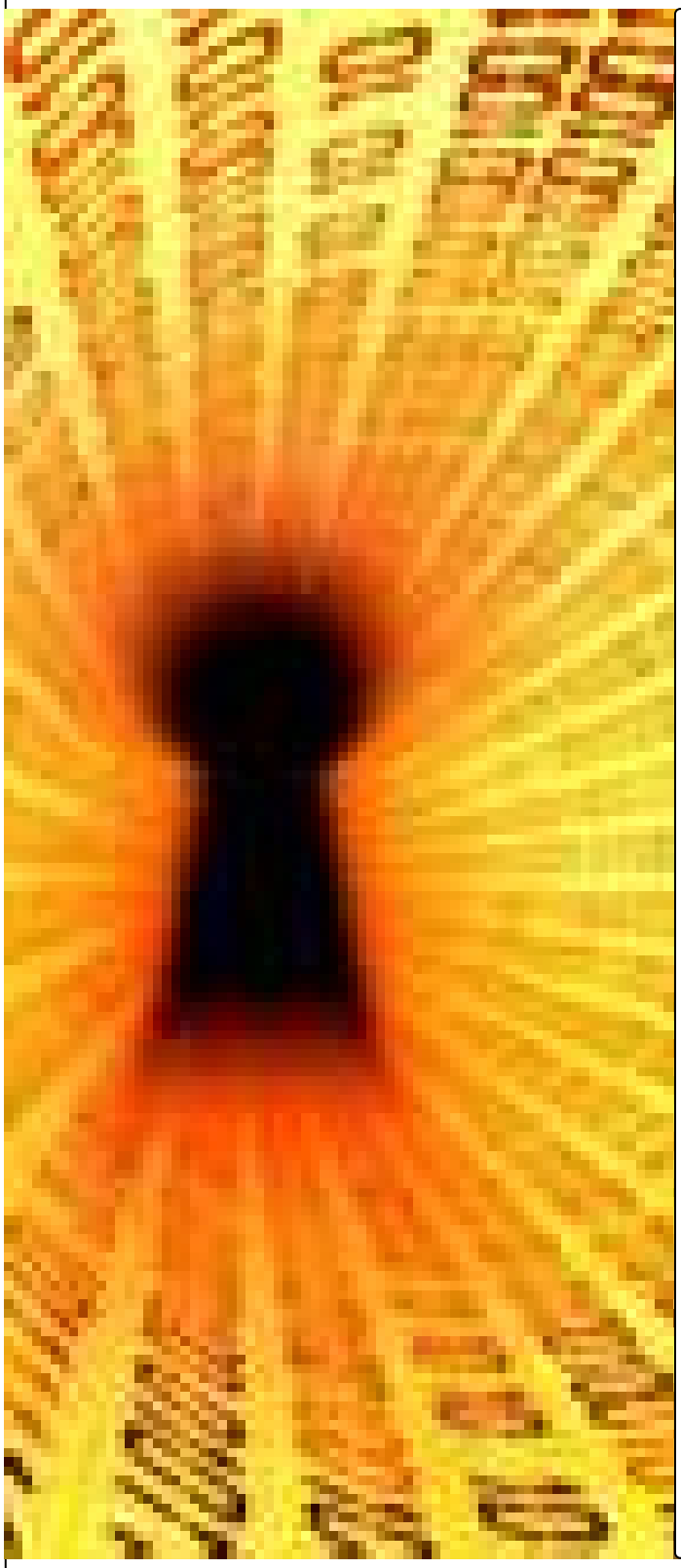
Vierte Dimension

3/2016

Das Forth–Magazin

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

In dieser Ausgabe:



Dienstleistungen und Produkte fördernder Mitglieder des Vereins

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

Liebe Leser,

die Forth–Macher sind aktiv dabei, der technischen Entwicklung der Hardware Rechnung zu tragen. Auf den verschiedenen Mikrocontroller–Familien laufen bereits diverse Forths, wie ihr wisst. Klaus Kohl–Schöpe hat sie alle, und ist, wie ihr seit unserer letzten Forthtagung wisst, *die* Anlaufstelle, wenn es darum geht, die richtige Hardware für das eigene Projekt zu finden, mit oder ohne Forth.
Im Sommer nun sind einige Forthler in Klausur. Das Gforth–Team ist aktiv, Matthias Trute erobert für amforth weitere MCU–Familien und die niederländische HCC bringt lehrreiche Bausätze heraus. Bin gespannt, wie sich das entwickelt und was bis zum Herbst dabei so herausgekommen sein wird.
Weiter gehts mit dem Thema Verschlüsselungs–Algorithmen, die von MCUs geleistet werden können. Rafael Deliano rollt die Entwicklung der Verschlüsselung auf, im letzten Heft das DES, nun das AES. Für das kommende Heft hat er SHA1 schon im Blick.
Ich hoffe, ihr habt alle schon *theforth.net* besucht und erwägt nun ebenfalls, so wie Martin Bitter, eure Schätze dort mitzuteilen. Wie das mit Wachstumskurven so ist, erst fängt es langsam an, dann steiler und steiler. Möge theForth.net ein Erfolg werden.
Was es mit dem Forth–Standard und seiner Entwicklung auf sich hat, erläutert Anton Ertl vom Institut für Computersprachen an der Technischen Universität Wien uns genauer. Wenn euer Weg dort vorbeiführt, versäumt nicht, ihn zu besuchen. Oder seine Webseite. Von den Grundlagen der Programmkonstruktion über wissenschaftliche Methodik und stackbasierte Sprachen (Forth, Postscript) bis zum Übersetzerbau für Programmiersprachen findet ihr Anregung und Skripte auf Anfrage.
Eine Methode, die auch im win32forth verwendet wird, hat Matthias Trute ins amforth geholt. Ob die flexibel ausführbaren Kommandofolgen ein Grund dafür sind, dass Microsoft Security Essentials das win32 hartnäckig als Trojaner einstuft und vor der Verwendung warnt?
Die Konfiguration von Forthprogrammen treibt Bernd Paysan systematisch voran. Dabei erhalten wir einen Ausblick auf Gforth–spezifische Features, die bisher nur in der aktuellen Entwicklerversion verfügbar sind. Praktische Verwendung findet das bereits für sein *net2o*.
Verwirrung stiften gern schon mal mehrzeilige Kommentare im Forth–Quellcode. Manche machen manuell ein \ vor jede Zeile, doch das ist unpraktisch. Das Erscheinungsbild für Mehrzeiliges ist uneinheitlich. Ich habe da schon `(*` und `\\` und `\*` gesehen. *VFX–Forth* hat `((` und Ulrich Hoffmann zeigt, wie das gemacht werden kann.
Und last but not least wurde das *Protokoll der Forth–Tagung 2016* zu Papier gebracht. Herzlichen Dank für deine unermüdliche Vereinsarbeit, Erich!

So, und nicht vergessen, im Herbst ist die *EuroForth–Tagung*. Im Süden des Landes dieses Mal, sehr reizvoll gelegen. 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-2016-03`

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

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

Multiboot–Stick mit SARDU

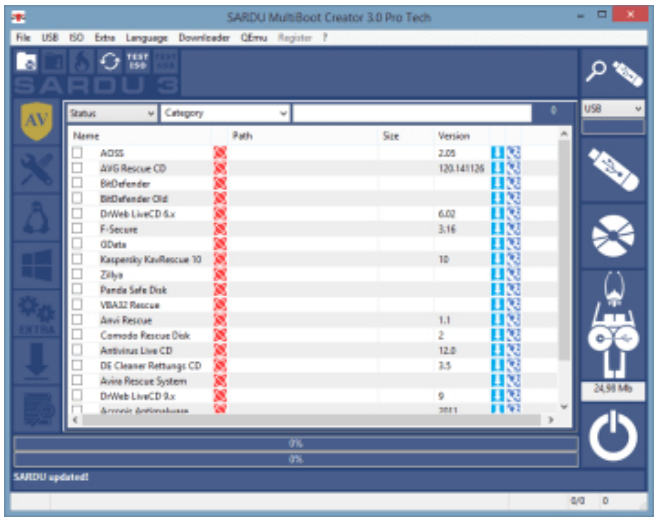
Ich bin dabei, mir einen solchen Stick zu erstellen. Auslöser war ein kürzlich bei Conrad erworbener 64–Gigabyte–Stick zum Sonderpreis von 14,– Euro. Da würden ja dann etwa 30 DVDs draufpassen — oder, etwas weniger riesengroß gedacht, mindestens 128 Live–CDs (als ISO–Dateien). Vom Stick! Direkt aufrufbare x86–Systeme! Eben auch live! Mit allen möglichen und — was mich besonders reizt — unmögichen Betriebssystemen! Und in Form von eigenen Entwicklungsarbeiten sofort beliebig ergänzbar! Die Stimmen, die noch vor 10 Jahren vor den nicht beliebig wieberbeschreibbaren Sticks warnten, sind seit dem großzügigen Ersatz der Festplatten beim Raspi durch SDs verstummt! Oder bin ich da falsch informiert? Und für sofortige Backups hat man ja in der Schublade noch einen zweiten Stick. (Eine ganze Schachtel voll!) Den zweiten kann man ohne Umstände gleich mitlaufen lassen.

