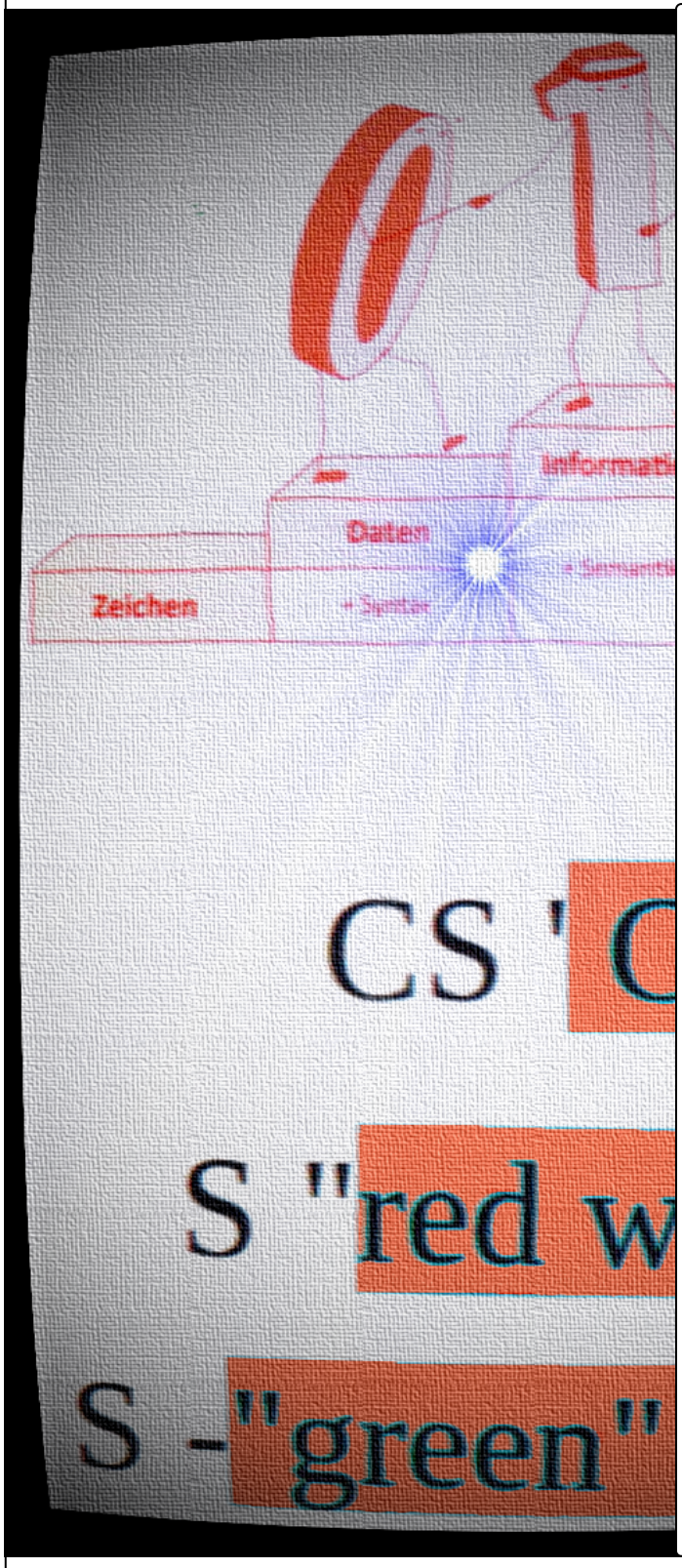
Vierte Dimension

1/2017

Das Forth–Magazin

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

In dieser Ausgabe:



What's in a name?

µC–Pakete

The Sockpuppet Forth to C interface

SSD1306 — A 128x64 Bit OLED Display

v4th

ROM und RAM in einem ROMforth

Vintage Computing - FORPS

Easy Forth - online ebook in JavaScript

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

Seit 2001 entwickeln und vertreiben wir unter dem Markennamen "Servonaut" Baugruppen für den Funktionsmodellbau wie Fahrtregler, Lichtanlagen, Soundmodule und Funkmodule. Unsere Module werden vorwiegend in LKW-Modellen im Maßstab 1:14 bzw. 1:16 eingesetzt, aber auch in Baumaschinen wie Baggern, Radladern etc. Wir entwickeln mit eigenen Werkzeugen in Forth für die Freescale-Prozessoren 68HC08, S08, Coldfire sowie Atmel 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

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.

KIMA Echtzeitsysteme GmbH

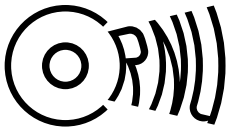
Güstener Strasse 72 52428 Jülich
Tel.: 02463/9967-0 Fax: 02463/9967–99
`www.kimaE.de` info@kimaE.de

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.



Cornu GmbH
Ingenieurdienstleistungen
Elektrotechnik

Weitlstraße 140
80995 München
`sales@cornu.de`
`www.cornu.de`

Unser Themenschwerpunkt ist automotive SW unter AutoSAR. In Forth bieten wir u.a. Lösungen zur Verarbeitung großer Datenmengen, Modultests und modellgetriebene SW, z.B. auf Basis eCore/EMF.

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`

Leserbriefe und Meldungen 5

What's in a name? 12
Albert Nijhof

µC–Pakete 15
Gerald Wodni

The Sockpuppet Forth to C interface 19
Stephen Pelc

SSD1306 — A 128x64 Bit OLED Display 23
Willem Ouwerkerk

v4th 25
Vic Plichota

ROM und RAM in einem ROMforth 28
Albert Nijhof

Vintage Computing - FORPS 31
Jostein Skjelstad

Easy Forth - online ebook in JavaScript 34
Nick Morgan

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,

vermutlich ist es Euch schon aufgefallen, dass das Internet der Dinge in dichtem Bodennebel gehüllt daliegt, und nicht in der Wolke schwebt. So ist denn Forth für viele Leute eher eine nebulöse Angelegenheit. Das zu ändern, hatte sich die Forth–Gesellschaft vorgenommen, damals, vor nun 33 Jahren. Wer hätte gedacht, dass diese Aufgabe schon so lange verfolgt wird? In der Satzung des Vereins könnt ihr das nachlesen : www-forth-ev.de
Unsere Webseite soll eine neue Grundlage bekommen. So wurde es beschlossen auf der Tagung 2016. *Gerald Wodni* hat das hinter den Kulissen inzwischen vorangetrieben, ich durfte schon einen Blick darauf werfen. Sieht gut aus. Und vor allem, viel simpler als Geeklog, unsere bisherige Maschine dafür. Geht auch für Smartphones. Nach Ostern, auf unserer diesjährigen Tagung dürfte es vorgestellt werden. Und dann geht es an die Arbeit, den ganzen Inhalt dorthin zu verlegen. Freiwillige vor! Und für das Bibliotheks-Vorhaben liegt auch die Software vor: μC-Pakete einstellen und verbreiten, wird auch so eine Aufgabe für viele Helfer sein. Herumsprechen sollte es sich schon mal, wie das geht — also: Weitersagen!
Simpel muss auch ein Forth-Compiler sein, wenn er in sehr kleine MCUs passen soll. eForth und noForth sind so klein, rund 4 KiB reichen denen schon. *Albert Nijhof* beschreibt, wie sein Datenmanager es schafft, den Forthkern so Kompakt zu halten und dennoch einen gewissen Komfort zu bieten — erstaunlich! Und dazu gehört auch der Umgang mit den verschiedenen Speicherbereichen. Sein Ansatz: Explizit adressieren durch eigene Forth–Worte.
Stephen Pelc stellt dafür Sockpuppet zur Verfügung. Damit lassen sich C–libraries einbinden in das MPE–Forth. Eine wunderbare Sache. Die Systeme sind allerdings nicht sooo klein, dass sie auf den ganz kleinen MCUs laufen könnten, aber mit ARM–boards ist man schon dabei.
Damit man in diese kleinen Dinger auch hineinsehen kann, hat *Willem Ouwerkerk* ein handliches OLED-Display angebunden. Es ist im Egel–Projekt enthalten, einer erstaunlichen Sammlung, mit der in den Gebrauch der MCU mittels noForth eingeführt wird. Eine sehr gute Unterstützung, wenn man Hardware erkunden will.
Vic Plichota zeigt, wie es noch kleiner geht! Dabei wird in Forth entwickelt, doch in die MCU assembliert. Dort liegt dann nur noch die reine Anwendung, ganz ohne Forthkern. Ausprobieren konnte ich das noch nicht, und bin gespannt auf eure Berichte dazu.
Jostein Skjelstad führt uns schließlich zu den Anfängen zurück. Altes Spiel auf neuem gforth, eine unterhaltsame Sache. Und angespielt hab ich dann noch Easy Forth, das Forth–Online–Ebook in JavaScript von *Nick Morgan* — wunderbar. Aber lest selbst!

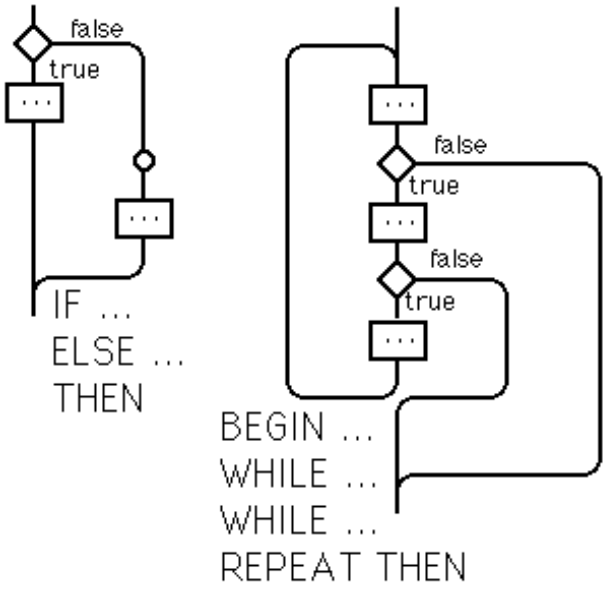
So, und im kommenden Frühjahr, gleich nach Ostern, wenn sich die Nebel gelichtet haben, ist unsere *Forth–Tagung* im Westen des Landes — in einem Atomkraftwerk, das nie eins wurde. Meldet Euch und Eure Beiträge jetzt an, denn bald ist es schon soweit!

Bis dahin!
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-2017-01`

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

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



Begin Again

Wil Baden hat sich über die Jahre viel mit Forth beschäftigt und darüber publiziert. Unter anderem verdanken wir ihm die Erkenntnis, dass man mit den 7 Wörtern `BEGIN AGAIN UNTIL IF AHEAD THEN` und `CS-ROLL` alle Kontrollstrukturen bauen kann, die man auch mit Sprüngen und bedingten Sprüngen bauen kann, nur dass man keine Labels oder Sprungadressen verwalten muss; man kann daher mit geringerem Aufwand weitere Kontrollstrukturen wie `BEGIN ... WHILE ... REPEAT, IF ... ELSE ... THEN` bauen, während Compiler für andere Sprachen da typischerweise einen Labelgenerator in Stellung bringen müssen.

Weitere Beiträge sind sein Forth-System *thisForth*, seine Sammlung nützlicher Wörter (*toolbelt*), und seine Implementierungen von Kompressions– und kryptographischen Algorithmen in Forth. AntonErtl

Wil Baden ist 2016 für immer von uns gegangen. Viele seiner Forth–Programme können auf folgendem Link gefunden werden:

http://www.wilbaden.com/neil_bawd/index.html

Die Forth Interest Group gedenkt ihrer Verstorbenen mit einer eigenen Seite:

http://www.forth.org/memory.html

Geflügelte Worte

> "When you've seen one Forth, you have seen one Forth."

Wil Baden interjected this at FORML, November 24, 1984, from the audience, responding to remarks of the speaker.

It's from an old family saying, going back to 1963. His six year old daughter corrected him after he said "When you've seen one, you have seen them all."

"No, Daddy. When you've seen one, you have seen one." mk

...

> „ANS Forth standard was formed to improve portability, but that has not always been well received, because portability is not necessarily the most important thing in a language where the application can be completely re-written much faster than a C program can be debugged."

Garth commented that on *Forth: The Hacker's Language* in response to Artenz,. http://hackaday.com/ db

Russische Ausgabe des Starting Forth

Auch im Osten ist Forth verbreitet. Das Buch kann als offizielle Ausgabe angesehen werden, enthält es doch ein Vorwort von Chuck Moore und Leo Brodie an die Soviet-Leser. Man kriegt es dort: http://www.nncron.ru/download/sf.pdf

Der Titel auf dem Buchdeckel lautet wörtlich übersetzt: L.Broudie, Anfänglicher Kurs Programmierung auf Sprache Forth. Der Name Brodie wurde dabei lautmalerisch ins Russische übertragen, damit er ausgesprochen wie im Englischen klingt.

Punyforth

Eine weitere, von Forth inspirierte, einfache stack–basierte Programmiersprache ist Punyforth. Sie compiliert indirect-threaded code, zielt auch auf das Internet der Dinge, und wurde für den ESP8266 gemacht. Es gibt wohl auch eine Version für x86 (Linux) und ARM (Raspberry PI), doch seien das nicht die primär unterstützten Zielprozesoren.

> Some of the differences: Punyforth is case sensitive. Strings are null-terminated. Strings are created and printed differently (str: "foobar", print: "foobar" instead of s" foobar", ." foobar"). Parsing words are ended with a colon character by convention (including variable:, constant:, create: does>). Defining a word in terms of itself results recursion by default (use the override word to alter this behaviour). Curly brackets denote quotations instead of locals. Punyforth supports exception handling, multitasking, socket and GPIO APIs and comes with a UART and a TCP REPL.

https://github.com/zeroflag/punyforth cs

Mecrisp-Across

So winzigklein ein gemütliches Forth für einen Mikrocontroller im Vergleich zu vielen anderen Programmen auch sein kann, so ist der Größe doch eine untere Schranke gesetzt, sollen nicht allzu viele Funktionen verloren gehen und auch noch Platz für Forth–Programme bleiben. Um einen *MSP430* zu erobern, sollte dieser mindestens 16 KiB haben, während ein Forth–Prozessor wie der *J1*, dessen Befehlssatz genau für unsere Lieblingssprache abgestimmt ist, mit 8 KiB gerade so auskommt. Irgendwo in dieser Größenordnung scheint die untere Grenze zu liegen, jenseits derer nur noch ein paar Exoten gedeihen.

Doch was ist mit den ganz kleinen Microcontrollern wie dem *MSP430F2012*, der mit 2 KiB Flash und 128 Bytes RAM ausgestattet ist und den ich so gerne für eine kleine Samstagabendbastelei verwende?

Um auch die allerkleinsten Käfer mit Forth auszustatten, ist Mecrisp-Across entstanden. Es ist ein *Crosscompiler*, welcher selbst auf Mecrisp—Stellaris in einem großen ARM läuft und sich über die SWD–JTAG–Schnittstelle mit einem kleinen MSP430 unterhält, um Portzugriffe in die Zielhardware weiterzuleiten, während die Forth–Programme interaktiv entwickelt werden können. Dafür braucht Mecrisp–Across im Gegensatz zu vielen anderen Crosscompilern *gar keinen Speicher im Zielchip* — es ist sogar möglich, einfach nur so mit der Hardware zu experimentieren, während im Flash ein ganz anderes Programm abgelegt ist und bleibt. Echtzeitfähig ist dieser interaktive Modus so natürlich nicht und unterstützt auch keine Interrupts, ist aber dafür sehr komfortabel und zum Testen der angeschlossenen Elektronik hoffentlich ausreichend.

Erst wenn alles fertig ist, wird das Binary für den Zielchip erzeugt. Und damit dieses winzigklein wird und viel in den sehr engen Speicher passt, hat Mecrisp-Across stärkere Optimierungen an Bord als alle meine Forth-Compiler je zuvor und beherrscht als erster mir bekannter Forth—Compiler *globale Optimierungen*:

- Konstantenfaltung
- Registerallokator für Daten- und Returnstack, beherrscht verschiedene Adressierungsmodi
- Beliebig wählbarer kanonischer Zustand
- Registerallokation über Kontrollstrukturen hinweg, ohne kanonischen Rückfall
- Konstantenfaltung in bedingte Sprünge hinein, Umwandlung in unbedingte Sprünge und Dead-Code-Elimination
- Call-Trace ausgehend von den Interruptvektoren, um herauszufinden, welche Definitionen verwendet werden
- Definitionen, die nur einmal verwendet werden, werden automatisch inline eingefügt und mit dem lokalen Registersatz überoptimiert
- Verfolgung der Registerbenutzung, um maßgeschneiderte Interrupt-Rahmen zu generieren

Anders ausgedrückt: Wer mag, kann in seiner Testphase viele, viele Definitionen für das Target interaktiv zur Verfügung haben — aber nur die, die dann in der Applikation auch verwendet werden, werden tatsächlich einkompiliert. Und von den Definitionen, die schließlich den Weg in den Zielchip finden, auch nur die Teile, die tatsächlich erreicht werden können. Trifft beispielsweise eine Konstante auf eine IF- oder CASE-Struktur, werden diese vom Compiler vollständig aufgeribbelt! In Kombination mit dem automatischen Inline ist das eine sehr mächtige Optimierung.

