zeit in der Lobby zum Austausch, den ich dazu nutzte, neue VD-Artikel einzuwerben – ich hoffe, es erinnern sich nun alle auch daran! Zum Nachmittag gab es einige Stegreifreden, so Jens zu Forth in der Ausbildung, mit anschließendem Brainstorming zum Thema unter den Teilnehmern, bei dem es sehr produktiv bis ideologisch zuging, letztlich aber Substantielles beigetragen wurde. Ich hoffe, dass Jens einen Beitrag darüber hier in der Zeitung bringt. Ullis Ankündigung am Ende dieses Blocks, dass die *EuroForth in Hamburg 2013* tagt, ging dann fast schon im Gedränge zur Kaffeepause unter. Es folgte Bernd mit der Demonstration des gforth auf Android-Tablet, bzw. Smartphone, mit einem kurzen Bericht darüber, wie er dies bewerkstelligt hat. Anschließend noch einmal der GA144 und ColorForth, wobei ich nur staunend zusehen konnte, was damit alles so möglich ist. Im Nachmittagsblock dann Anton, der Forth als Basis für einen portablen Sampler vorführte. Karsten stellte das JonesForth vor – und da war es endlich, das `NEXT` der Tagung! Das JonesForth ist gut dokumentiert und daher beliebt, auch um eigene Forth-Modelle zu implementieren. Das CamelForth von Brad und das daraus entstandene 4e4th reihen sich da ein, vorgestellt von mir selbst.

Zum Abend hin wanderte die Gesellschaft mit Fackeln um den Berg und war bald im Nebel verschwunden. Der Ausflug war wegen der wolkenverhangenen Gipfel dann nicht ganz so ausgedehnt, schien den Beteiligten dennoch Freude bereitet zu haben. Mich hat es dort nicht so hinausgezogen. Den Rest des Abends saß man beisammen und fachsimpelte. Am Sonntag dann die Vereinsversammlung – die Wolken verzogen sich endlich, und

[1] Benannt nach N(ijhof)–O(werkerk)

Abbildung 4: Gruppenfoto mit Gebirg' im Hintergrund

gaben den Blick auf die Berge frei. Das ganze wunderbare Panorama dort oben und die klare Luft wirkten sich günstig auf die Vereinsversammlung aus. Die Beschlüsse erfolgten flott. Der Bericht des Vorstandes zum Stand der Gesellschaft und der Finanzbericht und auch die Wiederwahl des Vorstandes gingen zügig vonstatten. Eine echte Kuhglocke sorgte für Ordnung und man beschloss pünktlich die Tagung im schönsten Sonnenschein. Die Details sind dem Protokoll zu entnehmen. Nach dem abschließenden gemeinsamen Mittagessen auschecken, also wieder runter mit der kleinen Kabinenbahn, quasi Koffer für Koffer und Grüppchen für Grüppchen, sodass auch hier noch Zeit blieb, zu fachsimpeln und sich gemütlich zu verabschieden. Die Rückreise war für mich entspannt, ich hatte mir einen weiteren Rückreisetag reserviert, um ins Ruhrgebiet zu kommen. Ich hoffe, auch alle anderen sind heile wieder angekommen, und verbleibe mit herzlichen Grüßen

Euer Michael

Links

`http://www.forth-ev.de/` –> Menü *Photos* –> Forth-Tagung 2013
`http://www.forth-ev.de/wiki/doku.php/events:tagungen` –> Forth-Tagung 2013 im Forsthaus Graseck/Garmisch-Partenkirchen

Abbildung 5: Und das nächste Treffen ist am Stand der Forth-Gesellschaft...

Forth–Gruppen regional

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

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

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

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

Gruppengründungen, Kontakte

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

μP–Controller Verleih

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

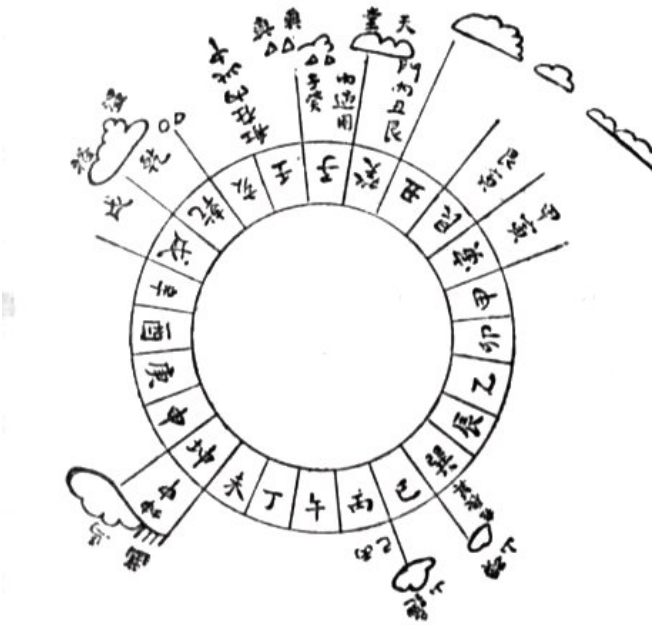
Spezielle Fachgebiete

FORTHchips (FRP 1600, RTX, Novix)	**Klaus Schleisiek–Kern** Tel.: **(0 40) – 37 50 08 03** (g)
KI, Object Oriented Forth, Sicherheitskritische Systeme	**Ulrich Hoffmann** Tel.: **(0 43 51) – 71 22 17** (p) Fax: **– 71 22 16**
Forth–Vertrieb volksFORTH ultraFORTH RTX / FG / Super8 KK–FORTH	Ingenieurbüro **Klaus Kohl–Schöpe** Tel.: **(0 82 66)–36 09 862** (p)

Termine

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

03. August 2013: Maker Faire, Hannover
25.–27. September 2013: Forth200x meeting
27.–29. September 2013: EuroForth 2013 conference
09.–10. November 2013: OpenRheinRuhr, Oberhausen



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

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

EuroForth 2013
29th EuroForth Conference
Haus Rissen, Hamburg, Germany
September, 27th to 29th, 2013

HAUS RISSEN HAMBURG
Internationales Institut für Politik und Wirtschaft
www.hausrissen.org

EuroForth is consistently the best international Forth conference, attracting people from as far away as Australia, South Africa and the USA. EuroForth 2013 will be held at Haus Rissen, Hamburg.

This year's EuroForth will be organized by Klaus Schleisiek and Ulrich Hoffmann. To book or to get further details, please contact Klaus Schleisiek `kschleisiek@send.de`

The conference will be preceded by a Forth 200x standards meeting.

July 1:	Deadline for draft papers (academic stream)
August 10:	Notification of acceptance of academic stream papers
September 18:	Deadline for camera-ready paper submission (academic and industrial stream)
September 25-27:	Forth200x meeting
September 27-29:	EuroForth 2013 conference
September 29-30:	4th day

Links

- EuroForth 2013 Home page `http://www.complang.tuwien.ac.at/anton/euroforth/ef13/`
- Call for Papers `http://www.complang.tuwien.ac.at/anton/euroforth/ef13/cfp.html`
- EuroForth Home `http://www.complang.tuwien.ac.at/anton/euroforth/index.html`