Geld spielt da keine Rolle mehr. Aber Zeit! Wie schnell könnte man sich verzetteln. Und eine solche Stick–Anlage baut man sich nicht ein zweites Mal zusammen. Also meine Frage: Wer hilft mir mit eigenen Erfahrungen? Die ganzen VD–Korrekturarbeiten laufen ja bei mir inzwischen auf einem 32–Gigabyte–Nano–Stick ab. Von daher bin ich also mit dem Arbeiten am Stick vertraut. Zwei Festplatten (zu je 200 Gigabyte), eine USB–Platte mit 250 Gigabyte und eine 1–Terabyte–USB–Platte laufen aus Nostalgiegründen bei mir noch mit — oder weil ich sie wirklich nicht sinnvoll einsetzen kann, ohne mich von kleinlichen Herumschraubereien auffressen zu lassen. Ändern tu ich an den Platten nichts mehr. Ich lasse alles so, wie es ist. Zu Vieles habe ich da mit wenig überlegtem „Ändern“ auf diese Weise schon unwiederbringlich verloren.

Hier ein paar Zeilen zu SARDU (Shardana Antivirus Rescue Disk Utility) von Davide Costa aus Italien, genauer aus Sardinien. Das Gerüst habe ich fertig. Hat alles bestens geklappt. Aber jetzt geht es an den Einbau der Bausteine (sprich ISOs). Was will ich, was soll ich, was brauch ich, was muss ich? Natürlich alles auf Forth Bezogene. Aber auch die Dinge aus meinem „früheren“ Leben, dem an der Mathematik orientierten, von dem ich inzwischen immer weniger verstehe.

„Sardu 2.0.2 verwandelt USB–Sticks und DVD–Rohlinge in multibootfähige Allround–Werkzeuge . . .“ (kostenlos).
Fred

www.sarducd.it



AmForth Release 6.2 und 6.3

Amforth hat mit den Versionen 6.2 und 6.3 die Unterstützung für die MSP430–Controller fast auf den Stand der Atmegas gebracht. Die Starthilfe durch Camelforth und 4€4th hat sich ausgezahlt, inzwischen geht amforth deutlich über diese Vorfahren hinaus. Für beide Typen gilt, dass die Unterstützung für den Forth–2012–Standard weit fortgeschritten ist. Viele der von Anton weiter hinten im Heft beschriebenen Dinge funktionieren bereits. Und noch immer passt das Kernsystem, das alles ermöglicht, in 8KB Speicher.
Matthias

amforth.sourceforge.net

Gforth 1.0

Das Gforth–Team (Anton, Bernd und Gerald) hat sich zu einer Klausur getroffen, und den Weg zur nächsten Release freigemacht. Die Release ist noch nicht da, kommt aber bald. Wir hoffen schon im nächsten Heft berichten zu können.
Bernd

Egel Project for MSP430G2553 on Launchpad or Egel Kit

Willem Ouwerkerk und Albert Nijhof haben im Juli diesen Jahres das *Egel Werkboek* in eine Webseite umgesetzt. Ausgelegt ist dieses Arbeitsheft für das *noForth* der beiden.

„The Egel project consists of about 50 elementary examples of hardware control with the MSP430 in the language forth. In each example you find a file with forth code, documentation and links to more information on the internet. It is our aim to lower the hurdles that beginners with the MSP430 MPU find on their way to hardware programming. With the many examples in the Egel project we hope to make it easier to understand the Texas Instruments documentation.“

mk

noforth.bitbucket.org

AES Advanced Encryption Standard

Rafael Deliano

DES war 1976 als Standard verabschiedet worden und in den 90er Jahren absehbar technisch überholt. Triple–DES konnte als Übergangslösung nicht von Dauer sein. Das Aufkommen des Internets ergab neue Anwendungen und Dringlichkeit. Ähnlich wie schon früher das Pentagon bei ADA, entschied sich die US–Behörde NIST dafür, im offenen Wettbewerb den AES genannten offiziellen Nachfolger von DES zu bestimmen [1]. Dessen Anwendung war „sensitive, but not classified information". Die Anforderungen an Sicherheit waren damit moderat. Es sollte genauso sicher, aber effizienter als triple DES sein. Der Algorithmus musste auf unterschiedlichen Plattformen implementierbar sein.

Die Spannweite war enorm: 32–Bit–Pentium als auch 8–Bit–CPUs in Chipkarten mit kaum RAM. Parallelverarbeitung war in schnellen Spezial–ICs wünschenswert. Evaluierung erfolgte, anders als bei DES, nicht durch die NSA. Per Fragebogen stimmten Fachleute anno 2000 auf einer AES–Konferenz von NIST in deutlicher Mehrheit für Rijndael [2] der beiden Belgier Joan Daemen und Vincent Rijmen. Es hatte genug Vorschläge gegeben, die „sicher", aber komplex waren. Der Sieger war auch sicher, aber relativ einfach, gut skalierbar und hatte nachgewiesen, dass er auch auf kleinen Controllern implementierbar war. Es gibt eine Menge gut gemeinter Standards, die sich nicht durchsetzten. AES hat DES noch nicht völlig ersetzt, wurde aber von der Industrie sofort akzeptiert.

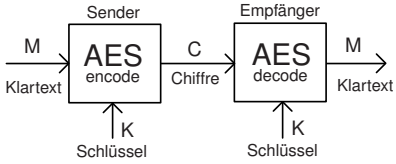


Abbildung 1: Simple Anwendung Blockchiffre

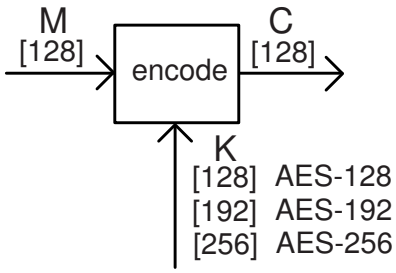


Abbildung 2: Wortlängen der Varianten

Algorithmus

Encoder und Decoder sind getrennte Programme (Abb.1). Um Rechenzeit und Sicherheit besser auf die Anwendung abstimmen zu können, sind drei verschiedene Schlüssellängen vorgesehen (Abb.2). Für ein Komplettpaket muss man also 6 Programme codieren und testen. Anwendungen auf Controllern beschränken sich meist auf AES–128. Die langen Datenworte werden für die weitere Verarbeitung als Byte–Array angeordnet (Abb.3). Der Ablauf teilt sich in die *Key–Expansion,* die für jeden Schritt einen neuen Teilschlüssel bereitstellt, und die Manipulation des Datenblocks. Für Round 0 ist noch keine Key–Expansion nötig: Beide Datenworte werden per XOR verknüpft (Abb.4). Die Zahl der Rounds variiert mit der gewählten AES–Variante.