Das Ergebnis kann anschließend direkt in den Zielchip geflasht oder als HEX-Image ausgegeben werden.

Doch leider gibt es auch zwei Wermutstropfen. Der Eine: Noch ist Mecrisp-Across in einem sehr frühen experimentellen Stadium, die Entwicklung ist zäh und wird noch eine ganze Weile dauern. Und der Zweite ist: Es gibt in Mecrisp-Across kein `'` („tick“) und kein `EXECUTE`. Denn was `der` Crosscompiler generiert, hat mit traditionellem Forth kaum noch etwas gemein - wo, wie und ob eine Definition am Ende in Maschinensprache existiert, lässt sich kaum im Voraus ermitteln. Das ist der Preis für globale Optimierungen und den Vorstoß in Neuland für Forth, aber vielleicht findet sich dafür in Zukunft noch eine Lösung.

MatthiasKoch

Various shades of Forth

Auf jeelabs.org beschäftigt sich Jean-Claude Wippler weiterhin sehr aktiv mit *environmental electronics*. Ich finde es spannend zu erleben, wie er sich dabei an Forth herantastet. Bin gespannt, was da noch draus wird.

> This week's episode will cover a variety of aspects related to Mecrisp Forth:
>
> - Using Linux i.s.o. a SerPlus
>
> - Forth over USB on STM32F103
>
> - Performance and I/O toggling
>
> As always, these posts are spread out over several days, so don't despair if you click a link early on and see a server 404 error ...

http://jeelabs.org/ db

A Gathering of Magicians

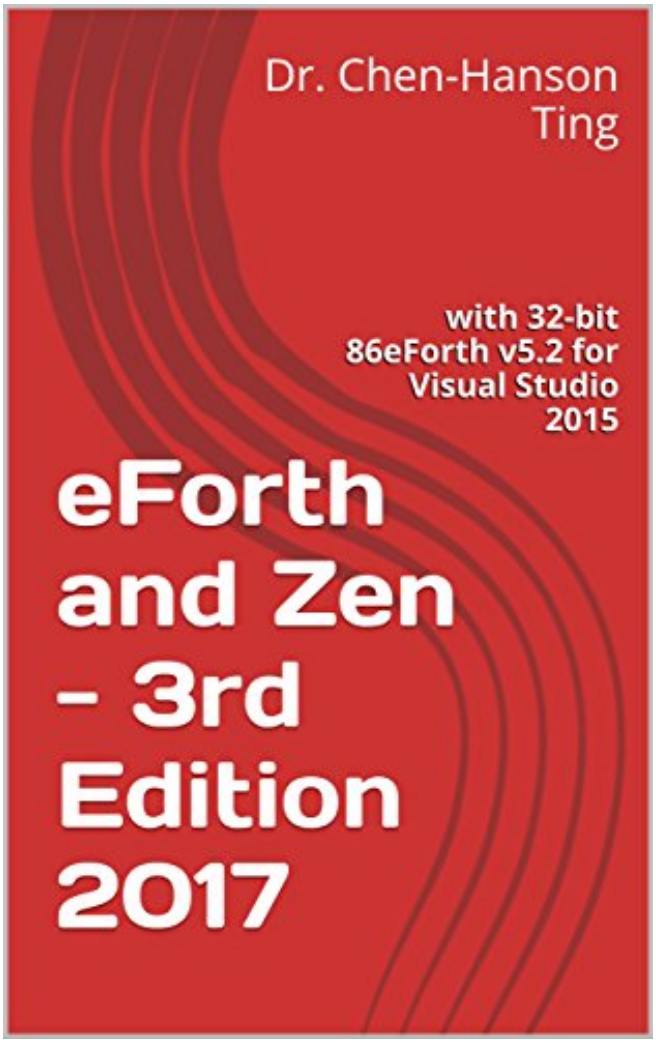
William Wong schrieb in ALT.EMBEDDED, einer Rubrik in der ELECTRONICDESIGN:

> Some of you may have received an email recently about the 2014 nominees for the Electronic Design Engineering Hall of Fame. Last year we inducted people like Bjarne Strostrup (see "Bjarne Stroustrup: C++ Creator Keeps Developing") and Thompson, Ritchie and Kernighan (see "Thompson, Ritchie, And Kernighan: The Fathers Of C").

Im Dezember des Jahres dann ging es um Roger Amidon, VALDOCS und den Epson QX-10.

> Underneath, the system and some applications like Mail were written in macro assembler. That thing called C wasn't really available. The other applications were written in *Forth*. I actually worked on the *Valdraw* drawing program that was written in Forth so I saw both sides of the system. The amazing thing was it was all packed into a floppy disk system that pushed the hardware past its limits ...

http://electronicdesign.com/blog/rising-star db



Ting's Electronic Forth Bookshelf

A couple of years ago, I closed my website www.offete.com and stopped distributing my publications on–line. Nevertheless, these publications still exist on *my electronic bookshelf*. If you need any of them, please send me a request at chenhting@yahoo.com.tw, I will sent it in a return email, and also bill you by a PayPal invoice. I know, we are in the 21st century now. You cannot do anything without a website. But, at least I got rid of lots of paper, and the snail mail.

Juergen Pintaske twisted my arm to get *Footsteps in an Empty Valley* updated from a printed copy, which was edited on an old word processor TMAKER on a CP/M machine and printed with a Diablo daisy wheel printer. Files got lost with the CP/M machine. I had to scan all the pages and used OCR to recover the text. The hardest part was CHUCK MOORE'S source code of CMFORTH, which he printed on an Epson dot matrix printer with a worn ribbon. Lots of the dots disappeared through copying processes. I tried my best to bring back the code, but couldn't be entirely sure. I hope nobody will use the code for any purpose other than reading.

Well. Let me know if you have any question.

Chen-Hanson Ting, San Mateo, California (February, 2017)

Ting's PDF Books

After I learnt a Forth system, I always tried to document it, so I could teach other people how to use it. So I wrote about *polyForth, figForth, F83, F-PC, and cmForth*. When Win32Forth came along, I gave up, because it was too large and too complicated. I then focused on developing eForth for microcontrollers. After retirement, I cleaned out the books off my shelves. People still asked for them, so I converted some to pdf files. Here is the list of available titles:

4001 Footsteps in an Empty Valley, 4th Ed., $15 Description of the first Forth chip NC4000 from Novix, and Chuck Moore's cmForth for it. cmForth was the simplest and most compact specification of a real Forth system for a real Forth computer. It contains a complete Forth system with a target compiler, an optimizing assembler, and a serial disk driver. Required reading for all Forth programmers.

1010 Systems Guide to figForth, 3rd Ed. $15 The most authoritative treatise on how's and why's of the figForth Model developed by Bill Ragsdale. Internal structure of the figForth system. Very detailed discussions on the inner interpreters and the outer (text) interpreter of Forth.

1003 Inside F83, $15 Everything you want to know about the Perry-Laxen F83 system but were afraid to ask. 288 packed pages divided into 4 parts: Tutorial on F83 system, Kernel, Utility, and Tools. It is based on 8086 vi F83 Version 2.1 for the IBM-PC, but useful as a reference manual for all other (8080 and 68000) F83 systems.

1008 F-PC Technical Reference Manual, $15 Narration on all words in the kernel and tools of F-PC, a practically useful Forth system for applications on PC. Functional description of the utilities and applications. Valuable guide to F-PC internals and assembly coding on segmented 80386 architecture.

1013 .eForth and Zen, 3rd Ed. $15 Complete description and exposition of the eForth Model: kernel, high level words, interpreters, compiler and utilities. Comparison of Forth and Zen, their similarities in simplicity and understanding. It is update based on 32-Bit 586 eForth v5.2 for Visual Studio Community 2015. It is in an assembly file as a C++ console project. It uses indirect thread model so that new colon words can be added to the .data segment. It is optimized with 71 code words and 110 colon words.

1015 Firmware Engineering Workshop, $15 A tutorial in 4 parts for building firmware for embedded systems, based on enhanced eForth. Hands-on experiments using CT100 Lab Board with 8051. 8086 eForth 2.02 and 8051 eForth 2.03 are included with the original eForth 1.01 Models for 8086 and 8051.

eForth Implementations I had always looked for low-cost microcontroller kits to teach people Forth. Over the years, these kits were getting cheaper and more powerful, and I ported eForth to a lot of them. I had lots of fun with them, and I enjoyed seeing others having fun (and making useful products) as well. eForth captures the essence of Forth, as an universal programming language for small, embedded systems. These eForth implementations are distributed with source code and substantial documentation.

2152 ADuC ARM7 eForth, $25 eForth for ADuC7020 MicroConverters from Analog Devices. It is written in ARM7 assembler on a Keil IDE. It uses the ARM7 link register for threading, and is fully optimized to make the best use of ARM7 core and analog peripherals integrated in this true microcontroller.

2153 SAM7 ARM7 eForth, $25 eForth for AT91SAM7X256 microcontroller from Atmel. It is in ARM7 assembler on Keil uVision3 RealView IDE. It uses the DBGU serial port to interact with user. Olimex's SAM7-EX256 Board has a very interesting color LCD module. This eForth has graphic primitives to drive the LCD display.

2154 cEF Version 1.0, $25 cEF is a Forth implementation based on eForth Model, and compiled by gcc compiler in Cygwin on a PC. The underlying Virtual Forth Machine has the standard 33 machine instructions defined in the original eForth Model. It is target to microprocessor without floating point coprocessor, and uses only integer arithmetic operations.

2155 cEF Version 2.0, $25 cEF is a Forth implementation based on eForth Model, and compiled by gcc compiler in Cygwin. The Virtual Forth Machine has 64 machine instructions. Multiplication and division are implemented using double arithmetic floating operations. It is highly optimized to take advantages of recent microprocessors with floating point coprocessors.

2157 eForth for STM8S,$25 STM8S is an 8 bit microcontroller from STMicroelectronics. ST is distributing a STM8S-Discovery Board for less than $10. It is an excellent kit to learn microcontroller programming. Now, a good Forth experimental kit is available for high school students.

2159 328eForth for Arduino Uno, $25 This is a very efficient implementation of eForth for ATmega328P microcontroller used on Arduino Uno Kit. It is using Subroutine Thread Model. It uses tools in NRWW memory to compile new words in main RWW flash memory. It allows you to build turnkey systems for commercial applications. It requires a flash programming tool.

2162 ceForth_328 for Arduino Uno, $25 This is an Arduino sketch which can be compiled and uploaded by Arduino IDE. The Forth Virtual Machine is coded in C, and the Forth dictionary is imported as a data array. The Forth dictionary can be extended into the RAM memory, so you can add new commands to this system. The dictionary is produced by a metacompiler running under F#. The source code of the metacompiler is included for you to enhance this system.

2164 430eForth for TI LaunchPad, $25 This is a Forth system for the MSP430G2553 microcontroller used on the LaunchPad from TI. It is a 16-bit Forth implementation to be assembled by the Code Composer Studio 5.2. It makes the best use of the 16 KB of flash memory, leaving about 10 KB for your applications.

2165 STM32eForth720 for STM32 F4 Discovery, $25 This eForth is for STM32F407 chip on STM32 F4 Discovery Kit from STMicroelectronics. This chip has 1 MB flash memory, 192 KB of RAM, and a ton of interesting IO devices. STM32 is no longer an ARM7 chip, but a THUMB2 chip. STMeForth720 is optimized for the new environment.

2166 430eForth v4.3 for TI LaunchPad, $25 This is a Forth system optimized for the MSP430G2553 microcontroller used on the LaunchPad from TI. It is changed from a subroutine threaded model to a direct threaded model, faster and more compact.

2167 8086 eForth Version 2.03, , $25 Enhanced 32-bit eForth for 80586 running under Visual Studio Community 2015. It is assembled by MASM buried under C++ as a console project. Now you can evaluate the eForth model conveniently in latest Windows environment.

2171 32-Bit 586eForth v.5.2 for Visual Studio, $25 It is an assembly file in a C++ console project on Visual Studio Community 2015. It requires library files supplied by Kip Irvine for Windows services. It uses indirect thread model so that new colon words can be added to the data segment. It is optimized with 71 code words and 110 colon words. Now you can test drive eForth on newer Windows PC.

2172 espForth for ESP866 Chip, $25 ESP8266 is a 32-bit microcontroller with integrated WiFi antenna and software drivers. Arduino IDE can compile and upload applications to it. espForth is an Arduino sketch which allows Forth commands to be sent to ESP8266 remotely as UDP packets. IoT for fun!

VHDL Forth Chip Designs. I had used VHDL to design Forth processors and tested them on FPGA's. They included a 16-bit processor eP16 and a 32-bit processor eP32. I ported eForth to these chips for design verification. In 2016, we ran a CPU Design Workshop in Silicon Valley Forth Interest Group, and I used designs of Intel 8080 and DEC PDP1 as exercises. It was interesting that eForth was used here as test benches, which were much more difficult to design than CPU themselves.

2163 eP16 in VHDL for LatticeXP2 Brevia Kit, $25 eP16 is a 16 bit microcontroller. It was implemented on LatticeXP2 Brevia Development Kit with LatticeXP2-5E FPGA. It included a CPU module, a UART module and a GPIO module. An eForth metacompiler producing eForth RAM image is included with all source code.

2158 eP32 in VHDL for LatticeXP2 Brevia Kit, $25 eP32 is a 32 bit microcontroller. It was implemented on LatticeXP2 Brevia Development Kit with LatticeXP2-5E FPGA. It includes a CPU module, a UART module and a GPIO module. An eForth metacompiler producing eForth RAM image. It is the best Forth engine design on the cheapest FPGA kit. All VHDL files and eForth files are included.

2169 80eForth202 for eP8080 Chip, $25 eP8080 was a CPU model used in SVFIG FPGA Design Workshop. It recreated an i8080 chip in FPGA. 80eForth202 was the Forth system embedded in VHDL for design verification and to help debugging the chip. The eForth RAM image was derived from 86eForth v2.2 and Z80eForth by Ken Chen, assembled with MASM.

2170 PDP1eForth for ePDP1 Chip, $25 ePDP1 was another CPU model used in SVFIG FPGA Design Workshop. It recreated a PDP1 chip in FPGA. PDP1eForth was the Forth system embedded in VHDL for design verification and to help debugging the chip. It was derived from eP16, and used a metacompiler in F# to create eForth dictionary to initialize RAM memory.

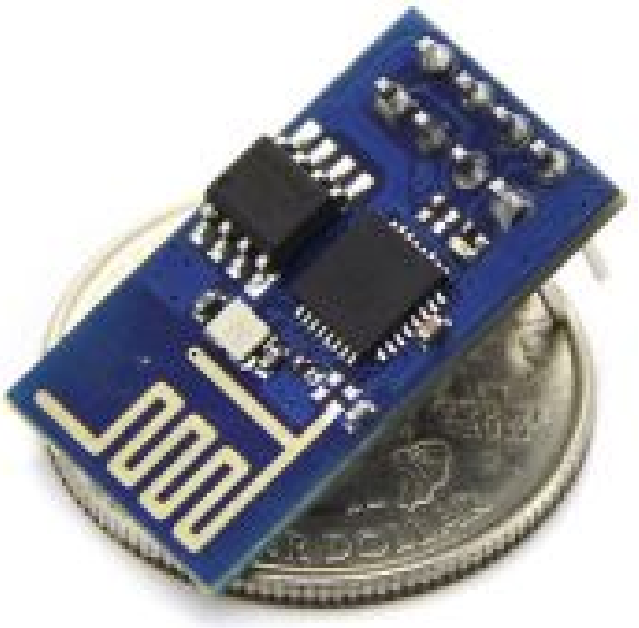
Ting's eBooks

There are as well some eBooks available as part of *Juergen's Forth Bookshelf* at

https://www.amazon.co.uk/Juergen-Pintaske/e/B00N8HVEZM:

- FIG-Forth Manual – including the 1802 IP running on Lattice ICE40 8k FPGA DevBoard
- eForth Overview
- Zen and the Forth Language
- Footsteps In An Empty Valley — In this eBook you find the background, how and why Forth was designed.
- eForth and Zen - 3rd Edition 2017; with 32–bit 86eForth v5.2 for Visual Studio 2015.

Der WiFi Chip ESP8266



Das ESP8266 von dem Hersteller Espressif ist ein programmierbarer WLAN-SoC mit UART- und SPI-Schnittstelle. Darin werkelt eine 80 MHz Xtensa–LX3–32bit–MCU, mit TCP/IP–stack, GPIO–pins und viel Flash–memory. WLAN-Funkmodule mit ESP8266 sind ab 3€ verfügbar. Die UART-Schnitttstelle ermöglicht eine einfache Integration in Mikrocontrollerprojekte, beliebt im IoT und der Heimautomation. http://www.mikrocontroller.net/articles/ESP8266 mk

eForth for ESP8266

Ting Chen Hanson hat Ende 2016 noch schnell sein eForth dorthin portiert. Was damit schon alles geht, wird sich zeigen. Anfragen bitte direkt an ihn senden: chenhting@yahoo.com.tw

> I got eForth to run on ESP8266 WiFi chip. I was able to run it through UDP packets. I am still learning WiFi tricks. I hope to show that I can compile and execute new words on ESP8266 remotely.