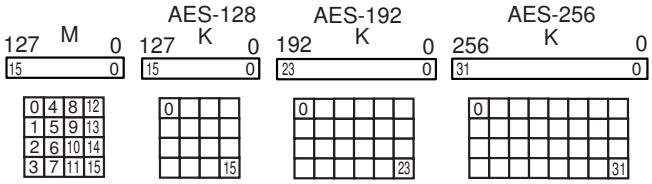


Abbildung 3: Bytes als Blocks angeordnet

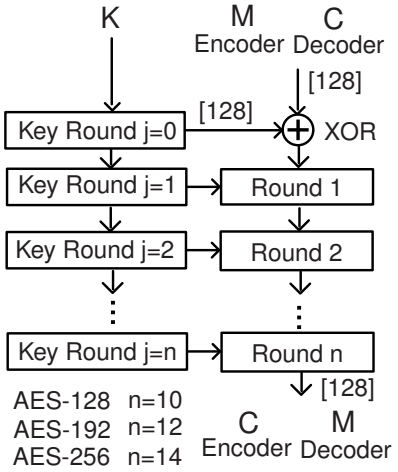


Abbildung 4: Rounds

Datenfluss

Es sind mehrere Teilschritte pro Round nötig (Abb.5). Bei *Subsitute Byte* adressiert der Inhalt des Bytes den Zugriff auf eine 256–Byte–Tabelle, die den neuen Wert enthält (Abb.6). *Shift Rows* rotiert die drei unteren Zeilen. Der *Add Round Key* ist wieder ein 128–Bit–XOR. Mühselig wird es dann bei der Galois–Field–Operation *Mix Columns*: Die Spalten sollen mit einer Matrix multipliziert werden. Die XORs sind 8 Bit breit (Abb.7). Für die Multiplikation werden hier LOG ind INVLOG Tabellen zu je 256 Byte verwendet (Abb.8) Für den Decoder sind außer für *Add Round Key* jeweils Funktionen nötig, die die Veränderung rückgängig machen.

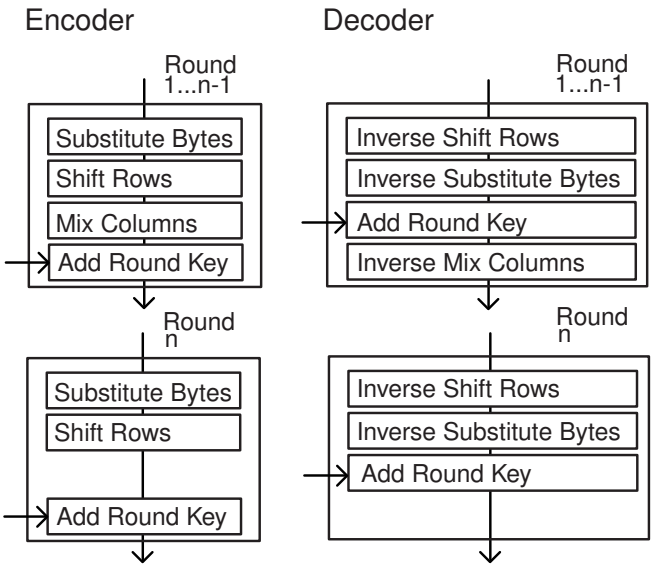


Abbildung 5: Datenfluss

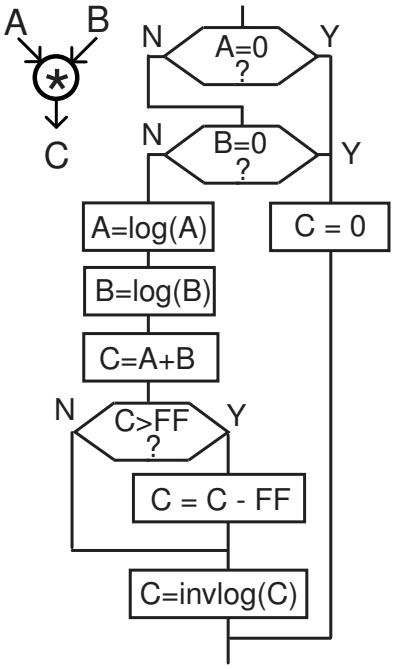


Abbildung 8: Multiplikation in GF(2^8) über LOG–Tabellen

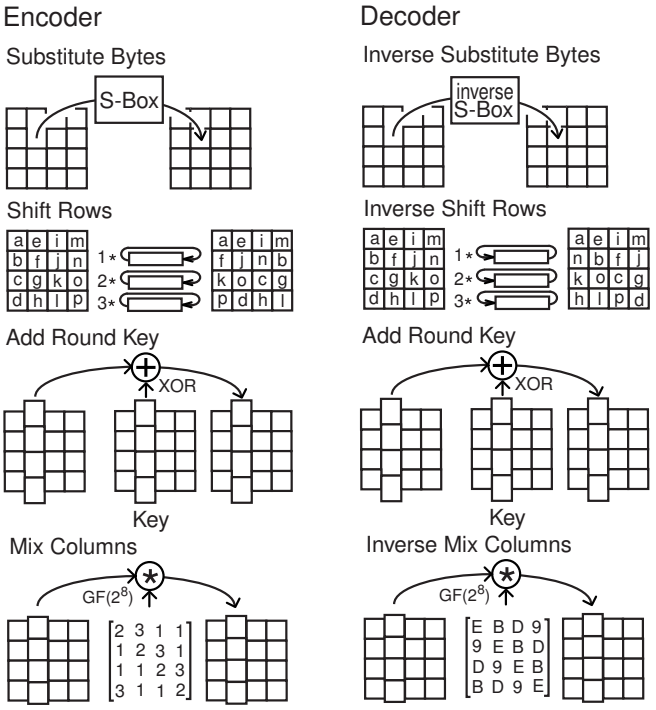


Abbildung 6: Teilfunktionen Datenfluss

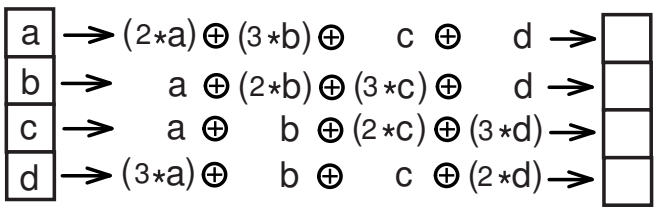


Abbildung 7: Berechnung *Mix Columns*

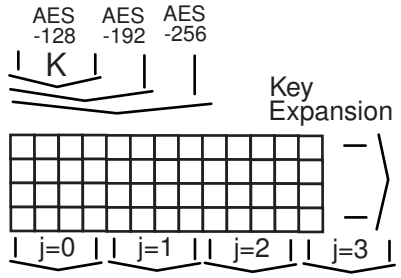


Abbildung 9: Key–Expansion

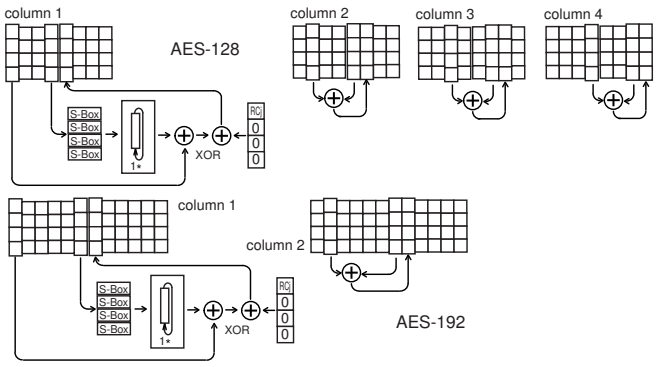


Abbildung 10: Teilfunktionen Key–Expansion

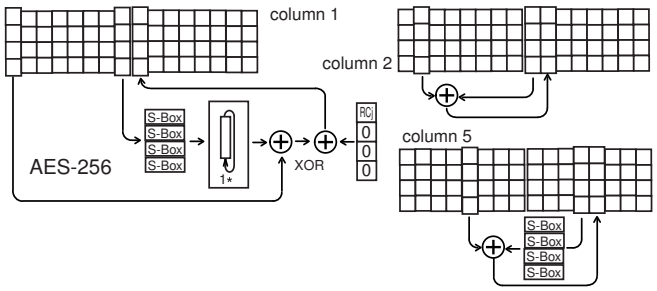


Abbildung 11: Teilfunktionen Key–Expansion AES–256

Key–Expansion

Nach dem Laden des Keys ist für Round 0 bereits ein Schlüssel verfügbar. Danach muss aber die Expansion–Routine anlaufen, die rekursiv weitere Schlüsselspalten erzeugt (Abb.9). Für die erste Spalte wird etwas mehr Aufwand betrieben als für die folgenden Spalten. Zuerst werden die S–BOXen der Datenverschlüsselung auf die letzte Spalte angewendet. Mit *Shift Columns* einmal hoch–rotiert. Dann wird die erste Spalte mit XOR addiert. Eine *Round–Constant* RCj wird addiert. Deren unterstes Byte ist nicht fix, sondern wird anhand des Indexes j aus einer kurzen Tabelle entnommen. Für AES–128 wird der Ablauf komplett für die restlichen Spalten 2, 3, 4 dargestellt. Er ist bei AES–192 nur für die Spalte 2 abgebildet, weil für den Rest völlig identisch. Genauso bei AES–256. Ausnahme dort ist Spalte 5. Der Decoder ruft die Keys in umgekehrter Reihenfolge auf (Abb.12). Es ist sinnvoll, die Key–Expansion nur einmal durchzuführen und das Ergebnis komplett im RAM zu halten. Man benötigt dafür aber 176 / 208 / 240 Byte RAM, je nach AES–Variante.

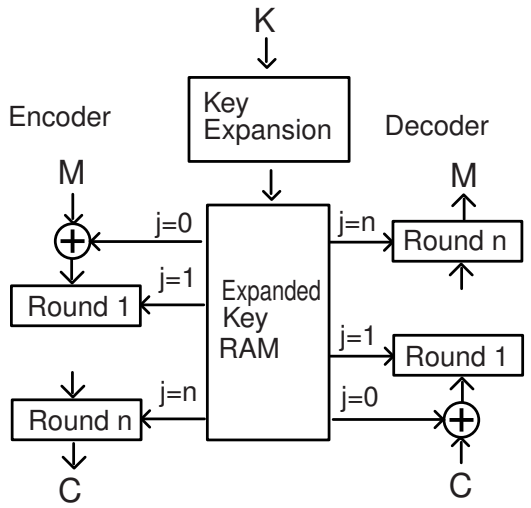


Abbildung 12: Komplette Key–Expansion

Implementierung

Neben 8–Bit–Chipkarten–Controllern lag das Augenmerk der Entwickler insbesondere auf 32–Bit–CPUs. 16–Bit–Forth ist also nicht ideal, aber ein guter Ausgangspunkt für die Portierung auf 8–Bit–Assembler. Detailierte Testvektoren finden sich besonders in [3]. Datenworte werden in der AES–Literatur immer *little endian* dargestellt. Das Daten–Register kann, abhängig von der Betriebsart, Klartext oder Chiffre enthalten. Es wird bei AES in der Literatur als STATE bezeichnet, dieser Name wurde beibehalten. Für KEY wurden hier 256 Byte RAM reserviert. Viel RAM erleichtert die Expansion weil man bequem volle KEY–Blocks berechnen kann. Die ergeben dann für AES–192 8 Byte und für AES–256 16 Byte mehr als benötigt werden. Getestet wurde auf einem MC68HC908GP32 mit 2,54MHz Busfrequenz (Tab.1und Tab.2). Verdoppelte Schlüssellänge schlägt nicht voll auf den Speicherverbrauch und die Geschwindigkeit durch. Alle drei Versionen zusammen würden nur ca. 4kByte Flash benötigen, da Unterprogramme und Tabellen einheitlich sind. Assembler dürfte meist nötig sein, um brauchbare Geschwindigkeit zu erreichen. Dem steht der Entwicklungsaufwand entgegen. Hier wurde auf die Portierung der *Key–Expansion* verzichtet und nur die Unterprogramme wurden neu codiert. STATE muss nun in die ZeroPage, damit passende Opcodes verfügbar sind. Wie man sieht, erreicht man mit wenig Aufwand gute Wirkung. 100% Assembler würden vermutlich nur noch 20% mehr Leistung bringen. Das Listing enthält nur AES–128. Für die anderen Varianten genügt eine email an den Autor. Man beachte bei der Portierung, dass bei DO...LOOP in nanoFORTH der Index anders zählt.

Forth	AES–128	AES–192	AES–256
Key Eyp. [msec]	60	65	70
Encoder [msec]	200	250	300
Decoder [msec]	230	350	400
Flash [kByte]	3,5K	3,5K	3,5K
RAM [Byte]	209	249	289

Tabelle 1: Speicher & Timing in Forth

Assembler	AES–128	AES–192	AES–256
Encoder [msec]	10	13	16
Decoder [msec]	20	25	30
Flash [kByte]	2,5K	2,5K	2,5K