Quelle: SVFIG December Meeting, December 17, 2016 THIRD Saturday 13:00 --- Forth for WiFi Chip ESP8266 --- C.H. Ting db

Starting Programming in Forth - no soldering, „no wires“

What is the real minimum you need to start programming in Forth? Cannot solder? Anybody can put this together in max 30 mins:

- 1 Breadboard
- 1 programmed MSP430G2553 in DIL package
- 1 resistor 33k (Reset)
- 5 LEDs, (Switches not connected yet)
- 8 resistors 10 Ohms[1].

Mount the Micro; bend pins a bit to ensure that all 20 pins slide easily into the breadboard. Mount Reset Resistor 33k across the chip from pin 1 to pin 16. 4 resistors to get connected to the USB-to-TTL module — GND pin 20, PLUS pin 1, RX pin 3, TX pin 4. One resistor connects pin 20 of the IC to the ground coming from the USB-to-TTL, as one resistor was not long enough – use 2 twisted together. In the picture pin 20 is in row 17 of breadboard, this is GND for all LEDs too.

LED 1 from pin 14 to GND, always on with 4e4th, flashing using MPE VFX LITE software.

LED2 pin 18 to GND – Port2 bit 7.

LED3 pin 19 to GND – Port2 bit 6.

LED4 pin 12 to GND – Port2 bit 5.

LED5 pin 13 to GND - Port2 bit 4.

To save on current limiting resistors, the LEDs are connected here directly from output to GND[2].

Connect the USB-to-TTL, start the IDE, here I used 4e4th, and Start programming, to control the LEDS, see pictures; Switches have not been connected yet, could be: IN0 pin 8, IN1 pin9 ... Connected to GND to see the uMMT examples AND, OR, XOR, NOT ... Needs enabling the internal resistors, or add 2 external 1k to 33k to PLUS.

Enjoy! Juergen

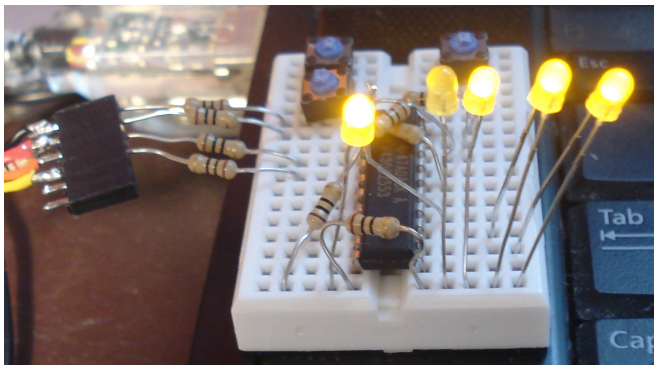


Abbildung 1: mounted micro ...

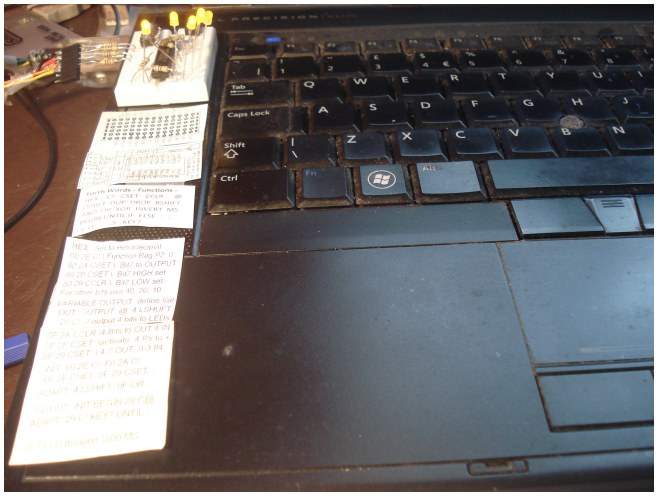


Abbildung 2: This tiny reference list is all you need ...

Hacker

We at TMRC use the term "hacker" only in its original meaning, someone who applies ingenuity to create a clever result, called a "hack". The essence of a "hack" is that it is done quickly, and is usually inelegant. It accomplishes the desired goal without changing the design of the system it

[1] (Why? As I could not find wire quickly, what else can be used? This time I just used the resistors that never get used really: 10 Ohms. Can be replaced by wire ...

[2] Yes, those ports can handle that!

is embedded in. Despite often being at odds with the design of the larger system, a hack is generally quite clever and effective.

This original benevolent meaning stands in stark contrast to the later and more commonly used meaning of a "hacker", typically as a person who breaks into computer networks in order to steal or vandalize. Here at TMRC, where the words "hack" and "hacker" originated and have been used proudly since the late 1950s, we resent the misapplication of the word to mean the committing of illegal acts. People who do those things are better described by expressions such as "thieves", "password crackers". or "computer vandals". They are certainly not true hackers, as they do not understand the hacker ethic.

Quelle: http://tmrc.mit.edu/hackers-ref.html

Der The Tech Model Railroad (TMRC) ist eine studentische Aktivität am Massachusetts Institute of Technology (MIT), die sich seit den Jahren 1946-1947 entwickelt. Er ist einer der älteten Clubs am MIT. Der TMRC wird im ersten Kapitel des Buches „Hackers“ von Steven Levy (New York: Anchor Press/Doubleday, 1984) als eine, wenn nicht *die* Quelle der Hacker-Kultur bezeichnet. mk

Link: http://web.mit.edu/

Egel project

The Egel project consists of about *50 elementary examples* of hardware control with noForth on the MSP430. In each example you find a file with forth code, documentation and links to more information on the internet.

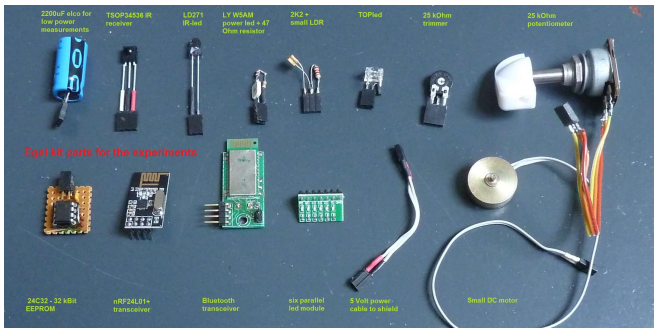


Abbildung 3: Egel kit parts for experiments

It is our aim to lower the hurdles that beginners with the MSP430 MPU find on their way to hardware programming using noForth software. With the many examples in the Egel project we hope to make it easier to understand the Texas Instruments documentation.

https://noforth.bitbucket.io/site/egel%20for%20launchpad.html?

noForth updates February 2016[3]

New:

- Low Power noForth for MSP430G2553 boards. In Low Power noForth all 'wait-loops' are replaced with 'sleep-until-interrupted' which relatively saves a lot of energy. Low power noForth is marked with a dash: noForth C- and noForth V-.
- noForth for *msp-exp430fr5969* experimenter board
- noForth for MSP430G2553 *egel kit*

New words:

- `GIE> >GIE ROUTINE` (see noForth documentation) GIE-bit in status register is now saved and restored correctly around flashwriting.
- `(*` Nestable multi line comment sign. Both `(*` and the closing `*)` must be the first word on a line!
- Assembler words `BIX` and `BIA` (aliases for `XOR>` and `AND>`)
- `0=?` and `0<>?` (aliases for `=?` and `<>?`)
- Extended memory: `X@ X! XC@` and `XC!` for MSP-EXP430FR5969
- The word `NOFORTH` 'forgets' all added code so that only the kernel is left. This word is renamed to `NOFORTH\` .

And finally:

- The multiply functions use the hardware multiplier if present.
- The meta compiler files are cleaned up and (should) run now on 32 bit and on 64 bit standard forth systems.

http://home.hccnet.nl/anij/nof/noforth.html

[3] 2/2017: FRAM board MSP-EXP430FR5994 added, baud rate 115K2, includes protected memory.

What's in a name?[1]

Albert Nijhof

In Texten, die zum Ziel haben, Nicht-Forthlern Forth zu erklären, wird zuweilen triumphierend ein Beispiel im Stil von `: 1 ." Hallo! " ;` *präsentiert. So flexibel! Tatsache ist, dass wir hier auf ein fundamentales Fehlverhalten des Interpreters stoßen!*

Der simple Interpreter

Der Forth-Interpreter ist überaus einfach: Er liest im Inputstrom den nächsten Namen ...

Moment mal, der Ordnung halber: Ein „Name" ist in Forth eine Aneinanderreihung darstellbarer Zeichen (ASCII 0x21—0x7E), an beiden Enden begrenzt durch Zwischenräume, Zeilenbeginn oder Zeilenende. Außer dem Zwischenraum gibt es dabei (beinah) keine Zeichen, die einer speziellen Behandlung bedürfen. Also formulieren wir es nochmal von vorn.

Formulierung-1

- Der Interpreter liest den nächsten Namen aus dem Inputstrom und sucht ihn im Wörterbuch auf.
- Gefunden? - dann ausführen oder compilieren, abhängig von STATE und IMM[2] .
- Nicht gefunden? - dann versuchen, es als Zahl zu sehen. Wenn das nicht gelingt, dann stoppt der Prozess.

Das habe ich aber immer wie folgt aufgefasst.

Formulierung-2

- Der Interpreter liest den nächsten Namen aus dem Inputstrom, sucht ihn im Wörterbuch auf und führt ihn aus oder compiliert ihn, in Abhängigkeit von STATE und IMM .
- Ist der Name nicht zu finden, dann probiert der Interpreter "großzügig" erst noch, ob er daraus eine Zahl machen kann. Klappt auch das nicht, dann stoppt der Prozess.

Das läuft natürlich auf dasselbe hinaus. Der Unterschied liegt allein darin, dass die Formulierung-2 durchschimmern lässt, dass der Interpreter eigentlich keine Zahlen zu bearbeiten braucht, dass er das aber trotzdem tut — als benutzerfreundliches Verhalten gegenüber dem Programmierer.

Wirklich äußerst merkwürdig: Der Programmierer weiß, dass es um eine Zahl geht, und doch lässt er den Interpreter zuerst die Frage klären, ob es ein Name ist.

Was steckt hinter dem Namen?

Darin liegt natürlich das Dilemma: Daten werden als Namen behandelt. Der Interpreter kann keine Gedanken lesen!

Daten sollten im Forth-Code aber unzweideutig als Daten erkennbar sein.

Nun ist ausgerechnet der Auftrag an den Interpreter, Daten im Inputstrom zu erkennen und zu verarbeiten, überaus weitreichend. Es scheint die Hauptaufgabe des Interpreters zu werden (für kleine Systeme, wobei wir für den Moment mal die optimalisierenden Forths vergessen wollen). Es geht darum, Daten (Zahlen, doppeltgenaue Zahlen, Gleitkommazahlen, Strings, oder was sonst noch) an ihrem Erscheinungsbild über das Auftreten von speziellen Zeichen in den Daten zu erkennen, Zeichen wie in

```
#10 &10 %10 "a "b" &a 3.E2 'a C:
```

und was es noch alles gibt. Der Interpreter muss diese Zeichen erkennen können. Das ist das, was man als „Schweizer Taschenmesser" bezeichnet. Selbst spezielle Strukturen, *recognizers*, werden dafür konstruiert. Wenn diese Strukturen produktiv sind, d.h. vom Anwender erweiterbar, wird hier tatsächlich in den Namen hinterrücks doch noch eine Syntax hineingeschmuggelt — siehe die unten stehenden Visionen. Das ist sehr schade, vor allem für kleine Forthsysteme ist das unbrauchbar!

Es bedeutet auch, dass Namen, die durch ihr Erscheinungsbild eventuell als Daten aufgefasst werden können, Problemträchtig sind. Man ist also in der Wahl seiner Namen nicht frei. Namen wie `A. 2,` oder `#3` verwendet man in Forth besser nicht, und das ist eine zumindest *bemerkenswerte Einschränkung.*

Intermezzo (Visionen)

```
   Modernes Forth    Veraltetes Forth
   --------------    ----------------
   :AMSTERDAM        : amsterdam
   \This is comment  \ This is comment
und
   CON:LONDON        constant london
   VAR:PARIS         variable paris
Auch sehr praktisch:
   !PARIS            paris !
   @PARIS            paris @
und
```

[1] Juliet: "What's in a name? That which we call a rose By any other name would smell as sweet." — Shakespeare, Romeo and Juliet (II, ii, 1-2)

[2] STATE und IMMEDIATE sind Systemvariablen.

```
   VAL:BERLIN        value berlin
   TOBERLIN          to berlin
Außerordentlich praktisch:
   ,IF               postpone if
   ,THEN             postpone then
```

Man braucht sich natürlich nicht auf zweiteilige Zusammensetzungen zu beschränken. Die Möglichkeiten sind endlos!

```
   ON@STATE          state @ if
   @STATE=0          state @ 0=
   @PARIS<0          paris @ 0<
   ON@STATE=0        state @ 0= if
   ON@PARIS=200      paris @ 200 = if
```

Und um das nervtötende Postfixgetue noch etwas weiter einzudämmen:

```
   +LONDON           london +
   *LONDON           london *
   -LONDON           london -
   +@PARIS           paris @ +
   -@PARIS           paris @ -
   +245              245 +
   *245              245 *
   -245 ...
```

Na, sowas! Aber das lässt sich vielleicht wie folgt lösen:

```
   -245              -245
   --245             -245 -
   ... ?
```

Ein einfacherer Interpreter

Aber jetzt ernsthaft: Nehmen wir im Geist von Formulierung-2 mal streng forthfundamentalistisch an, dass der Interpreter auf gar keinen Fall für die Verarbeitung von Zahlen zuständig ist:

```
: INTERPRET
begin bl word find dup
while 0< state @ and
  if compile, else execute then
repeat
0= abort" Name not found " ;
```

Das würde ich immer noch einen einfachen Interpreter nennen! Aber was machen wir jetzt im Inputstrom mit Daten? Dafür gibt es eine sehr einfache, effektive und zugleich forth-artige Lösung: Die Datenmanager.

Datenmanager

Datenmanager sind Worte, die Daten im Inputstrom ankündigen, lesen und interpretieren. Das sind Spezialisten. Jeder Datentyp hat seinen eigenen Datenmanager. Auch der Datentyp braucht also nicht mehr herausgefunden zu werden. Wir machen uns zu allererst ein Word `N` , das beliebige Zahlen verarbeitet. Aufgepasst, zwischen Datenmanager und Daten dürfen einzig und allein Zwischenräume stehen. Beispiel:

```
HEX
: VISIBLE? ( x -- flag )
  n 21 n 7F within ;
```

Bevor man das jetzt beiseitelegt: Ich will hier nicht den Vorschlag machen, ein solches `N` in Forth einzuführen. Ich will nur einen Anstoß geben, im Geiste dieser Lösung über die vielen anderen Sorten von Daten nachzudenken. Müssen die denn allesamt durch das kleine „Großzügigkeitstor" des herkömmlichen Interpreters gehen?

Wenn ich ein Interpreter wäre, würde ich sagen: „Da passe ich. Ich gebe Ihnen den kleinen Finger und Sie nehmen gleich die ganze Hand! Einfache einzelne Zahlen will ich noch gern für Sie behandeln, aber darüberhinaus müssen Sie die Sache selbst in die Hand nehmen."

```
Beispiele für Datenmanager
--------------------------
N 100       \ einzelne Zahl
NN 100 10   \ 2 Zahlen
DN 100      \ doppeltgenaue Zahl
FL 100      \ Gleitkommazahl
HX 100      \ Hexzahl
DM 100      \ Dezimalzahl
BN 100      \ Binärzahl
CH C        \ ASCII-Code von C
CTRL C      \ Control-C als Zahl
XT C@       \ Token von C@
S "ccc"     \ Adresse und Länge des Strings
CS "ccc"    \ Adresse des Counted-Strings
```

S "red wine"

S -"green" wine-

CS ' Ciao! '

Abbildung 1: Datenmanager vor Zeichenketten

Jetzt noch schnell einen Abstecher zu den Strings[3]: Zieh den Delimiter von `S"` raus und setz ihn direkt vor den String. Das Ergebnis ist besser lesbar als das mit dem herkömmlichen `S"` , bei dem man ja den einen Zwischenraum[4] immer wegdenken muss. Überdies kann man jetzt seinen eigenen Delimiter wählen, beispielsweise für Strings, in welchen Anführungszeichen vorkommen:

```
S "red wine"
S -"green" wine-
CS ' Ciao! '
```

Damit deutlicher wird was gemeint ist, wurden in Abb.1 die Strings farblich unterlegt.

[3] Zeichenketten

[4] Den da: `S"␣hallo world"`

Summa summarum

1. Statt den Interpreter erst vergeblich Daten im Wörterbuch aufsuchen und im Gefolge davon untersuchen zu lassen, wie er damit umgehen soll, setzt der Programmierer einen Datenmanager vor die Daten. Der Datenmanager *weiß*, wie er die Daten zu behandeln hat.

2. Man braucht den Datentyp und die Tatsache, dass es sich um Daten handelt, nicht mehr an den Daten selbst sichtbar zu machen. Das *erhöht die Lesbarkeit*, sowohl für den Interpreter als auch für den Menschen — auf die Dauer wenigstens, denn `DN 100` oder `FL 100` wirkt natürlich abschreckend auf alle, die es anders gewohnt sind. Umgewöhnung will gewollt sein.

3. Der Datenmanager bildet mit den Daten ein Tandem. Das Tandem als Ganzes verhält sich state-smart, so wie das die Zahlen im gewöhnlichen Forth auch tun: Abhängig von `STATE` werden Daten auf den Stack gelegt oder compiliert. Der Datenmanager ist immediate und wird selbst niemals compiliert. Das `POSTPONE`n eines Datenmanagers geschieht auf eigenes Risiko. Siehe auch den Abschnitt *State-smart-Angst* weiter unten.

4. Die Datenmanager lassen sich gegebenenfalls auch gut als Bestandteil einer Anwendung auffassen, denn zu jedem Forth-System kann man *mit ganz gewöhnlichen Forthdefinitionen* für jeden denkbaren Datentyp eigene neue Datenmanager defininieren. Das ist vor allem für kleine Forth-Systeme praktisch. Die brauchen dann nämlich nicht schon auf alle möglichen Datentypen vorbereitet zu sein.

5. Die Definition eines „Namens" in Forth wird *uneingeschränkt* gültig: Eine Aneinanderreihung von darstellbaren Zeichen, von denen kein einziges einer speziellen Behandlung bedarf. Mit `N 1` ist das Manko von

```
: 1 ." Hallo! " ;
```

beseitigt, denn jetzt besteht ein Unterschied zwischen der Zahl 1 und dem Namen `1` .

State-smart-Angst, CH und XT

`[CHAR]` und `[']` wurden seinerzeit in den Standard aufgenommen. Datenmanager wären da praktischer gewesen, beispielsweise `CH *` und `XT DROP` .

- Verwende `CH` und `XT` , wenn die Daten unmittelbar folgen.
- Verwende `CHAR` und `'` , wenn die Daten nicht unmittelbar folgen.

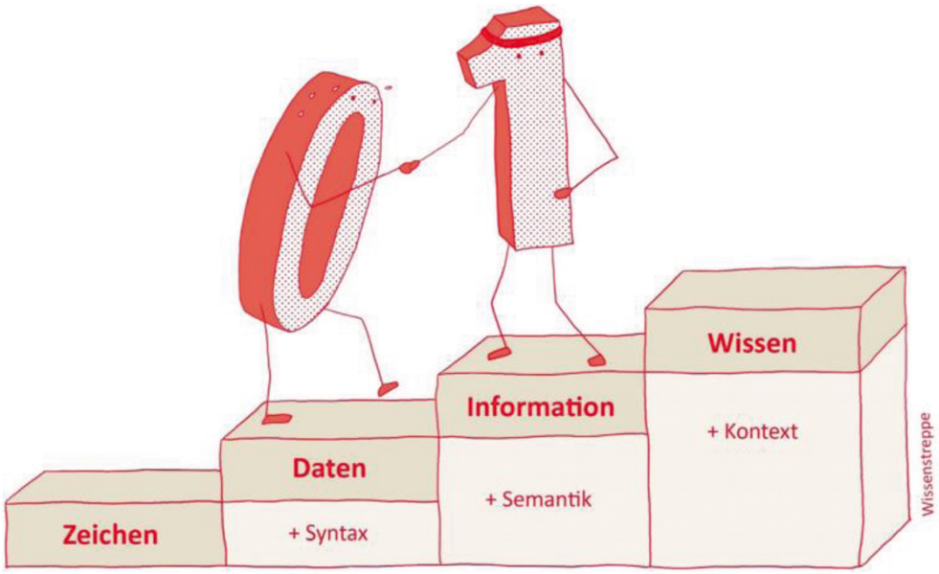
Diese Anwendungsvorschrift ist weniger esoterisch als die für `[CHAR]` und `[']` , zumal wir auch noch nicht über die grafische Unattraktivität jener Namen in den eckigen Klammern gesprochen haben. Die State–smart–Angst möchte ich relativieren: Natürlich kann man State–smart–Konstruktionen ersinnen — mit `FOO`, `BAR` und ausreichend vielen `POSTPONE`s oder `EVALUATE`s drin — um anzudeuten, dass dabei mitunter merkwürdige Dinge geschehen, aber niemand zwingt einen dazu, diese Sorte von Programmen zu machen.

Quellen und Links

Albert Nijhof hatte den Beitrag am 26.06.2016 in niederländischer Sprache veröffentlicht unter dem Titel „What's in a name? — Over datamanagers". Übersetzt ins Deutsche von Fred Behringer.

„Kleine Forth-Systeme" meint solche, die mit wenig Laufzeit–Code auskommen, typischerweise so um die 8KiB[5]. Das noForth von Albert Nijhof & Willem Ouwerkerk auf dem MSP430G2553 ist so eins, und ist dennoch komfortabel zu benutzen (Version vom 22 April 2016). Dort werden auch einige der Datenmanager verwendet.

http://home.hccnet.nl/anij/nof/noforth.html



[5] Ja. Acht. KiB = KiloByte !

µC–Pakete

Gerald Wodni

Softwarepakete auf dem PC sind längst eine Selbstverständlichkeit, doch ist Forth bei vielen Programmierern vor allem für die Anwendung in eingebetteten Systemen beliebt. Wie schnürt man nun ein Paket für µC, mit dem viele Forther eine Freude haben?

Stand der Technik

Einen Treiber in Forth zu programmieren, macht jede Menge Spaß, sobald das Basisprotokoll, wie I²C, SPI oder UART einmal steht, kann man den Bauteil interaktiv erforschen. Wenn der Bauteil sich bewährt, möchte man ihn vielleicht auch in Folgeprojekten einsetzen. Diesmal will man ihn aber nicht wieder komplett neu programmieren, sondern lieber den alten Treiber weiterverwenden.

Geht man nun einen Schritt zurück, um einen Blick auf alle Bauteile eines Projektes zu werfen, so stellt man schnell fest, wie viele davon digital sind und ein mehr oder weniger standardkonformes Protokoll sprechen.

Diese Art von Treiber sieht meist gleich aus:

1. µC–Protokollstack aufsetzen (Baudraten, Pinbelegungen)
2. Low–Level Wörter zum Senden und Empfangen
3. High–Level Wörter zur Nutzung der eigentlichen Funktionen
4. Initialisierungsroutine–Library
5. Initialisierungsroutine–µC

Plattformunabhängigkeit Das Aufsetzen des Protokollstacks (Punkt 1.) sowie die spezifische Initialisierung (Punkt 5.) muss für jeden µC aufs Neue geschrieben werden, was jedoch nicht unbedingt ein Nachteil ist, weil man so die von Forth gewohnte Freiheit der Konfigurierbarkeit hat. So kann man bei einem Chip, welcher keinen Hardware–I²C hat, einfach eine bit–banging–Variante verwenden.

Die Punkte 2.–4. sind im Gegensatz dazu sehr unabhängig und können wunderbar wiederverwendet werden.

Paketaufbau

Anhand des allgegenwärtigen HD44780–LCD–Treibers, der seit 1987 sein Unwesen treibt und bis heute ein Defacto–Standard für Dot–Matrix–LCDs ist, wird gezeigt wie daraus ein µC–unabhängiges Paket geschnürt wird.

Protokollstack

Dieser Punkt ist, wie besprochen, sehr davon abhängig, welcher µC eingesetzt wird und wie das Projekt generell aussieht. Deshalb dokumentiert das Paket die erwarteten Wörter, welche dann vom Benutzer kurzerhand implementiert werden.

`lcd-nibble ( x -{}- )` sends a nibble to the LCD (set D7 — D4 and strobe E)

`lcd-mode-data ( -{}- )` data mode (set RS)

`lcd-mode-cmd ( -{}- )` data mode (clear RS)

Der Vollständigkeit halber und um den geringen Aufwand zu zeigen, hier eine mögliche Implementierung vom Benutzer:

```
\ set bits at target address
: cbis! ( x c-addr -- )
    >r r@ c@ or r> c! ;

\ clear bits at target address
: cbic! ( x c-addr -- )
    >r invert r@ c@ and r> c! ;

\ Port the LCD is attached to
P2OUT constant LCD
\ select between command and data
$20 LCD 2constant RS
\ enable-pulse
$10 LCD 2constant ENA

: lcd-mode-data ( -- )
    RS cbis! ;

: lcd-mode-cmd ( -- )
    RS cbic! ;

: lcd-nibble ( x -- )
    \ clear lower nibble
    $F LCD cbic!
    \ set lower nibble to x
    $F and LCD cbis!
    1 us
    ENA cbic!
    4 us
    ENA cbis! ;
```

Forthprogrammierer lieben ihre Unabhängigkeit, während die obige Implementierung wie eine 08/15–Arduino–Library aussieht, hier noch eine alternative Implementierung, bei welcher die Datenleitungen über ein Schieberegister geschrieben werden:

```
0 variable display-data
\ serial data
$20 P1OUT 2constant SD
$10 P1OUT 2constant CLK
```

```
: 1clk ( 1 clock signal )
    CLK cbis! CLK cbic! ;

: >sr ( x -- )
    8 0 do
        dup $80 and if
            SD cbis!
        else
            SD cbic!
        then
        1 lshift
        1clk
    loop drop ;

: lcd-mode-data ( -- )
    $10 display-data cbis! ;

: lcd-mode-cmd ( -- )
    $10 display-data cbic! ;

: lcd-nibble ( x -- )
    $F P2OUT
    $F display-data cbic!
    $F and display-data cbis!
    display-data @ >sr
    1 us
    ENA cbic!
    4 us
    ENA cbis! ;
```

Durch die freie Implementierung der benötigten Wörter überlassen wir dem Benutzer also viele Freiheiten. Trotzdem sollte ein gutes Paket auch eine kleine Demo–Implementierung mitbringen, sodass der Benutzer schneller ans Ziel kommt.

Low–Level Interface

Sind die nötigsten Wörter durch den Benutzer definiert, kann er nun die Library laden, welche zuerst einige Basiswörter implementiert. Hier ein kurzer Auszug, der interessierte Leser findet die komplette Implementierung bei [2]:

```
\ send byte to display
: lcd-emit ( x -- )
    \ ." 8bit: " hex. cr ;
    \ higher nibble
    dup 4 rshift lcd-nibble
    \ lower  nibble
    $F and lcd-nibble ;

\ send command to display
: lcd-cmd ( x -- )
    lcd-mode-cmd
    lcd-emit
    lcd-mode-data ;

\ clear display content
: lcd-clear ( -- )
    $01 lcd-cmd ;
...
```

High–Level Interface

Steht die Basiskommunikation, ist es an der Zeit für etwas Luxus. Ohne den µC zuzuspammen, hier ein paar sinnvolle Erweiterungen:

```
\ send string to display
: lcd-type ( c-addr n -- )
    bounds do
        i c@ lcd-emit
    loop ;

\ like .( but send to lcd
: lcd( ( -- ) immediate
    [char] ) parse lcd-type ;

\ like ." but send to lcd
: lcd" ( -- ) immediate
    postpone s" postpone lcd-type ;
...
```

Initialisierung–Library

```
\ initialize 4 bit interface
: lcd-init ( f-lines -- )
    lcd-mode-cmd
    40 ms           \ powerup wait
    $03 lcd-nibble  \ function set
    4 ms
    $03 lcd-nibble
    1 ms
    $03 lcd-nibble
    1 ms
    $02 lcd-nibble

    \ 5x7, 4bits, f-lines
    -1 swap 0 lcd-function-set
    \ display on
    -1 0 0 lcd-display-control
    \ clear display
    lcd-clear
    \ increment on store, no shift
    -1 0 lcd-entry-mode ;
```

Initialisierung–µC

Zu guter Letzt muss der Benutzer noch seine Ports initialisieren und danach die Library.

```
: init-ports
    \ set outputs
    $20 P1OUT cbic!
    $23 P2OUT cbis!
    \ configure outputs
    $20 P1DIR cbis!
    $1F P2DIR cbis!
    \ initialize 2-lines
    2 lcd-init ;
```

Benutzeraufwand

Der Benutzer musste in unserem Beispiel ca. 20 kurze Zeilen programmieren, hat dabei aber stets die volle Kontrolle behalten, anstatt nur Ports und Pins zu spezifizieren und sich auf die Gnade der Library zu verlassen.

Das ist natürlich nicht bei jeder Art von Paket möglich. Bit–Banger werden sehr wohl direkt auf den Ports arbeiten.

Paketieren

Alles, was unserem Paket jetzt noch fehlt, ist eine schöne Dokumentation in Form einer (Markdown–)ReadMe, sowie eine package.4th–Beschreibungsdatei.

package.4th–Dateien machen ein Forth–Projekt zu einem Paket. Sie wurden vom Forth–200x–Komitee mitgestaltet und funktionieren wie folgt:

- Beginn des Paketes: `forth-package` sollte in der ersten Zeile stehen.
- `key-value <name> <value ...>` setzt einen Schlüssel `<name>` wobei der Rest der Zeile den Wert bildet.
- `key-list <name> <value ...>` Erstellt eine neue oder fügt einer bestehenden Liste `<name>` einen neuen Wert hinzu.
- Ende des Paketes: `end-forth-package` sollte in der letzten Zeile stehen.

Wobei folgende Schlüssel verpflichtend sind:

[name] Name des Paketes, folgt der Form {[a-z]+[-a-z0-9]*}

[version] Versionsnummer, bestehend aus 3 Zahlen, getrennt durch '.' (z.B. 1.23.42). Hierbei handelt es sich um eine semantische Versionierung, siehe dazu [5] und die Einschränkungen für Forth unter [4] .

[license] Kurzform der Lizenz des Paketes, z.B. "GPL", "GPLv3", "MIT", ...

Weitere optionale Schlüssel und Details finden sich unter [4]. Als konkretes Beispiel folgt die Paketbeschreibungsdatei des LCD-Treibers:

```
forth-package
    key-value name lcd-hd44780
    key-value version 0.1.0
    key-value description generic driver for HD44780 based Displays
    key-value license GPL
    key-list tags GPL
    key-list tags MCU
end-forth-package
```

Weiterentwicklungen

Es ist ein sehr einfaches UDP–Protokoll in Arbeit, welches auf Controllern mit Ethernet einen direkten Download bzw. ein direktes Interpretieren von Paketen erlaubt.

Auch ein µC ohne Ethernet profitiert von Paketen, indem man diese manuell als .zip oder mithilfe von f unter Gforth herunterlädt.

Terminals könnten ebenfalls die f–API unterstützen oder das vereinfachte UDP–Protokoll verwenden.

Zusammenfassung

Auch einfache Treiber wollen nicht auf jeder Plattform neu programmiert werden. Mit ein paar Handgriffen werden bestehende Treiber portabel. Fügt man noch eine package.4th–Datei hinzu, hat man ein ordentliches Paket, welches von anderen Forthern unkompliziert genutzt werden kann, und einem Ruhm und Ehre auf THEFORTH.NET garantiert ;-)

Bis zur nächsten Ausgabe, may the Forth Net be with you!