Tabelle 2: Speicher & Timing in Assembler

Referenzen

[1] Daemen, Rijmen "The Design of Rijndael" Springer 2001

[2] Daemen, Rijmen "AES–Proposal: Rijndael" Version 2 März 1999 pdf

[3] FIPS 197 "Advanced Encryption Standard (AES)" November 2001 pdf

[4] Gladman "A Specification for the AES Algorithm" 2002 pdf

Listings

```
1   <| \ AES128.txt
2
3   $HEAP  CONSTANT KEY       \ 256 byte RAM Exp.Key
4   10     ZVARIABLE STATE    \  16 byte RAM
5   10     ZVARIABLE STATE"   \  16 byte RAM Scratch
6   STATE" CONSTANT T         \   4 byte RAM
7   1      ZVARIABLE J"
8
9   TABLE SBOX
10  63 C, 7C C, 77 C, 7B C, F2 C, 6B C, 6F C, C5 C,
11  30 C, 01 C, 67 C, 2B C, FE C, D7 C, AB C, 76 C,
12  CA C, 82 C, C9 C, 7D C, FA C, 59 C, 47 C, F0 C,
13  AD C, D4 C, A2 C, AF C, 9C C, A4 C, 72 C, C0 C,
14  B7 C, FD C, 93 C, 26 C, 36 C, 3F C, F7 C, CC C,
15  34 C, A5 C, E5 C, F1 C, 71 C, D8 C, 31 C, 15 C,
```

```
16   04 C, C7 C, 23 C, C3 C, 18 C, 96 C, 05 C, 9A C,
17   07 C, 12 C, 80 C, E2 C, EB C, 27 C, B2 C, 75 C,
18   09 C, 83 C, 2C C, 1A C, 1B C, 6E C, 5A C, A0 C,
19   52 C, 3B C, D6 C, B3 C, 29 C, E3 C, 2F C, 84 C,
20   53 C, D1 C, 00 C, ED C, 20 C, FC C, B1 C, 5B C,
21   6A C, CB C, BE C, 39 C, 4A C, 4C C, 58 C, CF C,
22   D0 C, EF C, AA C, FB C, 43 C, 4D C, 33 C, 85 C,
23   45 C, F9 C, 02 C, 7F C, 50 C, 3C C, 9F C, A8 C,
24   51 C, A3 C, 40 C, 8F C, 92 C, 9D C, 38 C, F5 C,
25   BC C, B6 C, DA C, 21 C, 10 C, FF C, F3 C, D2 C,
26   CD C, 0C C, 13 C, EC C, 5F C, 97 C, 44 C, 17 C,
27   C4 C, A7 C, 7E C, 3D C, 64 C, 5D C, 19 C, 73 C,
28   60 C, 81 C, 4F C, DC C, 22 C, 2A C, 90 C, 88 C,
29   46 C, EE C, B8 C, 14 C, DE C, 5E C, 0B C, DB C,
30   E0 C, 32 C, 3A C, 0A C, 49 C, 06 C, 24 C, 5C C,
31   C2 C, D3 C, AC C, 62 C, 91 C, 95 C, E4 C, 79 C,
32   E7 C, C8 C, 37 C, 6D C, 8D C, D5 C, 4E C, A9 C,
33   6C C, 56 C, F4 C, EA C, 65 C, 7A C, AE C, 08 C,
34   BA C, 78 C, 25 C, 2E C, 1C C, A6 C, B4 C, C6 C,
35   E8 C, DD C, 74 C, 1F C, 4B C, BD C, 8B C, 8A C,
36   70 C, 3E C, B5 C, 66 C, 48 C, 03 C, F6 C, 0E C,
37   61 C, 35 C, 57 C, B9 C, 86 C, C1 C, 1D C, 9E C,
38   E1 C, F8 C, 98 C, 11 C, 69 C, D9 C, 8E C, 94 C,
39   9B C, 1E C, 87 C, E9 C, CE C, 55 C, 28 C, DF C,
40   8C C, A1 C, 89 C, 0D C, BF C, E6 C, 42 C, 68 C,
41   41 C, 99 C, 2D C, 0F C, B0 C, 54 C, BB C, 16 C,
42
43   TABLE INV-SBOX
44   52 C, 09 C, 6A C, D5 C, 30 C, 36 C, A5 C, 38 C,
45   BF C, 40 C, A3 C, 9E C, 81 C, F3 C, D7 C, FB C,
46   7C C, E3 C, 39 C, 82 C, 9B C, 2F C, FF C, 87 C,
47   34 C, 8E C, 43 C, 44 C, C4 C, DE C, E9 C, CB C,
48   54 C, 7B C, 94 C, 32 C, A6 C, C2 C, 23 C, 3D C,
49   EE C, 4C C, 95 C, 0B C, 42 C, FA C, C3 C, 4E C,
50   08 C, 2E C, A1 C, 66 C, 28 C, D9 C, 24 C, B2 C,
51   76 C, 5B C, A2 C, 49 C, 6D C, 8B C, D1 C, 25 C,
52   72 C, F8 C, F6 C, 64 C, 86 C, 68 C, 98 C, 16 C,
53   D4 C, A4 C, 5C C, CC C, 5D C, 65 C, B6 C, 92 C,
54   6C C, 70 C, 48 C, 50 C, FD C, ED C, B9 C, DA C,
55   5E C, 15 C, 46 C, 57 C, A7 C, 8D C, 9D C, 84 C,
56   90 C, D8 C, AB C, 00 C, 8C C, BC C, D3 C, 0A C,
57   F7 C, E4 C, 58 C, 05 C, B8 C, B3 C, 45 C, 06 C,
58   D0 C, 2C C, 1E C, 8F C, CA C, 3F C, 0F C, 02 C,
59   C1 C, AF C, BD C, 03 C, 01 C, 13 C, 8A C, 6B C,
60   3A C, 91 C, 11 C, 41 C, 4F C, 67 C, DC C, EA C,
61   97 C, F2 C, CF C, CE C, F0 C, B4 C, E6 C, 73 C,
62   96 C, AC C, 74 C, 22 C, E7 C, AD C, 35 C, 85 C,
63   E2 C, F9 C, 37 C, E8 C, 1C C, 75 C, DF C, 6E C,
64   47 C, F1 C, 1A C, 71 C, 1D C, 29 C, C5 C, 89 C,
65   6F C, B7 C, 62 C, 0E C, AA C, 18 C, BE C, 1B C,
66   FC C, 56 C, 3E C, 4B C, C6 C, D2 C, 79 C, 20 C,
67   9A C, DB C, C0 C, FE C, 78 C, CD C, 5A C, F4 C,
68   1F C, DD C, A8 C, 33 C, 88 C, 07 C, C7 C, 31 C,
69   B1 C, 12 C, 10 C, 59 C, 27 C, 80 C, EC C, 5F C,
70   60 C, 51 C, 7F C, A9 C, 19 C, B5 C, 4A C, 0D C,
71   2D C, E5 C, 7A C, 9F C, 93 C, C9 C, 9C C, EF C,
72   A0 C, E0 C, 3B C, 4D C, AE C, 2A C, F5 C, B0 C,
73   C8 C, EB C, BB C, 3C C, 83 C, 53 C, 99 C, 61 C,
74   17 C, 2B C, 04 C, 7E C, BA C, 77 C, D6 C, 26 C,
75   E1 C, 69 C, 14 C, 63 C, 55 C, 21 C, 0C C, 7D C,
76
77   TABLE RC
78   8D C,  \ unused dummy
79   01 C, 02 C,
80   04 C, 08 C,
81   10 C, 20 C,
82   40 C, 80 C,
83   1B C, 36 C, \ AES-128
84   6C C, d8 C, \ AES-192
85   AB C, 4D C, \ AES-256
86
87   TABLE LOG
88   00 C, 00 C, 19 C, 01 C, 32 C, 02 C, 1a C, c6 C,
89   4b C, c7 C, 1b C, 68 C, 33 C, ee C, df C, 03 C,
90   64 C, 04 C, e0 C, 0e C, 34 C, 8d C, 81 C, ef C,
91   4c C, 71 C, 08 C, c8 C, f8 C, 69 C, 1c C, c1 C,
92   7d C, c2 C, 1d C, b5 C, f9 C, b9 C, 27 C, 6a C,
93   4d C, e4 C, a6 C, 72 C, 9a C, c9 C, 09 C, 78 C,
94   65 C, 2f C, 8a C, 05 C, 21 C, 0f C, e1 C, 24 C,
95   12 C, f0 C, 82 C, 45 C, 35 C, 93 C, da C, 8e C,
96   96 C, 8f C, db C, bd C, 36 C, d0 C, ce C, 94 C,
97   13 C, 5c C, d2 C, f1 C, 40 C, 46 C, 83 C, 38 C,
98   66 C, dd C, fd C, 30 C, bf C, 06 C, 8b C, 62 C,
99   b3 C, 25 C, e2 C, 98 C, 22 C, 88 C, 91 C, 10 C,
100  7e C, 6e C, 48 C, c3 C, a3 C, b6 C, 1e C, 42 C,
101  3a C, 6b C, 28 C, 54 C, fa C, 85 C, 3d C, ba C,
102  2b C, 79 C, 0a C, 15 C, 9b C, 9f C, 5e C, ca C,
103  4e C, d4 C, ac C, e5 C, f3 C, 73 C, a7 C, 57 C,
104  af C, 58 C, a8 C, 50 C, f4 C, ea C, d6 C, 74 C,
105  4f C, ae C, e9 C, d5 C, e7 C, e6 C, ad C, e8 C,
106  2c C, d7 C, 75 C, 7a C, eb C, 16 C, 0b C, f5 C,
107  59 C, cb C, 5f C, b0 C, 9c C, a9 C, 51 C, a0 C,
108  7f C, 0c C, f6 C, 6f C, 17 C, c4 C, 49 C, ec C,
109  d8 C, 43 C, 1f C, 2d C, a4 C, 76 C, 7b C, b7 C,
110  cc C, bb C, 3e C, 5a C, fb C, 60 C, b1 C, 86 C,
111  3b C, 52 C, a1 C, 6c C, aa C, 55 C, 29 C, 9d C,
112  97 C, b2 C, 87 C, 90 C, 61 C, be C, dc C, fc C,
113  bc C, 95 C, cf C, cd C, 37 C, 3f C, 5b C, d1 C,
114  53 C, 39 C, 84 C, 3c C, 41 C, a2 C, 6d C, 47 C,
115  14 C, 2a C, 9e C, 5d C, 56 C, f2 C, d3 C, ab C,
116  44 C, 11 C, 92 C, d9 C, 23 C, 20 C, 2e C, 89 C,