Demo

Abschließend noch ein Listing des Paketes im Einsatz auf einem MSP430G2553 mit Mecrisp [7]. Das `#include` wird in diesem Beispiel von e4thcom [6] aufgelöst.

```
\ set outputs
: init-ports
    $3F P2OUT c!
    $3F P2DIR c! ;

\ provide required words
: lcd-mode-data ( -- )
    $10 P2OUT cbis! ;

: lcd-mode-cmd ( -- )
    $10 P2OUT cbic! ;

: lcd-nibble ( x -- )
    \ clear data
    $F P2OUT cbic!
    \ set data
    $F and P2OUT cbis!
    1 us
    \ pulse E
    $20 P2OUT cbic!
    4 us
    $20 P2OUT cbis!
    \ break between nibbles
    43 us ;

\ include library
#include hd44780.4th

\ swap-dragon image (3 by 2 custom chars)
create d0
```

```
$00 c, $02 c, $03 c, $07 c,
$06 c, $00 c, $00 c, $0C c,
create d1
$00 c, $00 c, $00 c, $00 c,
$11 c, $11 c, $0A c, $1B c,
create d2
$00 c, $08 c, $18 c, $1C c,
$0C c, $00 c, $00 c, $0C c,
create d3
$1F c, $1D c, $18 c, $10 c,
$00 c, $00 c, $00 c, $00 c,
create d4
$1B c, $1F c, $1F c, $0E c,
$0A c, $0A c, $1B c, $00 c,
create d5
$1F c, $17 c, $03 c, $01 c,
$00 c, $00 c, $00 c, $00 c,

: demo
    \ set ports
    init-ports
    \ 2 line lcd
    2 lcd-init
    \ transmit dragon custom chars
    d0 0 lcd-char
    d1 1 lcd-char
    d2 2 lcd-char
    d3 3 lcd-char
    d4 4 lcd-char
    d5 5 lcd-char
    lcd" theforth.net   "
    0 lcd-emit 1 lcd-emit 2 lcd-emit
    $40 lcd-ddram \ 2nd line
    lcd" /lcd-hd44780   "
    3 lcd-emit 4 lcd-emit 5 lcd-emit ;
```

LCD	µC
D4	P2.0
D5	P2.1
D6	P2.2
D7	P2.3
RS	P2.4
E	P2.5
RW	GND

Tabelle 1: Verbindungen

Links

[1] https://theforth.net/

[2] http://theforth.net/projects/lcd-hd44780

[3] http://theforth.net/projects/f

[4] https://theforth.net/guidelines

[5] http://semver.org/

[6] https://wiki.forth-ev.de/doku.php/projects:e4thcom

[7] http://mecrisp.sourceforge.net/

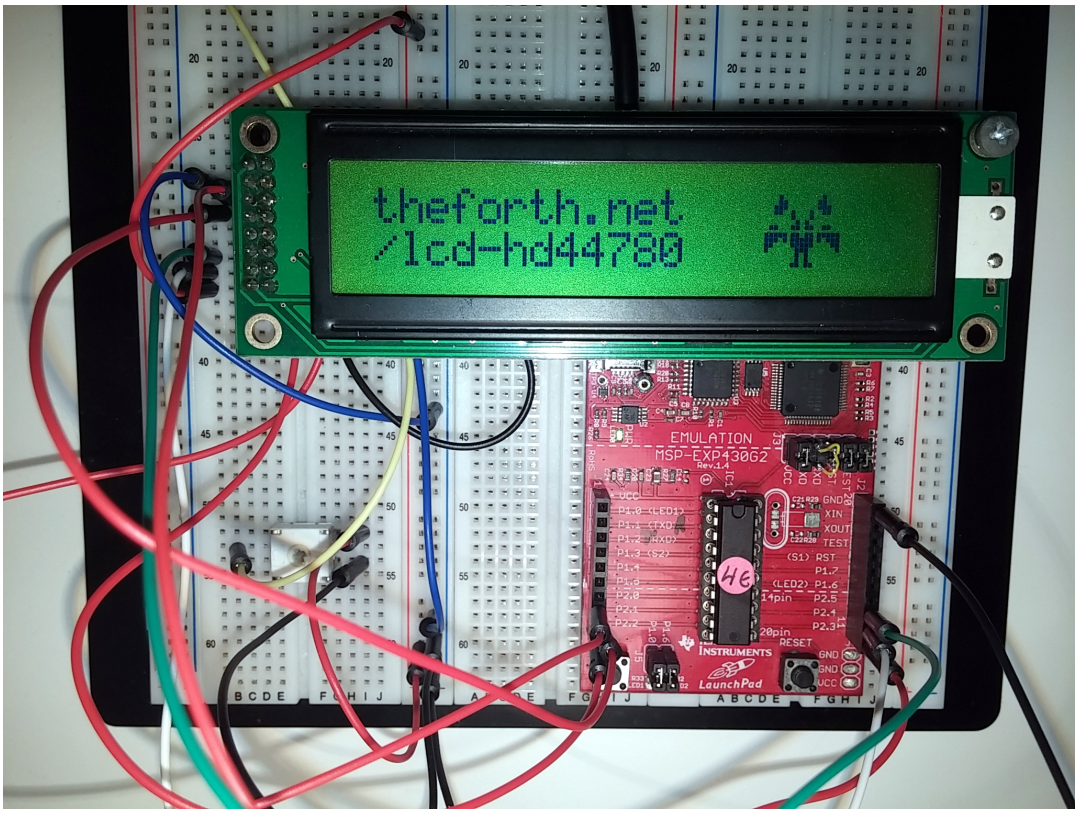


Abbildung 1: Das µC–Paket in Aktion

The Sockpuppet Forth to C interface

Stephen Pelc

As processors become ever more complex and the software we are asked to write becomes more complex, it takes ever longer to write the basic drivers for an embedded system. A full digital audio chain is vastly more complex than pumping DAC output into an audio amplifier. Silicon vendors provide C libraries to make using their chips easier. Rather than convert these libraries to Forth, MPE now provides a mechanism to call the C library from Forth.

Introduction

For microcomputers such as the Cortex cores and systems, manufacturers are providing software systems based on C libraries to make using their chips easier. Such libraries reduce the requirement for chip documentation at the expense of software documentation. This tendency has increased to the level that C header files include registers undocumented in the chip user manual. The manufacturers focus has changed from documentation to time-to-first-hello. The penalty is that chips take longer to learn and the documentation has more errors.

The conventional approach to providing support for development boards in Forth has been to manually port the C library sources to Forth. The SockPuppet system takes a different approach by providing an interface solution between Forth and C; the Forth system calls the underlying C libraries. In turn, this allows the details of the hardware to be abstracted away by the C libraries, whilst allowing the Forth system to provide a powerful, uniform and interactive user interface. This MPE code is directly inspired by Robert Sexton's Sockpuppet[1] interface. His contribution and permission are gratefully acknowledged[2].

Figure 1: Who am I?

Interfacing code from programming language to another is usually called *mixed language programming*. The MPE ARM/Cortex Forth cross-compiler supports Forth calling functions in C or any language that can provide functions that use the AAPCS calling convention. This is an ARM convention documented in IHI0042F aapcs.pdf. Calls with a variable number of parameters (varargs) are not supported.

The example code in both Forth and C is available for the Professional versions of the ARM Cortex cross compiler with automatic code generation of five interface types. The example code provides a simple GUI for an STM32F429I Discovery board using sample C code provided by ST and others. A version for the BBC micro:bit is in preparation. The interface is currently defined for Cortex-M CPUs only. All versions of the compiler can be used with hand-written assembler code.

How the Forth to C interface works

Both Flash and RAM memory are partitioned, one pair for C and the other for Forth. Because of the arcane and undocumented nature of start-up for C compiler target code, the initial boot of the system is performed by the C code in order to make sure that the initialisation is correct.

Every function that is exported from the C world to the Forth world appears as one of a number of types of call. These words are called *externs*. You can handcraft these words in assembler, but the MPE cross compiler compiler includes code generators for several techniques. The call format and return values match the AAPCS standard used by ARM C compilers.

Each calling technique has its own pros and cons. They are discussed in following sections.

- SVC calls. You just need to know the SVC numbers. SVC calls provide the greatest isolation between sections of code written in other languages. The functions foreign to Forth are accessed by SVC calls and/or jump tables. The example solution uses SVC calls for most foreign functions. Regardless of the call technique used by the majority of your code, all techniques rely on a small number of SVC calls.
- Jump table. The base address of the table can be set at run time, e.g. by making a specific SVC call. The calling words fetch the run-time address from the table, given an index. This technique has good performance and few problems.

[1] ... pretending to be another person.

[2] https://github.com/rbsexton/sockpuppet

- Double indirect call. A primary jump table is at a fixed address and contains the addresses of secondary tables, which hold the actual routine addresses. The fixed address and both indices must be known at compile time. This technique is used by TIs Stellaris and some NXP parts to access driver code in ROM.
- Direct calls to the address of the routine. You need to know the address at compile time.

There is a practical limit of four arguments if you use SVC calls for the insulation between Forth and C because Cortex CPUs automatically stack four registers for an interrupt. The other interface methods do not suffer from this limit. It is a matter of convention between the Forth and C code as to parameter passing order. It can be changed by either side. MPE convention is for the left-most Forth parameter to be passed in R0. This matches the AAPCS code used by the hosted Forth compilers such as VFX Forth for ARM Linux.

SVC calls

The examples use the MPE calling convention and are illustrated in assembler as well as by using the code generator. The code generator interface is much to be preferred and preserves far more of the information in the C prototype. The decision to use the C prototype is deliberate and follows long-established practice in MPEs hosted systems.

```
SVC( 67 )
void BSP_LCD_DrawCircle( int x, int y, int r );
\ SVC 67: draw a circle of radius r
\ at position (x,y).
```

The code generator parses the extern definition above and generates the extern as a function with three parameters implemented as SVC call 67. If you really want to demonstrate your assembler prowess, the code below performs the same operation.

```
CODE BSP_LCD_DrawCircle    \ x y r --
\ SVC 67 draw a circle
\ of radius r at position (x,y).
  mov r2, tos              \ r
  ldr r1, [ psp ], # 4     \ y
  ldr r0, [ psp ], # 4     \ x
  svc # __SAPI_BSP_LCD_DrawCircle
  ldr tos, [ psp ], # 4    \ restore TOS
next,
END-CODE
```

When the SVC call occurs, the Cortex CPU stacks registers R0-R3, R12, LR, PC, xPSR on the calling R13 stack with R0 at the lowest address. The SVC handler places the address of this frame in R0/R4, extracts the SVC call number, reloads the AAPCS parameters from the frame and jumps to the appropriate C function. In this case

```
void BSP_LCD_DrawCircle(
  uint16_t Xpos, uint16_t Ypos, uint16_t Radius
);
```

SVC calls provide the highest insulation between Forth and C, but suffer from several issues.

- The SVC call mechanism is part of the Cortex interrupt and exception system. The assembler and/or C side of this uses code written in assembler to allow the C routines called from a jump table to to be AAPCS compliant.
- The SVC mechanism is inefficient compared to a direct AAPCS handler.
- Because SVC calls are part of the CPU interrupt mechanism, you have to care how long a call takes. Playing games with the Cortex interrupt mechanism can fix this, but is complex.

Jump table

In order to avoid the run-time penalties of the SVC call mechnism, you can make an array of function pointers in C or assembler and call functions using an index into the table.

```
jumptable:
dd func0 ; address of function 0
dd func1 ; address of function 1
..
```

We still need to know the address of the jump table. This is found using an SVC call (15) and stored in a variable. The jump table address could be hard-coded, but given the horrors of perverting link map files and the like, the overhead of a single SVC call is preferable.

```
SVC( 15 ) void * GetDirFnTable( void );
\ Returns the address of the jump table.
variable JT \ -- addr
\ Holds the address of the jump table.
JT holdsJumpTable
\ Tell cross compiler
\ where jump table address is held.
: initJTI \ -- ; initialise jump table calls
  GetDirFnTable JT ! ;
JTI( n ) int open(
  const char * pathname, int flags, mode_t mode
);
```

If constructed in assembler, the SVC despatch table and the main jump table can be the same table; its just a question of what you put in the table.

Double indirect call tables

Some vendors, particularly TI, use a table of tables approach. The sub-tables provide the API for a particular peripheral, e.g. UARTs. Before use, you have to declare the base address of the primary ROM table used for calling ROM functions. For Luminary/TI CPUs, this will probably be:

```
$0100:0010 setPriTable
```

Now you can define a set of ROM calls, for example, again for a TI CPU.

```
DIC( 4, 0 ) void ROM_GPIOPinWrite(
  uint32 ui32Port, uint8 ui8Pins, uint8 ui8Val
);
```

where:

- ROM_APITABLE is an array of pointers located at 0x0100.0010.
- ROM_GPIOTABLE is an array of pointers located at ROM_APITABLE[4].
- ROM_GPIOPinWrite is a function pointer located at ROM_GPIOTABLE[0].

Parameters:

- ui32Port is the base address of the GPIO port.
- ui8Pins is the bit-packed representation of the pin(s).
- ui8Val is the value to write to the pin(s).

To call this function, use the Forth form:

```
port pins val ROM_GPIOPinWrite
```

Direct calls

Where the address of the routine is known at the Forth compile time, you can use a direct call.

```
DIR( addr ) int foo( int a, char *b, char c );
```

The Forth word marshalls the parameters and calls the subroutine at target address addr.

Extracting information from C

It is convenient to have a certain amount of information available from the C portion of the code. This is supported by a few SVC calls that exist in all versions of the Sockpuppet API.

```
svc( 0 ) int SAPI-Version( void );
SVC 00:
 Return the version of the API in use.

svc( 1 ) int GetSharedVars( void );
SVC 01:
 Get the address of the shared variable list.

svc( 15 ) int GetSvcFnTable( void );
SVC 15:
 Get the address of the SVC function table.
```

In order to support data sharing between C and Forth, the C can export named objects which can appear as Forth words.

C Linkage structure

```
#define DYNLINKNAMEMLEN 22

typedef struct {

// This union is a bit crazy,
// but its the simplest way of
// getting the compiler to shut up.

union {
void (*fp) (void);
int* ip;
unsigned int ui;
unsigned int* uip;
unsigned long* ulp;

// Pointer to the object of interest (4)
} p;

// Size in bytes (6)
int16_t size;

// How many (8)
int16_t count;

// Is this a variable or a constant? (9)
int8_t kind;

// Length of the string (10)
uint8_t strlen;

// Null-Term C string.
const char name[DYNLINKNAMEMLEN];

} runtimelink_t;
```

When the Forth system powers up it runs the Forth word `dy-populate` which uses SVC call 01 to get the address of the `dynamiclinks[]` table, and walks through the table creating Forth named variables whose addresses match those in the C system. A Forth word `dy-show` is provided to list the entries in the table.

Forth Linkage structure

```
interpreter
: hword 2 field ;
: byte 1 field ;
target
struct /runtimelink \ -- len
\ Forth equivalent of the C structure above.
  int fdy.val     \ usually a pointer 0, 4
  hword fdy.size  \ size in bytes 4, 2
  hword fdy.count \ how many 6, 2
  byte fdy.type   \ variable or constant 8, 1
  byte fdy.nlen   \ name length 9, 1
  22 field fdy.zname \ ...
  \ ... zero terminated name 10, 22
end-struct
```

The accessor words just read the fields defined above. They are defined as compiler macros. For interaction on the target, use the field names above.

```
compiler
: dy.val fdy.val @ ;       \ addr -- n
: dy.size fdy.size w@ ;    \ addr -- w
: dy.count fdy.count w@ ; \ addr -- w
: dy.type fdy.type c@ ;    \ addr -- c
: dy.name fdy.nlen ;       \ addr -- addr
target
```

A set of support words allow us to run down the table and create Forth `VALUE`s and `CONSTANT`s.

Demonstration code

In order to evaluate the Sockpuppet technique and to provide a demonstration environment we decided to port the MPE PowerView GUI code to an STM32F429I Discovery board, which includes a small QVGA colour panel.

Figure 2: STM32F429I Discovery board

We made a decision to standardise on the gcc compiler maintained by ARM, see link below. This seems to be a clean compiler, but it has a few deficiencies:

- It does not include a make utility.
- Every silicon vendor ships a different version of Eclipse with different make tools.
- They are all incompatible.

A good alternative for supported hardware is the online mbed compiler. Or just take the silicon vendors „free“ tools, accepting that we will need a huge amount of disc space (almost free these days) and a degree of pain in learning the tool-chain. The days when you could just download and go are long past. Whatever you do, there will be pain.

Conclusions

- Mixed language programming for embedded systems is entirely feasible and productive.
- Do not assume that the C libraries provided by the silicon vendors are bug-free.
- You can use the Forth to debug the C.
- Once you have set it up, it all works surprisingly well, but compared to Forth cross-compilation, the C compilation chain is baroque.
- Using the C libraries for hardware access saves a huge amount of time reading chip documentation. As the use of silicon vendor C libraries increases, silicon vendor are placing less importance on correct documentation. We have already found devices whose C libraries depend on undocumented registers.

Acknowledgements

Robert Sexton is responsible for the ideas and implementation of Sockpuppet. His dedication has made it a production-grade environment.

Elizabeth Rather convinced me long ago of the necessity of good notations. This convinced us to port the extern interface from hosted systems to the cross-compiler.

Links to additional information

As short and general information the Press Relaese (Pressmeldung und für alle verstaendlich in deutsch und englisch):

http://www.mpeforth.com/wp/wp-content/uploads/2015/12/MPE_adds_Interaction_to_C_2016_05_04_German.pdf

http://www.mpeforth.com/wp/wp-content/uploads/2015/12/MPE_Adds_Interaction_to_C_2016_05_04.pdf

A lot more details for the specialists at (Detaillierter für die Spezialisten):

http://www.mpeforth.com/wp/wp-content/uploads/2015/12/Mixed_Language_Programming_using_SockPuppet_v2.pdf

or as part of the Cortex M Manual from page 51 ff (oder als Teil des Cortex–M–Manuals, Seite 51ff):

http://www.mpeforth.com/wp/wp-content/uploads/2015/12/MPE_VFX_CortexCode_including_Mixed_Language_Section.pdf

Stephen Pelc, MicroProcessor Engineering, 133 Hill Lane, Southampton SO15 5AF, England; t: +44 (0)23 8063 1441, e: sfp@mpeforth.com

www.mpeforth.com

gcc compiler maintained by ARM: https://launchpad.net/gcc-arm-embedded

online mbed compiler: https://developer.mbed.org/

SSD1306 — A 128x64 Bit OLED Display

Willem Ouwerkerk

I wanted a small, cheap and easy to address display for a new robot I am designing as part of the Egel project.

The Robot

It has to run on MSP430 with a single cell LiPo battery, has two motors, some reflection sensors and a VL53L0X LIDAR mounted on a tiny servo to scan its surroundings. And a tiny graphic display was added to show the robots inner workings.

It's Tiny Display

The SSD1306 is a small and cheap OLED display of 0.96". It needs only 3.3 Volt an has a two wire I2C interface. It is addressable as 8 lines of 128 bytes each, each byte is a row of pixels of 8-bits vertically.

I2C

Because it has an I2C interface I can use one of the library's I made for the *Egel project*, published in chapter 37.