117  b4 C, 7c C, b8 C, 26 C, 77 C, 99 C, e3 C, a5 C,
118  67 C, 4a C, ed C, de C, c5 C, 31 C, fe C, 18 C,
119  0d C, 63 C, 8c C, 80 C, c0 C, f7 C, 70 C, 07 C,
120
121  TABLE INVLOG
122  01 C, 03 C, 05 C, 0f C, 11 C, 33 C, 55 C, ff C,
123  1a C, 2e C, 72 C, 96 C, a1 C, f8 C, 13 C, 35 C,
124  5f C, e1 C, 38 C, 48 C, d8 C, 73 C, 95 C, a4 C,
125  f7 C, 02 C, 06 C, 0a C, 1e C, 22 C, 66 C, aa C,
126  e5 C, 34 C, 5c C, e4 C, 37 C, 59 C, eb C, 26 C,
127  6a C, be C, d9 C, 70 C, 90 C, ab C, e6 C, 31 C,
128  53 C, f5 C, 04 C, 0c C, 14 C, 3c C, 44 C, cc C,
129  4f C, d1 C, 68 C, b8 C, d3 C, 6e C, b2 C, cd C,
130  4c C, d4 C, 67 C, a9 C, e0 C, 3b C, 4d C, d7 C,
131  62 C, a6 C, f1 C, 08 C, 18 C, 28 C, 78 C, 88 C,
132  83 C, 9e C, b9 C, d0 C, 6b C, bd C, dc C, 7f C,
133  81 C, 98 C, b3 C, ce C, 49 C, db C, 76 C, 9a C,
134  b5 C, c4 C, 57 C, f9 C, 10 C, 30 C, 50 C, f0 C,
135  0b C, 1d C, 27 C, 69 C, bb C, d6 C, 61 C, a3 C,
136  fe C, 19 C, 2b C, 7d C, 87 C, 92 C, ad C, ec C,
137  2f C, 71 C, 93 C, ae C, e9 C, 20 C, 60 C, a0 C,
138  fb C, 16 C, 3a C, 4e C, d2 C, 6d C, b7 C, c2 C,
139  5d C, e7 C, 32 C, 56 C, fa C, 15 C, 3f C, 41 C,
140  c3 C, 5e C, e2 C, 3d C, 47 C, c9 C, 40 C, c0 C,
141  5b C, ed C, 2c C, 74 C, 9c C, bf C, da C, 75 C,
142  9f C, ba C, d5 C, 64 C, ac C, ef C, 2a C, 7e C,
143  82 C, 9d C, bc C, df C, 7a C, 8e C, 89 C, 80 C,
144  9b C, b6 C, c1 C, 58 C, e8 C, 23 C, 65 C, af C,
145  ea C, 25 C, 6f C, b1 C, c8 C, 43 C, c5 C, 54 C,
146  fc C, 1f C, 21 C, 63 C, a5 C, f4 C, 07 C, 09 C,
147  1b C, 2d C, 77 C, 99 C, b0 C, cb C, 46 C, ca C,
148  45 C, cf C, 4a C, de C, 79 C, 8b C, 86 C, 91 C,
149  a8 C, e3 C, 3e C, 42 C, c6 C, 51 C, f3 C, 0e C,
150  12 C, 36 C, 5a C, ee C, 29 C, 7b C, 8d C, 8c C,
151  8f C, 8a C, 85 C, 94 C, a7 C, f2 C, 0d C, 17 C,
152  39 C, 4b C, dd C, 7c C, 84 C, 97 C, a2 C, fd C,
153  1c C, 24 C, 6c C, b4 C, c7 C, 52 C, f6 C, 01 C,
154
155  : (*)         \ ( UC1 UC2 --- UC3 )
156  OVER IF LOG + C@ SWAP LOG + C@ +
157  DUP FF00 AND IF FF - THEN
158  INVLOG + C@ ELSE 2DROP 0 THEN  ;
159
160  : 2(*) 2 (*) ;  : 3(*) 3 (*) ;
161  : E(*) E (*) ;  : B(*) B (*) ;
162  : D(*) D (*) ;  : 9(*) 9 (*) ;
163
164  : 4+ 4 + ; : 3+ 3 + ; : 4- 4 - ;
165
166  : KEY+   KEY +   ; : STATE+ STATE + ;
167
```

```
\ AES-128 Key Expansion ------------------

: 10- 10 - ;

: 1.COL \ ( i --- i+4 )
3 0 DO
DUP KEY+ I + 4- C@ SBOX + C@ T I + C!
LOOP
T C@ T 1+ C@  T    C!  \ rotate column up
     T 2+ C@  T 1+ C!
     T 3+ C@  T 2+ C!
              T 3+ C!
3 0 DO
  DUP KEY+ I + 10- C@ T I + C@ XOR T I + C!
LOOP
DUP 4SHIFT> RC + C@ T C@ XOR T C!
3 0 DO
  T I + C@ OVER KEY+ I + C!
LOOP 4+ ;

: nCOL \ ( i --- i+4 )
3 0 DO
   DUP KEY+  4- I + C@
  OVER KEY+ 10- I + C@ XOR
  OVER KEY+     I + C!
LOOP 4+ ;

: SKEY \ ( i --- i+16 ) new 16 bytes
1.COL nCOL nCOL nCOL ;

: KEY-EXP \ ( --- ) Key Expansion
10 A 1 DO SKEY LOOP DROP ;

\ AES-128 Encoder --------------------------

: SUB-BYTE     \ ( --- )
F 0 DO I STATE+ C@ SBOX + C@ I STATE+ C! LOOP ;

: M   \ ( to from --- ) move byte
STATE" + C@ SWAP STATE+ C! ;

: SHIFT-ROW
STATE STATE" 10 CMOVE
1 5 M  5 9 M  9 D M  D 1 M
2 A M  6 E M  A 2 M  E 6 M
3 F M  7 3 M  B 7 M  F B M ;

: MIX-COLUMNS
D% 16 0 DO
I STATE+    C@ 2(*)
I STATE+ 1+ C@ 3(*) XOR
I STATE+ 2+ C@      XOR
I STATE+ 3+ C@      XOR \ ( --- UC0+ )
I STATE+    C@
I STATE+ 1+ C@ 2(*) XOR
I STATE+ 2+ C@ 3(*) XOR
I STATE+ 3+ C@      XOR \ ( --- UC0+ UC1+ )
I STATE+    C@
I STATE+ 1+ C@      XOR
I STATE+ 2+ C@ 2(*) XOR
I STATE+ 3+ C@ 3(*) XOR \ ( --- UC0+ UC1+ UC2+ )
I STATE+    C@ 3(*)
I STATE+ 1+ C@      XOR
I STATE+ 2+ C@      XOR
I STATE+ 3+ C@ 2(*) XOR I STATE+ 3+ C!
                        I STATE+ 2+ C!
                        I STATE+ 1+ C!
                        I STATE+    C!
4 +LOOP ;

: ADD-KEY   \  ( J --- J )
F 0 DO
DUP KEY+ I + C@ I STATE+ C@ XOR I STATE+ C!
LOOP ;

: ENC     \ (  ---  ) Keys are exp. in KEY
0 ADD-KEY 10 +
9 1 DO
  SUB-BYTE SHIFT-ROW MIX-COLUMNS ADD-KEY 10 +
LOOP
  SUB-BYTE SHIFT-ROW           ADD-KEY DROP ;

\ AES-128 Decoder --------------------------

: INV-SUB-BYTE     \ ( --- )
F 0 DO I STATE+ C@ INV-SBOX + C@ I STATE+ C! LOOP ;

: INV-SHIFT-ROW
STATE STATE" 10 CMOVE
5 1 M  9 5 M  D 9 M  1 D M
A 2 M  E 6 M  2 A M  6 E M
F 3 M  3 7 M  7 B M  B F M ;

: INV-MIX-COLUMNS
D% 16 0 DO
I STATE+    C@ E(*)
I STATE+ 1+ C@ B(*) XOR
I STATE+ 2+ C@ D(*) XOR
I STATE+ 3+ C@ 9(*) XOR \ ( --- UC0+ )
I STATE+    C@ 9(*)
I STATE+ 1+ C@ E(*) XOR
I STATE+ 2+ C@ B(*) XOR
I STATE+ 3+ C@ D(*) XOR \ ( --- UC0+ UC1+ )
I STATE+    C@ D(*)
I STATE+ 1+ C@ 9(*) XOR
I STATE+ 2+ C@ E(*) XOR
I STATE+ 3+ C@ B(*) XOR \ (--- UC0+ UC1+ UC2+ )
I STATE+    C@ B(*)
I STATE+ 1+ C@ D(*) XOR
I STATE+ 2+ C@ 9(*) XOR
I STATE+ 3+ C@ E(*) XOR I STATE+ 3+ C!
                        I STATE+ 2+ C!
                        I STATE+ 1+ C!
                        I STATE+    C!
4 +LOOP ;

: DECO     \ (  ---  ) Keys are exp. in KEY
A0 ADD-KEY 10 -
9 1 DO
  INV-SHIFT-ROW INV-SUB-BYTE ADD-KEY INV-MIX-COLUMNS 10 -
LOOP
  INV-SHIFT-ROW INV-SUB-BYTE ADD-KEY DROP ;

\ AES-128 Test -------------------------------

TABLE TEST-DATA1 \ little endian  Plain text
32 C, 43 C, f6 C, a8 C, 88 C, 5a C, 30 C, 8d C,
31 C, 31 C, 98 C, a2 C, e0 C, 37 C, 07 C, 34 C,

TABLE TEST-KEY1 \ little endian
2B C, 7E C, 15 C, 16 C, 28 C, AE C, D2 C, A6 C,
AB C, F7 C, 15 C, 88 C, 09 C, CF C, 4F C, 3C C,

: LOAD1     \ (  ---  )
TEST-DATA1 STATE D% 16 CMOVE
TEST-KEY1  KEY   D% 16 CMOVE ;

: TEST-ENC     LOAD1 KEY-EXP ENC ;

TABLE TEST-iDATA1 \ little endian   Chiffre
39 C, 25 C, 84 C, 1d C, 02 C, dc C, 09 C, fb C,
dc C, 11 C, 85 C, 97 C, 19 C, 6a C, 0b C, 32 C,

: iLOAD1
TEST-iDATA1 STATE D% 16 CMOVE
TEST-KEY1   KEY   D% 16 CMOVE ;

: TEST-DEC  iLOAD1 KEY-EXP DECO ;

|>
```