The file used is: `„e37um - basics for usci i2c.f“`

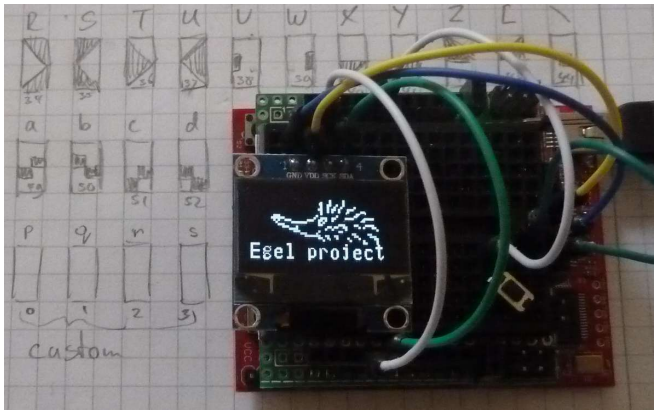


Figure 1: OLED Display on Egel Board

Forth to control the Display

The code published here does setup the display for use and adds the basic control routines like:

- `CONTRAST` - Set display brightness
- `ON/OFF` - Display on or off
- `INVERSE` - Black on white or white on black
- `XY` - Go to a given pixel row/column position
- `OL-SETUP` - Initialise OLED screen for use
- `OL-FILL` - Fill OLED screen with bit rows
- `OL-PAGE` - Erase OLED screen and to upperleft corner

Other additions

- 5x7 bit small character set
- 8x14 bit fat character set
- 8x14 bit thin character set
- 4x8 bit graphic character set
- Egel graphic start screen
- Graphic test screen for sensors mounted on the robot

Please visite the Egel Project Page or contact the author for more information.

Links

Egel Project : http://noforth.bitbucket.org/site/egel%20for%20launchpad.html

SSD1306 : https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf

Buy SSD1306 : https://www.amazon.de

Listing

```
1   (* I2C driver for SSD1306 0.96 inch 128x64 pixels oled screen using USCI I2C
2      routines. Separated files with a small and big character set. Extra fast
3      character output, a simple DO .. LOOP is to slow!
4   *)
5
6   hex
7   : {ol       ( b -- )        78 {i2write  i2out1 ;   \ Start an oled command: b=00 or old data: b=40
8                                                       \ Single byte command: b=80, single byte data: b=C0
9   : ol}       ( b -- )        i2out  i2stop} ;        \ End an oled stream
10  : CMD       ( b -- )        80 {ol  noop  ol} ;     \ Single byte oled command
```

```
11  : 2CMD      ( b1 b0 -- )    00 {ol  i2out  ol} ;    \ Dual byte oled command
12  : CONTRAST  ( b -- )        81 2cmd ;               \ b = 0 to 255 (max. contrast)
13  : ON/OFF    ( flag -- )     1 and  AE or cmd ;      \ Display on/off
14  : INVERSE   ( flag -- )     1 and  A6 or cmd ;      \ Display black or white
15
16  value x  value y
17  : XY        ( x y -- )          \ Set column and row
18      00 {ol                      \ Command stream
19      dup to y  7 and B0 or i2out \ Set page
20      dup to x  dup 0F and i2out  \ Set column
21      F0 and 4 rshift 10 or ol} ;
22
23  : OL-SETUP  ( -- )
24      00 {ol                  \ Start oled-command stream
25      0AE i2out               \ Display off
26      0A8 i2out  03F i2out    \ Set multiplexer ratio
27      0D3 i2out  000 i2out    \ Display offset = 0
28      040 i2out               \ Display starts at line 0
29      0A1 i2out               \ Mirror X-axis
30      0C8 i2out               \ Mirror Y-axis
31      0DA i2out  012 i2out    \ Alternate Com pin map
32      081 i2out  0C0 i2out    \ Set contrast to 75%
33      0A4 i2out               \ Enable rendering from GDRAM
34      0A6 i2out               \ oled in normal mode
35      0D5 i2out  080 i2out    \ Set oscillator clock
36      08D i2out  014 i2out    \ Charge pump on
37      0D9 i2out  022 i2out    \ Set precharge cycles to high cap.
38      0DB i2out  030 i2out    \ VCOMH voltage to max.
39      020 i2out  000 i2out    \ Horizontal display mode
40      0AF ol} ;               \ Display on, end stream
41
42  : OL-FILL       ( +n b -- ) \ Pattern 'b' to +n columns
43      40 {ol   swap           \ Start oled-data stream
44      begin                   \ Whole screen buffer
45          over i2out          \ Output pattern
46      1- ?dup 0= until  i2stop}  drop ;   \ End stream
47
48  : OL-ERASE      ( -- )          400 0 ol-fill ;     \ Erase screen
49  : OL-HOME       ( -- )          0 0 xy ;            \ To upper left corner
50  : OL-PAGE       ( -- )          ol-erase  ol-home ;
51
52  shield SSD1306\  freeze
53
54  \ End
```

Figure 2: Smal break-out-board SSD1306 128x64 Bit OLED Display

v4th

Vic Plichota

v4th is a Forth–like programming platform/framework for embedded microcontrollers. Much like Forth and LISP derivatives, it provides another „secret weapon“ in the savvy programmer's arsenal/toolkit/bag–of–tricks. Since its first version on the RCA 1802, v4th has been used to build many successful products, and ported to a wide variety of CPUs over the years, with continuous evolution and improvements. Today, versions exist for MSP430, MIPS, RX, and various flavors of ARM targets. v4th is not Forth — not quite. It only runs an „inner interpreter“ threading engine, and thus has no interactive REPL console, native compiler, nor its own IDE.

Then why use it?

High Performance. v4th's speed is in the same league as the very best fully–compiled Forths or C, along with a very compact code size footprint. Note that v4th was developed not for the mere ego–gratification of offering Yet Another Forth, but to fulfill a real need that classic Forths cannot.

Just like Forth, v4th uses a dual–stack Virtual Machine and nested definitions, and allows you to use the same design methodology and programming techniques as Forth. The author finds it much more productive and less annoying than C.

v4th is intended for embedded turnkey systems that are purpose–built machines, where user–programmability is undesirable and inappropriate. Nevertheless v4th is completely „open“; an experienced MCU jockey can easily modify the software at will.

In the hands of a savvy programmer, v4th can easily produce high–quality results similar to Forth Inc.'s SwiftX or MPEforth's VFX. Forth programmers (especially those with assembly–language experience) have no trouble grokking v4th's idioms. It is in fact an assembly language package that rides on top of the assembler, and thus provides all the convenient and productive features that modern developers expect.

This actually has some advantages:

(caveat: we are now into nerdy technicalities that will make the most sense to people who are familiar with Forth internals.)

Extremely compact footprints are possible; apps can be built within (for instance) 256 bytes of ROM and 16 bytes of RAM. There is no dictionary structure, so all v4th words are headerless.

Transparent access to the datasheet's standardized symbols; there is no need for a lot of 'constant' declarations; the assembler already knows the vendor's standardized labels (e.g. for the UART's baud–rate–divisor register), because the vendor has already pre–supplied the appropriate header files that contain the necessary `#define` and/or EQU directives. It's a nice thing, when your symbols agree with the manufacturer's documentation.

Inherits the assembler's IDE, with all those nifty features, and all the JTAG debug amenities that the assembler provides.

WYWIWYG debugging; there is no doubt about what machine code will execute, because What You Wrote Is What You Get. There is no compiler that makes any (perhaps unexpected) decisions on your behalf.

Very high performance; v4th is single–indirect (aka „direct–threaded“), and (depending on the target CPU) can often use a one-instruction NEXT. The v4th nucleus is practically guaranteed to remain resident in caches.

Highly optimized primitives; v4th provides state–of–the–art efficiency; needless stack–pumping is eliminated.

- Non-destructive variants that obviate 'dup' and 'over'.
- Reversed variants that obviate 'swap'.
- Fully conjugated conditionals, e.g. 'nif' instead of „not if“.
- Multi–way and table–based branches for state–machine designs.
- Branch target addresses are absolute, not relative offsets; thus no calculation cycles are required.
- ToS (and typically also NoS) are cached in CPU registers, instead of being held in RAM.
- High-level inline literals that use scratchpad registers, instead of the parameter stack. e.g. `addk, 5` instead of `5 +` and `strva, VALUE, ADDR` instead of `VALUE ADDR !` and wasteful push/pop thrashing is eliminated.

For example, Forth's :

```
Daddr @ PIXSIZE CHARWIDTH * - Daddr !
```

is

```
DW  pstrkk,-(PIXSIZE * CHARWIDTH),Daddr
```

in v4th's idiom, and (of course) runs much faster.

Low-level inline macros that eliminate nesting; many v4th words have both inline and „worded“ cognates, e.g. for ARM (note the upper–/lower–case difference):

```
NEXT MACRO
ldr PC, [i], #4
ENDM
```

```
DUP MACRO
str n, [p, #-4]!
mov n, t
ENDM

dup DUP
NEXT
```

Here is v4th's NEXT for some other targets:

```
MSP430:
mov @i+, PC

RX:
RTS  ; the UserStackPointer is hijacked,
     ; and used as the Interpreter Pointer.
```

RX again, with an alternate implementation for the meek and timid:

```
mov.l [i+], w
jmp w

MIPS:
lw w, 0(i)
jr w
addi i, i, #4 ; executed in branch delay-slot.
```

Utterly flexible; because v4th words are all in assembly language, you can quite seamlessly switch to/from writing in machine code and high–level, on the fly. This point deserves a bit of amplification: Forth is not a perfect language, and some jobs (e.g. DSP or graphics) can become awkward and klunky. v4th allows you to completely bypass the Forth Virtual Machine, and take full advantage of the CPU's general–purpose register set and complete instruction repertoire.

Lastly, **v4th is interoperable with C;** you can take advantage of existing middleware such as USB drivers or TCP/IP protocols (like it or not, they're probably implemented in C), without re–inventing the wheel.

Demo

As a comparative example, there are three ways of flashing an LED given in the listing below. The LED blinks quickly at first, then slows down.

And of course, for those instances where normal v4th may not be the most appropriate means of solving the problem, you can still write a v4th word as fully–handcrafted assembly code, and that word is added to the vocabulary like any other.

Acknowledgement

Finally, I must give my sincere thanks and acknowledgement to all the fine Forth folks who have helped me with v4th's evolution and improvement over the years. There are too many people to list here, but I hope that they all know who they are ...

I'm somewhat clever, but really it has been other people's feedback and input to v4th that makes me look more brilliant. :-)

Two people deserve special mention:

1. My brother Myron Plichota; he is brilliant.
2. I feel truly honored that Cyde W. Philips Jr. (another brilliant fellow) has seen fit to incorporate some v4th constructs into his recent *FISH forth*. [feb 2017]

Contact

Vic Plichota, embedded systems architect
vic.plichota (@"at") gmail ("dot") com
vic (@"at") MachineQuiltingRobot ("dot") com
http://MachineQuiltingRobot.com/
7 Centre St., P.O.B. 91, Port Rowan, ON,
Canada, N0E 1M0

Listing

```
<pre>
<code>

in Forth:
---------

literal1 constant LEDPORT
literal2 constant LEDBIT

: toggleLED     \ read/modify/write, toggle LEDBIT only
LEDPORT @       \ get LED status
dup not         \ toggle status
LEDBIT and      \ isolate bit
swap LEDBIT not and     \ clear LED bit
or LEDPORT !    \ merge new status, and update port
;

\ : toggleLED
\ LEDPORT @ dup not LEDBIT and swap LEDBIT
\ not and or LEDPORT ! ;
\ this two-line version is bad style,
\ but it's fun to say the words out loud...   :-)

: LEDflash
0
begin
  1 + dup       \ increment delay
  begin
    1 - dup 0=
  until
  drop toggleLED
again
;


in v4th:
--------

LEDPORT EQU     AsmSymbolForPortReg
LEDBIT  EQU     AsmSymbolForPortPin

; 'NEST' is v4th's equivalent of 'DOCOL'.
 NEST   toggleLED
  DW    atk, LEDPORT, nott, rmwam, LEDPORT, LEDBIT
  ; read/modify/write using inline address and mask.
  DW nexit ; 'nexit' is v4th's equivalent of 'SEMIS'.
```

```
 NEST   LEDflash
  DW zero
  DW begin
  DW  inc, dup  ; increment delay
  DW  begin
  DW   dec
  DW   zuntiln  ; v4th's equivalent of "dup 0= until"
  DW   drop, toggleLED
  DW  again
; infinite loop, so 'nexit' is not required


using v4th inline machine-code macros:
---------------------------------------

LEDflash
    ZERO
_flashloop
        INC     ;  increment delay
        DUP
_delayloop
        DEC
        bnz     _delayloop

        DROP
        ATK     LEDPORT
        NOTT
        RMWAM   LEDPORT, LEDBIT         ; read/modify/write,
                                        ; toggle LEDBIT only
        b       _flashloop

</code>
</pre>
```

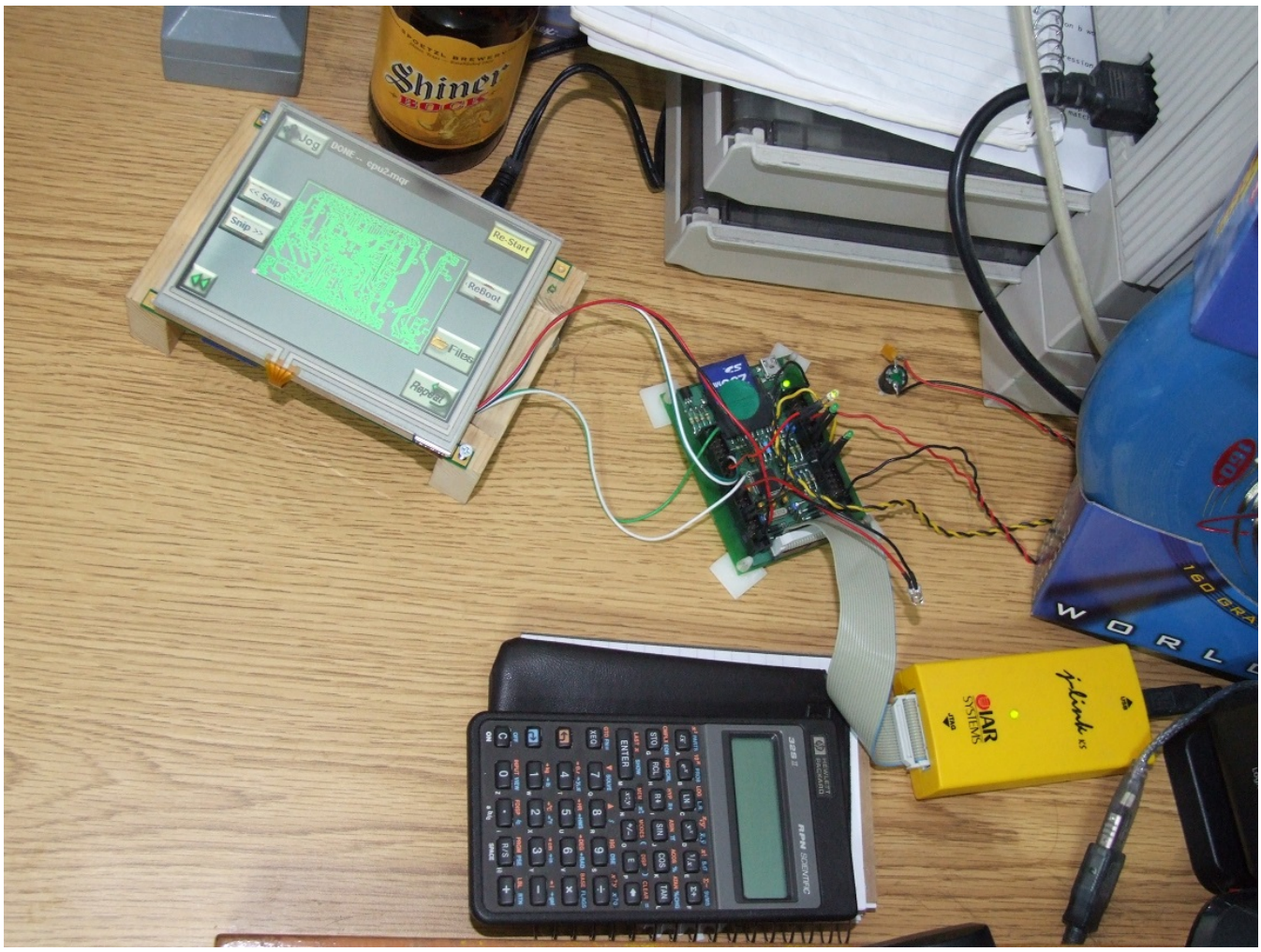


Figure 1: The initial bench-testing of my NXP LPC2148-based CPU PCB in 2007, which is the brain for my MQR MachineQuiltingRobot commercial product.

ROM und RAM in einem ROMforth

Albert Nijhof

Ein wirklich simples und dennoch ausgefuchstes Forth für so kleine CPUs zu haben, wie dem MSP430G2553, ist von Anfang an das Ziel von noForth *gewesen. Der Forth-Standard ist für solche Maschinchen nicht immer passend und für manche Fragen bei der Implementation der Sprache liefert der Standard keine Antworten. Es kursieren voneinander abweichende Lösungen dafür bei den jeweiligen Herausgebern von Forth. Der Autor verficht hier den noForth-Weg (mk).*

Die Fragen

- **Zwei Mal HERE?**
  Für ein Forth, das ins (Flash)ROM compiliert, geht man davon aus, dass der Programmcode und die unveränderlichen Daten ins ROM gehen, wohingegen die veränderbaren Daten ins RAM gehören. Benötigt man dann also auch zwei `HERE`s?
- **Befehlssatz verdoppeln?**
  Ins ROM schreiben ist technisch ein anderer Vorgang als ins RAM schreiben. Benötigen wir also einen kompletten doppelten Satz von Befehlen für Worte, die etwas in den Speicher setzen oder dort verändern?
- **CFA im RAM?**
  `DOES>` überschreibt die CFA eines Wortes, was aber im ROM nicht möglich ist. Müssen CFAs daher im RAM liegen?
- **Body im RAM oder im ROM?**
  `CREATE TEST`
  Steht da nun auf dem Stack nach Ausführung von `TEST` eine RAM- oder eine ROM-Adresse?

Da der Forth-Standard diese Dinge nicht regelt, wird ein Forth, das ins ROM compiliert, immer vom Standard abweichen!

Die Antworten

Für das ROM und das RAM braucht es die Funktion der Worte aus Tabelle 1. Wie kann man diese Funktionen implementieren?

ROM	*RAM*
ROMHERE	RAMHERE
ROM! ROMC!	RAM! RAMC!
ROMMOVE	RAMMOVE
ROM, ROMC,	RAM, RAMC,
ROMALIGN	RAMALIGN
ROMALLOT	RAMALLOT
ROMCREATE	RAMCREATE

Tabelle 1: Theoretische ROM– und RAM–Worte

Lösung 1: Worte, die mit einer Zieladresse auf den Stack geschickt werden, sollten *address-smart* sein. Forth sucht dann selbst heraus, ob es ROM oder RAM sein soll. Das funktioniert jedoch nicht mit Worten, die rund um `HERE` arbeiten. Da die Zieladresse von beispielsweise `C,` nur implizit gegeben ist, muss der Programmierer auf die eine oder andere Weise anzeigen, ob es hier ROMHERE oder RAMHERE heißen muss.

Lösung 2: Man verwende zwei lose Worte, ROM und RAM, die zwischen RAM-Modus und ROM-Modus umschalten. Den Haken dabei, und was dann noch alles zu berücksichtigen wäre, lasse ich auf sich beruhen; denn es gibt eine einfachere Lösung: Unsere.

Die noForth-Lösung

Für noForth haben wir uns eine einfachere Lösung ausgedacht. Bei näherer Betrachtung erkennt man, dass die Liste der doppelten Befehle viel kleiner ausfallen kann: Wir kommen allein mit den Standard-Forthworten, ohne die Präfixe ROM oder RAM, aus, wenn wir nur zwei neue Worte verwenden, nämlich `CHERE` und `M,` . Das sind ja zwei `HERE`s? Ja, wir haben *zwei* `HERE`s nötig. `HERE` ist der Data-Space-Pointer im RAM und `CHERE` ist der Dictionary-Pointer im ROM, also das *code-HERE*. Man wird `CHERE` allerdings selten explizit verwenden. Und als ROM-Worte werden dann nur die Kommaworte `, C, M,` und `ALIGN` gebraucht, die ab `CHERE` compilieren. Und mit `M,` wird bei `CHERE` ein String ans Dictionary angefügt. Er hat die Funktion eines `MOVE` ins ROM:

```
: M, ( adr len -- )
  0 ?DO COUNT C, LOOP DROP ;
```

Detail: In noForth schaut ein Wort, das ein Zeichen x ins ROM schreiben soll, erst nach, ob x dort schon steht. Wenn ja, dann wird der Schreibvorgang unterlassen. Damit wird überflüssiges Schreiben ins ROM vermieden.

Warum werden fürs RAM keine Kommaworte und kein ALIGN benötigt?

Angenommen, man habe ein Programm in noForth geladen und es mit `FREEZE` eingefroren. Wenn das Programm während des Compilierens RAM reserviert hat und wenn es das RAM gleichzeitig mit Daten gefüllt hat — das ist das, was `RAM,` und `RAMC,` tun — dann geht der Inhalt des reservierten RAMs verloren, sobald der Chip ausgeschaltet wird. Wie löst man dieses Problem?

Dadurch, dass man das RAM während des Compilierens mit Hilfe von `ALLOT` nur reserviert, aber nicht füllt. Der Inhalt dieses RAMs ist dann unbestimmt. Später, beim Ausführen, setzt man dann mit den gewohnten Store-Worten die Anfangswerte in das reservierte RAM ein.

Aus diesem Grund ist in noForth beim Definieren eines Values keine Zahl auf dem Stack nötig. Der Inhalt eines

gerade definierten Values ist unbestimmt. (Vielleicht wäre `VAL` *ein besserer Name gewesen?)*

Bei `ALLOT` gibt man explizit an, wieviel Platz man reservieren möchte. Man kann selbst dafür sorgen, dass das immer ein geradzahliges Vielfaches ist (abhängig vom Prozessor). Das macht ein `ALIGN` fürs RAM überflüssig.

Merke: `ALLOT` reserviert Platz von `HERE` an im RAM. Der kann dann später durch `! C!` und `MOVE` ausgefüllt werden.

Auch andere Nicht-Kommaworte, die etwas im Speicher verändern, wie `+! TO *BIS *BIC *BIX UPPER MOVE FILL` usw. funktionieren nur im RAM.

Warum sind `! C!` und `MOVE` fürs ROM nicht nötig?

Das Schreiben ins unbeschriebene ROM geschieht immer inkrementell über `CHERE` mit den Kommaworten (ausgenommen: siehe das gleich folgende "Patchen im ROM"). Worte, die im Speicher etwas verändern, wie beispielsweise `+!` , sind fürs ROM unbrauchbar. Von den RAM/ROM-Worten aus Tabelle 1 übrig bleiben also nur noch die Worte, die in Tabelle 2 wiedergegeben sind.

ROM	*RAM*
CHERE	HERE
	! C!
M,	MOVE
, C,	
ALIGN	
	ALLOT

Tabelle 2: ROM und RAM-Worte im noForth

Patchen im ROM

Wenn man, in einem absoluten Ausnahmefall, jemals ein `ROMALLOT` benötigen sollte, z.B., um eine Tabelle im ROM anzufertigen, die man aus irgendeinem Grund erst später ausfüllen zu können glaubt, dann kann man das bewältigen z.B. über

```
CREATE LISTE   CHERE 8 M,
```

Das schafft 8 Bytes Platz bei `CHERE`, aber es wird nichts ins ROM geschrieben. Um die Tabelle aufzufüllen, würde man `ROM! ROMC` oder `ROMMOVE` benötigen. Die gibt es in noForth auch, sie sitzen in den Kommaworten, aber als Programmierer wird man sie selten verwenden.

DOES>

In noForth bewältigt `CREATE` alle folgenden Situationen:

- `CREATE` im ROM ohne `DOES>`
- `CREATE` im ROM mit `DOES>`
- `CREATE` mit `ALLOT` im RAM ohne `DOES>` (Allot-Bit = 0)
- `CREATE` mit `ALLOT` im RAM und `DOES>` (Allot-Bit = 0)

Der Trick dabei ist: `CREATE` setzt *nichts* in die CFA des neuen Wortes, weil da noch ein `DOES>` kommen kann. Doch wie gelangt dann die richtige Adresse in die CFA bei einem `CREATE` ohne `DOES>` ? Wenn man ein Wort ausführen will, muss man es zuerst einmal finden. `FIND` überprüft die CFA eines jeden gefundenen Wortes. Ist es leer? Dann wird es nachträglich mit `doROMbody` oder `doRAMbody` gefüllt, *passend zum Allot-Bit* (siehe weiter unten).

```
/ * CREATE im ROM ohne DOES>
create HI ( -- ROMadr )
S" Hello! " dup c, m,  align
hi count type <enter> Hello!

/ * CREATE im ROM mit DOES>
: CONSTANT ( x 'name' -- ) create , does> @ ;
12 constant DUTZEND
dutzend . <enter> 12
```

CREATE

`n ALLOT` reserviert n Bytes von `HERE` an im RAM. Wenn nach der CFA des neuen Wortes noch nichts compiliert wurde, *dann und nur dann,* passiert der Trick: `ALLOT` setzt einen Zeiger (`HERE ,`) auf das reservierte RAM in den `ROMbody` und setzt danach das Allot-Bit auf null.

In noForth hat jedes Wort nach einem Immediate-Bit auch ein Allot-Bit. So wie `IMMEDIATE` das Immediate-Bit auf den Wert Null setzt, tut `ALLOT` das mit dem Allot-Bit.

Um zu vermeiden, dass Allot-Bit und Immediate-Bit zusammen in einem Byte stehen, verwendet noForth das unterste Bit in der CFA als Allot-Bit. Dieses Bit wird irgendwann einmal null werden, da die CFA stets aus einer geradzahligen Adresse besteht.[1] Eine leere CFA wird in Übereinstimmung mit dem Allot-Bit durch `FIND` gefüllt. Danach spielt das Allot-Bit keine Rolle mehr.

```
/ CREATE mit ALLOT ohne DOES> (Allot-Bit = 0)
: STRING ( #bytes 'name' -- )
  create allot ;
8 string HI ( -- RAMadr )

: PLACE ( adr len dest -- )
  2dup c! 1+ swap move ;
S" Ciao! " hi place
hi count type <enter> Ciao!

S" Ola! " hi place
hi count type <enter> Ola!

/ CREATE mit ALLOT und DOES> (Allot-Bit = 0)
: VVALUE ( 'name' -- )
  create 2 allot does> @ @ ;
vvalue TRY
```

[1] Technisch kann ein einzelnes Bit im ROM auf Null gesetzt werden, wohingegen das Umgekehrte nicht möglich ist, ohne einen ganzen Block zu löschen.

Tja, wie kann man das TRY nun füllen?

```
: TTO ( 'name' -- )
  ' >body @ state @
  if postpone literal postpone ! exit
  then ! ; immediate
9 tto try   try . <enter> 9
```

Beim echten Value wird das sicherer über TO gelöst, aber es führt zu weit, das hier zu beschreiben.

Für die Tüftler

Eine der folgenden Definitionen ist falsch. Welche? Warum?

```
: VARIABLE1
  here 2 allot create , does> @ ;

: VARIABLE2
  create here , 2 allot does> @ ;

: VARIABLE3
  create 2 allot here , does> @ ;

: VARIABLE4
  create 2 allot does> @ ;

: VARIABLE5
  here 2 allot constant ;

: VARIABLE6
  here constant 2 allot ;

: VARIABLE7
  create 2 allot ;
```

Links

http://home.hccnet.nl/anij/nof/noforth.html

http://forth-standard.org/

Abbildung 1: Der Tüftler

Vintage Computing - FORPS

Jostein Skjelstad

Nur so, zum Vergnügen, hab ich FORPS in Gforth gefasst. In der Ausgabe „The Journal of Forth Application and Research“ von 1986 waren Expertensysteme Thema. In einem Expertensystem ist das Wissen und die Problemlösung nicht in der Software gemischt, sondern die Bereiche sind vollständig getrennt in zwei Module: Aufstellung von Regeln und die inference machine.

Die alte Ausgabe beschreibt das *FORth–based Production System* (FORPS). Es ist ein Beispiel für ein Expertensystem, das sowohl den Forth–Code als auch eine funktionierende Demo enthält, die Lösung des mathematischen Knobel- und Geduldspieles „Die Türme von Hanoi“[1].

Der Quellcode wurde im Jahre 1985 für Polyforth geschrieben und funktioniert nicht mehr mit modernen Forth–Kompilatoren. Deshalb habe ich den Quellcode umgearbeitet, so dass er jetzt wieder funktioniert mit gforth, VFX, bigforth und Swiftforth. Auch Euch viel Vergnügen damit[2].

Quellen

The Journal of FORTH Application and Research, 1986, Volume 4, Number 1, Page 7: „The Internals of FORPS: A FORth–based Production System“, by Christopher J. Matheus

http://soton.mpeforth.com/flag/jfar/vol4/no1/article1.pdf

Christopher J. Matheus: FORPS: A Forth–Based Production System and its Application to a Real–Time Robot Control Problem.

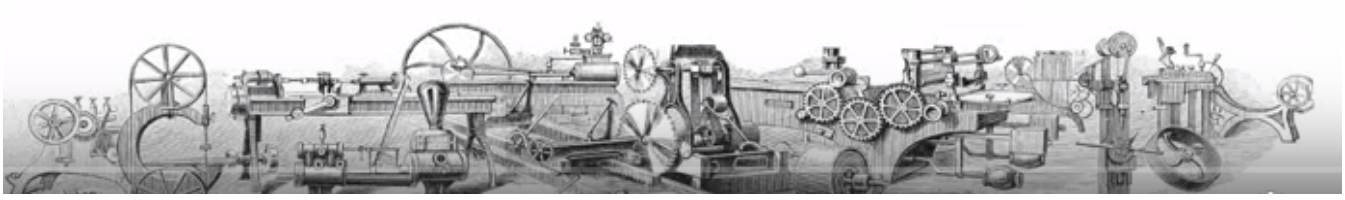
https://www.osti.gov/scitech/biblio/5816571

29Dec1986 Greg Guerin: Public domain version of FORPS for MacFORTH Plus

https://de.wikipedia.org/wiki/Türme_von_Hanoi

Jostein Skjelstad N-4230 Sand, Norwegen;

jskjelstad@me.com



Listing

```
\ FORPS: A Forth-based Production System
\ A "production" is
\ a rule (antecedent)
\ and action (consequent)
\ Rules are defined to produce a system.

\ Heavily modified from the public-domain source
\ presented in:
\   The Journal of FORTH Application and Research 1986
\   Volume 4  Number 1  Page 7
\     "The Internals of FORPS:
\     A FORth-based Production System"
\     by Christopher J. Matheus
\ http://soton.mpeforth.com/
          flag/jfar/vol4/no1/article1.pdf
\ See this article for explanation and use.
\
\ The original source code for FORPS presented in the
\ above article has the following restriction:
\ "FORPS is under the copyright of Martin Marrieta
\ Energy Systems. It is placed in the public domain
\ and may be used freely, as long as no false claims
\ are made to its authorship."

\ Since this code was derived from the original FORPS,
\ it is under a similar restriction.

\ This version for GFORTH
\ by Jostein Skjelstad 2017.01.21
\    Also works under VFX, Bigforth and Swiftforth
\    Implicit ANDing omitted
\    Original names in CAPITALs

\ monotype array, from FSL library --------------------

1 cells constant integer   \ size of a regular integer
1 cells constant flag      \ size of a flag
                           \ (for readability)
1 cells constant pointer   \ size of a pointer or xt

: marray ( n cell_size -- | -- addr )
    create
      dup , * allot
    does> cell+
;

\ word that fetches 1-D array addresses:
```

[1] Vermutlich wurde das Spiel 1883 vom französischen Mathematiker Édouard Lucas erfunden – deshalb auch manchmal Lucas-Türme (engl. Lucas Tower) genannt. Er dachte sich dazu die Geschichte aus, dass indische Mönche im großen Tempel zu Benares, im Mittelpunkt der Welt, einen Turm aus 64 goldenen Scheiben versetzen müssten, und wenn ihnen das gelungen sei, wäre das Ende der Welt gekommen. (Quelle: Wikipedia)