mwords — ein Tool

Martin Bitter

Seit Jahren begleitet mich auf verschiedenen Forth–Systemen ein Wort, das ich `mwords` *nenne. Nachdem Gerald Wodni auf der Forth–Tagung 2016 in Augsburg das Forth–Repository [1] vorgestellt hat, habe ich* `mwords` *überarbeitet und an den Standard Forth–2012 (hoffentlich) angepasst.* `mwords` *sollte nun auf allen Forth–Systemen dieses Standards funktionieren. Interessenten können es im Forth–Repository finden.*[1]

Warum habe ich `mwords` geschrieben? *Bigforth*, *gforth* und andere Systeme kommen mit einer Vielzahl an Worten. Im Falle von *gforth* sind es in der Grundinstallation über 2000 Worte! `WORDS` listet sie alle blitzschnell auf, ist aber durch die schiere Menge der Worte unübersichtlich. Dem hilft `mwords` ab. Es liest einen String von der Eingabe und listet alle Words auf, die diesen String enthalten. Wenn ich nicht mehr weiß, ob es `open-file` oder `file-open` heißt, gebe ich einfach

```
mwords open
```

ein und erhalte eine kurze Liste von Worten, die „open“ enthalten. Voilà! `open-file` ist auch dabei. Mittels `mwords` kann ich ebenfalls alle Versionen von `dup` herausfinden:

```
mwords dup
```

zeigt mir `?DUP-0=-IF ?DUP-IF fdup 2dup ?dup dup ?dup-0=-?branch ?dup-?branch`

Der Vollständigkeit halber habe ich ein weiteres Wort `voc-mwords` geschrieben, das das Gleiche wie `mwords` macht, aber dabei alle erreichbaren Vocabularies (wordlists) durchsucht.

Sag mir, wo du stehst![2]

Der Quelltext findet sich an oben genanntem Ort. `mwords` macht Gebrauch von `map-wordlist` und `voc-mwords` zusätzlich von `traverse-wordlist`. Beide formatieren sie ihre Ausgabe. Das war nicht so einfach, denn gforth hat keine Abfrage der Cursorposition. Grund dafür ist, dass die verwendete Terminalemulation (OS–Aufruf) dies ebenfalls nicht anbietet. Um so etwas wie `?cr` zu verwenden, musste ich einen eigenen Positionscounter implementieren[3]. Nun zählt jede Ausgabe von `mwords` bzw. `voc-mwords`, wie viele Stellen sie verbraucht, und führt gegebenenfalls ein *Carriage–Return* aus. Der Quelltext im Forth–Repository ist gut in BSE[4] kommentiert (hoffe ich). Wer's lieber in Deutsch mag, kann sich an mich wenden.

Link

[1]www.theforth.net

Listing

```
1   Get-current               \ get wid of current wordlist; wid is on TOS
2   vocabulary mwords-hide    \ create a wordlist 'mwords-hide'
3   also mwords-hide          \ put it into the search order
4   definitions               \ make it the compilation wordlist
5
6   : UPPERCASE ( c-str u -- )      \ changes all chars of c-str to upper case
7     0 ?DO count toupper over 1- c!
8       LOOP drop ;
9
10  : target$ ( -- c-str u )        \ string to hit
11     here count ;
12
13  : name$ ( -- c-str u )          \ name string of actual word
14     here count + count ;
15
16  : match? ( nt -- flag )         \ does the name token match the target string?
17       name>string                \ get the corresponding string and
18       target$ + place            \ place as 2nd string to here
19       name$ 2dup UPPERCASE       \ capitalize it
20       target$                    \ string to match to is here
21       search nip nip  ;          \ part of 2nd string?
22
23  : target ( /str -- )            \ reads string to match to from input
24       bl word count 2dup UPPERCASE    \ in gforth the parsed word is in here
25       here place  ;              \ but could we rely on it?
```

[1] Entweder das Zipfile downloaden oder besser: Das Repositorytool 'f' installieren und `finclude mwords 1.0.0` aufrufen.

[2] https://de.wikipedia.org/wiki/Sag_mir,_wo_du_stehst

[3] Einfaches Ändern von `type`, `emit`, `cr` hat nicht funktioniert.

[4] BSE = bad simple English

```
26
27
28
29  \ ********** formatting words ************************************
30
31  5 Value margin                  \ left margin for formatted output
32
33  : cr+margin ( -- n )            \ start a new line beginning at position of margin
34    cr margin dup spaces ;
35
36  : new_pos ( n -- n' )           \ look into the glass bowl where the new (cursor) position will be
37     name$ nip 1+ +               \ new position (old pos + length of name string + space)
38     dup cols margin - >          \ will reach end of line?
39     IF drop cr+margin name$ nip +    \ if --> cr
40     THEN ;
41
42
43
44  \ ********** the searching routines ******************************
45
46  : .mword ( x-pos nt -- x-pos )      \ prints the name of current word in wordlist if it matches the target
47     dup                              \ duplicate the name token
48     match?                           \ does it match?
49     IF swap new_pos swap .name       \ if print it
50     ELSE drop                        \ else drop name token
51     THEN ;                           \ finish
52
53  : exist-mword? ( flag flag nt -- flag ) \ 'match?' version fitting to traverse-wordlist
54     match?                           \ old match?
55     IF    drop true false            \ drop flag leave one flag for traverse-wordlist one for further use
56     ELSE  true                       \ did not match so true tells traverse-wordlist to go on
57     THEN ;                           \ finish
58
59  : mwords? ( wid -- flag )           \ looks if there is at least one word in wordlist (wid) that matches
60    false                             \ if this flag isn't changed later there is no matching word
61    ['] exist-mword?                  \ xt of searching routine
62    rot traverse-wordlist ;           \ traverse-wordlist will use the above xt with the given wid
63
64  : .voc-mword ( wid -- )             \ formatted list of matching words of wordlist wid
65    dup mwords?                       \ duplicate wid and look if there is at least one word in wordlist (wid)
66                                      \ that matches
67    IF dup cr ." Vocabulary: " .voc   \ if print name of wordlist
68       cr+margin                      \ new line with margin
69       swap ['] .mword map-wordlist drop  \ list matching words
70    ELSE  drop                        \ if not drop duplicated wid
71    THEN ;                            \ finish
72
73
74
75  \ ********** the main words: mwords and voc-mwords ******************************
76
77  set-current                         \ restore original (i.e., public) compilation wordlist
78
79  : mwords ( / "STR" -- )             \ list all words of current wordlist that match the parsed string
80    target                            \ parse string to here
81    cr+margin get-current             \ new line with margin, get wid of current wordlist
82    ['] .mword map-wordlist drop ;    \ map wordlist executes .mwords
83
84  : voc-mwords ( / "STR" -- )         \ list all matching words in all wordlists
85    target                            \ parse string to here
86    ['] .voc-mword map-vocs ;         \ map-vocs will excute .voc-mword
87
88  previous                            \ restore original search order (helper words become invisible)
89
90
91
92  \ ********** primary versions with minor formatting ******************************
93  \\\
94  : .voc-mword ( wid -- )
95    dup
96    cr ." Vocabulary: " .voc
97    25 swap ['] .mword map-wordlist drop ;
98
99  : voc-mwords ( / "STR" -- )
100   bl word count 2dup UPPERCASE here place
101   ['] .voc-mword map-vocs ;
```

Forth–2012: Der neue Standard

M. Anton Ertl

Bernd Paysan hat den neuen Standard Forth–2012 bereits in Heft 3–4/2014 vorgestellt, allerdings vor allem als Aufzählung von neuen Wörtern. In diesem Artikel beschreibe ich, wozu diese Neuerungen gut sind.

Was ist ein Standard?

Es gibt eine Reihe von Missverständnissen über die Rolle von Standards.

Ein Standard für eine Programmiersprache beschreibt eine Schnittstelle zwischen Programmen in dieser Programmiersprache und Implementierungen (Compiler, Interpreter) dieser Sprache. Ein Standard–Programm läuft also auf einem Standard–System. Im Forth–Standard haben wir optionale Wörter, was die Sache etwas komplizierter macht: Ein Standard–Programm, das eine bestimmte Menge optionaler Wörter verwendet, läuft auf Standard–Systemen, die mindestens diese optionalen Wörter implementieren. Zum Glück stellen die meisten populären Forth–Systeme fast alle optionalen Wörter zur Verfügung (manche aber erst durch das explizite Nachladen gewisser Dateien).