[2] Herzlichen Dank, Jostein! Mehr davon ... :-)

```
: } ( addr n -- addr[n])
          over [ 1 cells ] literal -  @ * +
;

\ Rule table ----------------------------------------

\ maximum number of rules allowed:
10 constant MAX-#RULES

MAX-#RULES pointer marray cond{
MAX-#RULES pointer marray action{
MAX-#RULES flag    marray fire{
MAX-#RULES integer marray priority{

\ number of current rule:
variable ThisRule  0 ThisRule !

\ number+1 of last rule in table!! :
variable LastRule

: *ERROR*  1 abort" No rules loaded" ;

: *RESET-FORPS*   ( -- )
    MAX-#RULES 0   do
       0 priority{ i } !  loop  \ erase  array
    \ ready for first rule: rule number 0
    0 ThisRule !
    ['] *ERROR*   cond{ 0 } !
;

\ puts the rule-tables limits on the stack
: RT-LIMITS   ( -- n 0 )   LastRule @  0 ;

\ Rule defining words -------------------------------

: action-xt! ( xt -- )
    action{ ThisRule @ } !   1 ThisRule +! ;
: cond-xt!   ( xt -- )
    cond{ ThisRule @ }  !   ;

\ A rule consists of a row in rule table
\ and 3 compiled words:
\ First word is the name only, compiled by RULE:
\ Second word is :noname definition with conditions
\ Third word is :noname definition with actions

\ Rule name compiles into dictionary,
\ but not into rule table.
\ Not used by the inference machine,
\ for readability only.

: RULE:   ( "name" -- )
    ThisRule @   MAX-#RULES = abort" no room"
    : postpone ;         \ begins and ends first word
;

: PRIORITY:   ( word  ( -- )
\ compiles nothing into dictionary,
\ storing a number only.
\ word must be a number
\ or a single word leaving a number.
    0. bl parse
     >number    dup  ( d  addr u u )
     if
        \ leaving a number on stack:
        rot drop rot drop evaluate
     else
        2drop d>s
     then
     priority{ ThisRule @ } !
;

: *IF*    :noname ;     \ begins second word

: *THEN*
    postpone ;          \ ends second word
    cond-xt!            \ save execution token
    :noname             \ begins third word
;  immediate

: *END*
    postpone ;          \ ends third word
    action-xt!          \ save execution token.
                        \ incrms rule counter
;  immediate

\ For debugging ------------------------------------

: .Rules   ( -- )       \ print ruletable
    ThisRule @   LastRule !
    RT-LIMITS   do
      cr i .
      cond{ i } ?
      action{ i } ?
      fire{ i } ?
      priority{ i } ?
    loop
;
\ Inference engine ---------------------------------

\ xt of highest priority active rule
variable BEST-ACTIVE-RULE

\ Highest Priority off all rules fired
variable HIGH-PRI

\ number of cycles executed
variable CYCLE
variable 'NOOP

\ used as default BEST-ACTIVE-RULE
' noop 'NOOP !

\ true if no rules fired during the cycle
variable NO-ACTIVITY

: BEST  ( -- )
    RT-LIMITS   do
       fire{ i } @  if
        priority{ i } @ DUP HIGH-PRI @   >   if
            HIGH-PRI !
            action{ i } @
            BEST-ACTIVE-RULE !
            FALSE NO-ACTIVITY !
        else
            drop
        then
    then
    loop
;

: SET-DEFAULT   ( -- )
    -1 HIGH-PRI !  'NOOP @  BEST-ACTIVE-RULE ! ;

: TEST-RULE-CONDS    ( -- )
    RT-LIMITS   do
       cond{ i } @ execute  fire{ i } ! loop ;

: SELECT-BEST-RULE  ( -- )
    TRUE NO-ACTIVITY !
    SET-DEFAULT
    BEST
;

: CLEAR-FIRES   ( -- )
    RT-LIMITS   do   0 fire{ I } !   loop ;

: FIRE-RULE    ( -- )
    BEST-ACTIVE-RULE @   execute ;

: FORPS   ( -- )
    0 CYCLE !
```

```
    ThisRule @   LastRule !
    begin
        1 CYCLE +!
        CLEAR-FIRES
        TEST-RULE-CONDS
        SELECT-BEST-RULE
        FIRE-RULE
        NO-ACTIVITY @
    until
;
\ Towers of Hanoi in FORPS ---------------------------
\ The 1986 PolyForth version code, unmodified

10 CONSTANT MAX-#DISKS  \ maximum number of disks
1 CONSTANT HOME         \ the original home peg
2 CONSTANT GOAL         \ the original goal peg
1 CONSTANT MOVE-TOWER   \ the code for a tower move
2 CONSTANT MOVE-DISK    \ the code for a disk move

\ the beginning of the goalstack.
\ its size is dependent on the max number of disks.
CREATE GOALSTACK   MAX-#DISKS 1 - DUP * 2 + 16 * ALLOT

\ a pointer to the top of the goalstack
VARIABLE GS.PTR

\ empty, top=bottom of the goalstack
GOALSTACK GS.PTR !

VARIABLE SOURCE     \ the last goals source peg
VARIABLE TARGET     \ the last goals target peg
VARIABLE SPARE      \ the last goals spare peg
VARIABLE #DISKS     \ the last goals number of disks
VARIABLE STARTED    \ a flag to indicate
                    \ that solving has begun

\ Goalstack support words

: >GSTACK    ( n addr1 -- addr2 )
   DUP ROT SWAP ! 4 + ;
: GSTACK>    ( addr1 -- addr2 n )
   4 - DUP @ ;

: SPARE!     ( -- )
   SOURCE @   TARGET @   OR 7 XOR   SPARE ! ;
: ADDGOAL    ( n n n n -- )
   GS.PTR @ >GSTACK >GSTACK >GSTACK >GSTACK GS.PTR ! ;
: REMOVEGOAL ( -- )
   GS.PTR @   GSTACK> DROP   GSTACK> #DISKS !
   GSTACK> SOURCE !
   GSTACK> TARGET !
   SPARE!   GS.PTR ! ;

\ references the goal type
\ of the last goal to be removed.
: IS-GOAL    ( n -- flag )
   GS.PTR @ 4 - @ = ;

\ references the # of disks of the last goal
: IS-#-OF-DISKS    ( n -- flag )
   GS.PTR @ 8 - @ = ;

\ prints A, B, or C
\ depending on the peg number (1,2, or 4)
: .PEG  ( n -- )
    DUP 1 = IF
        ." A" DROP
    ELSE
        2 = IF
            ." B"
        ELSE
            ." C"
        THEN
    THEN
;

\ Towers rules ---------------------------------------

\ PRIORITY: 0 is default

\ Like the 1986 PolyForth version except:
\ 0= instead of NOT
\ TRUE STARTED  !  instead of   STARTED TRUE
\ MOVE-TOWER IS-GOAL   1 IS-#-OF-DISKS  AND instead of
\ MOVE-TOWER IS-GOAL AND 1 IS-#-OF-DISKS

*RESET-FORPS*

RULE: START-TOWERS
PRIORITY: 10
*IF* STARTED @ 0=
*THEN*
    GOALSTACK GS.PTR !
    0 0 0 0 ADDGOAL     \ 0 0 0 0 marks bottom of stack
    MOVE-TOWER   #DISKS @   HOME   GOAL   ADDGOAL
    TRUE STARTED  !     \ preventing START-TOWERS from
*END*                   \ every firing again

Rule: MOVE-TOWER-RULE
*IF* MOVE-TOWER IS-GOAL
*THEN*
    REMOVEGOAL
    MOVE-TOWER   #DISKS @ 1 -
      SPARE @   TARGET @   ADDGOAL
    MOVE-DISK    #DISKS @
      SOURCE @   TARGET @   ADDGOAL
    MOVE-TOWER   #DISKS @ 1 -
      SOURCE @   SPARE  @   ADDGOAL
*END*

RULE: MOVE-SINGLE-DISK-TOWER
PRIORITY: 1
*IF* MOVE-TOWER IS-GOAL   1 IS-#-OF-DISKS  AND
*THEN*
    REMOVEGOAL
    MOVE-DISK    1    SOURCE @   TARGET @   ADDGOAL
*END*

RULE: MOVE-SINGLE-DISK
*IF* MOVE-DISK IS-GOAL
*THEN*
    REMOVEGOAL
    TARGET @   SOURCE @
    ."   Move disk on peg " .PEG ."  to peg " .PEG   CR
*END*

\ After compiling the rules, ThisRule
\ contains last rule's number + 1

\ High level Towers word -----------------------------

: .HEADER  ( -- )
    CR CR ." TOWERS OF HANOI   DISKS: "
    #DISKS @ . CR CR ;

: TOWERS   ( disks -- )
    CR  #DISKS !  FALSE STARTED ! .HEADER  FORPS   CR ;
```

Easy Forth - online ebook in JavaScript

Nick Morgan

This small ebook[1] *is here to teach you a programming language called Forth. Forth is a language unlike most others. It's not functional or object oriented, it doesn't have type–checking, and it basically has zero syntax. It was written in the 70s, but is still used today for certain applications. Why would you want to learn such an odd language? Every new programming language you learn helps you think about problems in new ways. Forth is very easy to learn, but it requires you to think in a different way than you're used to. That makes it a perfect language to broaden your coding horizons. This book includes a simple implementation of Forth I wrote in JavaScript. It's by no means perfect, and is missing a lot of the functionality you'd expect in a real Forth system. It's just here to give you an easy way to try out the examples. (If you're a Forth expert, please contribute and make it better!)*

I'm going to assume that you know at least one other programming language, and have a basic idea of how stacks work as a data structure.

Adding Some Numbers

The thing that separates Forth from most other languages is its use of the stack. In Forth, everything revolves around the stack. Any time you type a number, it gets pushed onto the stack. If you want to add two numbers together, typing + takes the top two numbers off the stack, adds them, and puts the result back on the stack. Let's take a look at an example. Type (don't copy–paste) the following into the interpreter, typing Enter after each line.

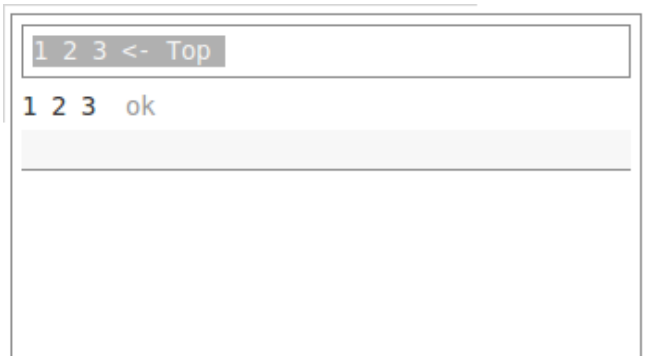


Abbildung 1: Editor–Window

Every time you type a line followed by the Enter key, the Forth interpreter executes that line, and appends the string ok to let you know there were no errors. You should also notice that as you execute each line, the area at the top fills up with numbers. That area is our visualization of the stack. It should look like this:

```
1 2 3 <- Top
```

Now, into the same interpreter, type a single + followed by the Enter key. The top two elements on the stack, 2 and 3, have been replaced by 5.

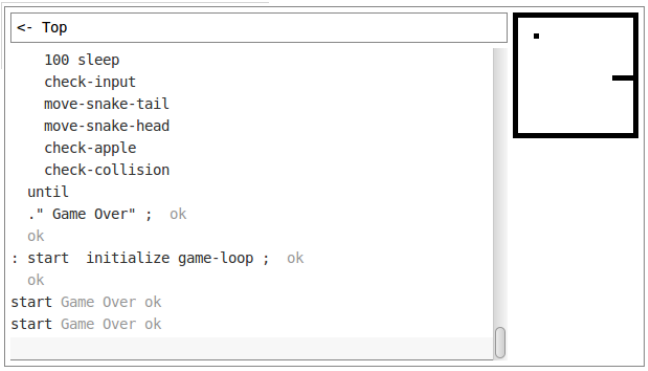
```
1 5 <- Top
```

At this point, your ***editor window*** should look like this:

```
1 ok
2 ok
3 ok
+ ok
```

Type + again and press Enter, and the top two elements will be replaced by 6. If you type + one more time, Forth will try to pop the top two elements off the stack, even though there's only one element on the stack! This results in a Stack underflow error ...

Morgan beschreibt dann die Stack Effects, Defining Words, Stack Manipulation, Generating Output, Conditionals and Loops, Variables and Constants, Arrays, Keyboard Input und schließlich die ***Snake!*** *Und alles immer mit aktivem* ***Editor–Window*** *dazwischen, so dass man die Lektion sofort ausprobieren kann, ohne erst ein Forth installieren zu müssen! Das ist wirklich gut gemacht.*



http://skilldrick.github.io/easyforth/#introduction

[1] *Diese Seite hat Jürgen Pintaske für Euch aus dem Internet gefischt. Hier drucken wir einen Auszug davon an, um zu zeigen, was Nick Morgan da Tolles gemacht hat. Ich bin begeistert! Aber folgt dem Link und seht selbst ... mk*

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
`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 **Ulrich Hoffmann**
Tel.: (04103) – 80 48 41
`uho@forth-ev.de`
Treffen alle 1–2 Monate in loser Folge
Termine unter: `http://forth-ev.de`

Ruhrgebiet **Carsten Strotmann**
`ruhrpott-forth@strotmann.de`
Treffen alle 1–2 Monate Freitags im Unperfekthaus Essen
`http://unperfekthaus.de`.
Termine unter: `http://forth-ev.de`

Mainz Rolf Lauer möchte im Raum Frankfurt, Mainz, Bad Kreuznach eine lokale Gruppe einrichten.
Mail an `rolf@llar.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

Forth-Hardware in VHDL microcore (uCore)	**Klaus Schleisiek** Tel.: **(0 75 45) – 94 97 59 3**$_p$ kschleisiek@freenet.de
KI, Object Oriented Forth, Sicherheitskritische Systeme	**Ulrich Hoffmann** Tel.: **(0 41 03) – 80 48 41** `uho@forth-ev.de`
Forth–Vertrieb volksFORTH ultraFORTH RTX / FG / Super8 KK–FORTH	Ingenieurbüro **Klaus Kohl–Schöpe** Tel.: **(0 82 66)–36 09 862**$_p$

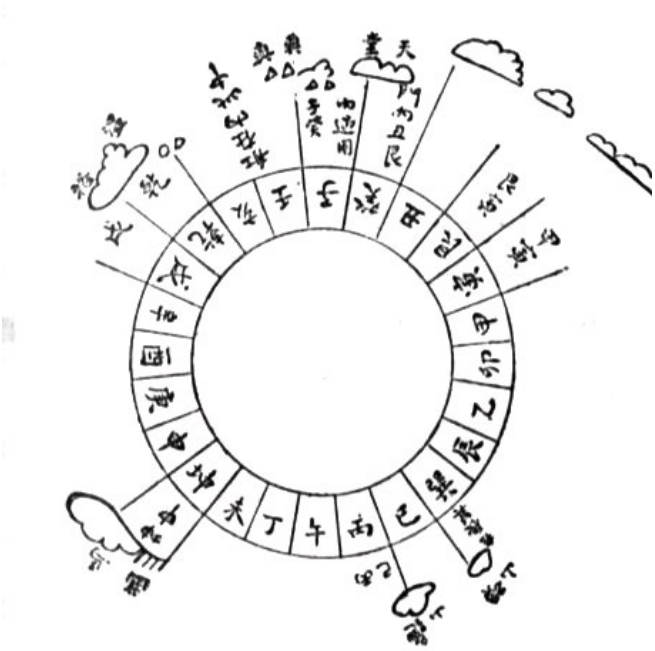
Termine

Donnerstags ab 20:00 Uhr
Forth–Chat net2o `forth@bernd` mit dem Key `keysearch kQusJ`

Freitag, 21.-23.04
Forth–Tagung in Kalkar
`https://tagung.forth-ev.de`

25.-27. August 2017
HCC Hannover Congress Centrum
`https://maker-faire.de/hannover/`

Details zu den Terminen unter `http://forth-ev.de`



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

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

Einladung zur

Forth–Tagung 2017 vom 21. bis 23. April in Kalkar

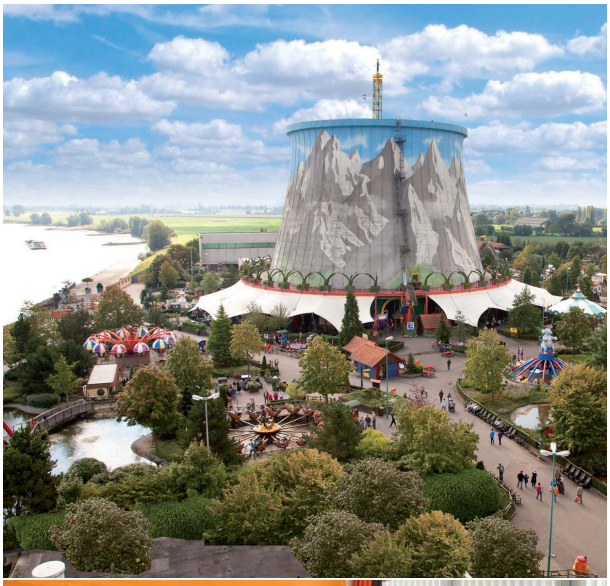
Die Tagung der Deutschen Forthgesellschaft e.V. 2017 findet vom 21. April bis zum 23. April 2017 im Wunderland Kalkar statt. Das Wunderland Kalkar wurde in den Bauruinen des niemals fertiggestellten Kernkraftwerkes Kalkar am Niederrhein errichtet und ist heute ein Freizeitpark mit einigen einmaligen Attraktionen wie dem Kühlturm des geplanten Kraftwerkes. Begleiter und Tagungsteilnehmer können die Angebote des Wunderlandes frei nutzen. Langeweile dürfte daher nicht aufkommen. Im unterirdischen Teil befindet sich eine Partymeile mit vielfältiger Gastronomie.

Kernkraftwerke baut man gemeinhin nicht in Innenstädten von Ballungsgebieten. Das Wunderland liegt daher jwd (janz weit draußen). Die Anreise mit dem PKW wird empfohlen, vom Flughafen Düsseldorf oder von Weeze aus kann ein Shuttelservice genutzt werden. Wir empfehlen die Bildung von Fahrgemeinschaften. Die Familien Bitter bieten an, bis zu 8 Personen ab Mehrhoog (Anbindung an das Netz der DB) mitzunehmen.

Zum Online–Anmeldeformular geht es hier:
`https://tagung.forth-ev.de.`
Bitte auch Mitfahrangebote machen.

Programm

Do.	ab 13:00	Frühankommer(–Workshops)
Fr.	vormittags	Frühankommer(–Workshops)
	nachmittags	Begin der Tagung, Vorträge und Workshops
Sa.	vormittags	Vorträge und Workshops
	nachmittags	Exkursion
So.	09:00	Mitgliederversammlung
	13:00	Ende der Tagung



Bildquelle: Broschüren des Wunderland Kalkars