Was kann der Standard und was nicht?

Unter den Standards gibt es welche, die spezifizieren wollen, was in Programmen und Systemen üblich ist (z.B. der Forth–Standard), aber auch solche, die den Stand der Technik weiterentwickeln wollen (z.B. WLAN–Standards). Da der Forth–Standard zur ersteren Art zählt, ist er dementsprechend nicht der Platz, wo Innovation und neue Features zu finden sind. Wenn man ein neues Feature im Standard haben will, ist der Weg, es in einem System zu implementieren, dafür zu werben, dass Programmierer es verwenden, und andere Systeme es auch implementieren, und bei Erfolg dieser Bemühungen ist das Feature dann irgendwann üblich und kann standardisiert werden.

Bestimmte Features sind in verschiedenen Systemen verschieden implementiert. Mit Glück kommt im Standardisierungskomitee eine Einigung auf eine gemeinsame Variante zustande (wobei die Systeme zusätzlich ihre ursprünglichen Versionen implementieren), manchmal aber auch nicht. Und in jedem Fall muss sich auch noch jemand finden, der einen Vorschlag zur Standardisierung macht und den (relativ aufwändigen) Prozess dazu durchzieht.

Im Ergebnis enthält der Standard nur einen Teil der Features, die man sich als Anwendungsprogrammierer wünschen würde. Dann kann man systemspezifische Erweiterungen verwenden (eventuell in abgegrenzten Bereichen des Programms), aber bindet sich damit an ein System; in einigen Fällen kann man auch um die Abwesenheit eines Features herumprogrammieren; die Frage ist, ob sich das auszahlt.

Auf der anderen Seite gibt es immer wieder Kritik, dass der Standard für kleine Systeme und minimalistische Ansprüche zu groß ist. Programme für kleine Systeme weichen häufig vom Standard ab, sodass der Vorteil des Standards dabei wohl vor allem in einem gemeinsamen Grundvokabular besteht, weniger in der Portabilität von Programmen. Für minimalistische Ansprüche ist der Standard (und vermutlich jeder andere mögliche) wohl ungeeignet.

Was ist Forth–2012?

Forth–2012 ist der neue Standard. Er ist eine Weiterentwicklung des als ANS–Forth bekannten Standards von 1994 (im folgenden Forth–94). Dabei haben wir großen Wert auf Kompatibilität gelegt, sodaß Forth–94–Programme auch Forth–2012–Programme sind.[1] Forth–2012 ist daher nicht revolutionär, sondern evolutionär.

Das Standard–Dokument gibt es einerseits in druckbarer Form[2]. Die Änderungen im Vergleich zu Forth–94 sind im Anhang C.7 aufgeführt, werden aber im Rest dieses Artikels vielleicht etwas verständlicher erklärt.

Andererseits ist aber oft die Online–Version[3] praktischer als ein Ausdruck. Dort kann man auch über Wörter und andere Teile des Standards diskutieren.

Entfernte Wörter und Features

Eine Reihe von Wörtern wurden in Forth–94 abgekündigt und die meisten von ihnen wurden in Forth–2012 dann auch entfernt:

- `Tib #tib` wurden entfernt, dafür gibt es seit Forth–94 `source`.
- `Convert` wurde entfernt, dafür gibt es seit Forth–94 `>number`.
- `expect` und `span` wurden entfernt. Ersatz: `accept`
- `query` wurde entfernt. Ersatz: `refill`.
- `Word` hinterlässt jetzt keinen Space mehr hinter dem Wort.

[1] mit Ausnahme der Programme, die Wörter und Features benutzen, die schon in Forth–94 abgekündigt (*obsolescent*) waren und in Forth–2012 entfernt wurden.

[2] `http://www.forth200x.org/documents/forth-2012.pdf`

[3] `http://forth-standard.org/standard/words`

- `Forget` wurde zwar auch in Forth–94 abgekündigt, aber hat noch soviele Anhänger, dass es nicht aus Forth–2012 entfernt wurde.

Forth–94 hat Abfragen von wordsets mit `environment?` eingeführt (z.B. `s" string" environment?` zur Abfrage, ob das wordset vorhanden ist. Dieser Mechanismus war allerdings wenig populär und hätte für Forth–2012 erweitert werden müssen (damit man auch die Forth–2012–Varianten der Wordsets abfragen kann). Wir haben uns daher entschlossen, keine Forth–2012–Variante dieser Abfragen hinzuzufügen, und haben die Forth–94–Varianten abgekündigt; sie werden im nächsten Standard entfernt. Stattdessen gibt es jetzt:

[Defined]

Mit `[defined]` *`word`* bzw. `[undefined]` *`word`* kann man abfragen, ob ein Wort vorhanden ist. Wenn man nach einem Standard–Wort fragt, und es wird gefunden, gibt es zwar (ausser für `core`–Wörter) keine Garantie, dass es sich so verhält wie im Standard beschrieben, aber im Normalfall reicht das aus.

Jenseits von ASCII — XChars

Für Zeichensätze mit mehr als 256 Zeichen gibt es weitgehend ASCII–kompatible Kodierungen, die aus 8–bit–Elementen aufgebaut sind, wobei ein Zeichen dann aus einem oder mehreren 8–bit–Elementen bestehen kann. Am bekanntesten ist die UTF–8–Kodierung von Unicode, dem universellen Zeichensatz, aber auch z.B. die Kodierungen Big5 und GB2312 für chinesische Zeichen. Im Folgenden konzentriere ich mich aber auf UTF–8.

Das Schöne an den ASCII–kompatiblen Kodierungen ist, dass aufgrund der Kompatibilität der meiste Code ohne Änderungen damit funktioniert. Meistens wird ohnehin mit Strings hantiert, und wenn der String einfach nur eingelesen und dann wieder ausgegeben wird, oder schlimmstenfalls zusammengehängt, ist es egal, wie die Zeichen in den Strings kodiert sind.

So funktionieren auch Forth–Systeme wie Gforth 0.6.2 (aus 2003), die ohne Rücksicht auf UTF–8 unter der Annahme von 8–bit–codierten Zeichen geschrieben wurden, recht gut mit UTF–8; nur der Kommandozeilen–Editor offenbart ein paar Schwächen, und die Anzeige von Fehlermeldungen ist auch suboptimal.

In vielen Fällen verwendet man daher bevorzugt Wörter, die Strings verarbeiten. Wenn man z.B. ein Unicode–Zeichen zweimal ausgeben will, kann man das mit

```
s" Δ" 2dup type type
```

machen, auch wenn man mit dem Xchar–Wordset auch

```
char Δ dup xemit xemit
```

schreiben könnte. Oder wenn man in einem String nach einem UTF–8–codierten Zeichen sucht, geht das z.B. so:

```
buf len s" Δ" search
```

Wobei `search` schon seit Forth–94 standardisiert ist, das nach einem einzelnen Byte suchende `scan` dagegen nicht.

Da alle Strings in einem Forth–System mit UTF–8 kodiert werden können, kann man auch in Wortnamen Unicode–Zeichen verwenden, z.B.:

```
variable Δ
1 Δ !
1 Δ +!
Δ @ .
```

Für die wenigen Fälle, in denen man auf einzelne Zeichen zugreifen muss, stellt Forth–2012 das Xchar–wordset zur Verfügung. Diese Wörter können für verschiedene Kodierungen verwendet werden, sowohl für die klassische 8–bit–Kodierung, also auch für UTF–8, oder die oben erwähnten älteren Kodierungen für chinesische Zeichen, wenn das Forth–System diese Codierungen unterstützt. Derzeit unterstützt Gforth 8–bit–Codierungen (z.B. ASCII oder Latin–1) und UTF–8.

Forth–2012 garantiert allerdings nicht, dass ein bestimmtes System, selbst wenn es das Xchar–Wordset hat, tatsächlich mit UTF–8 umgehen kann. Allerdings zeigen einige Tests mit SwiftForth und VFX Forth, dass diese Systeme UTF–8 verarbeiten können, auch wenn sich bei meinen Tests Schwächen mit dem Kommandozeileneditor und den Fehlermeldungen offenbarten (wobei ich nicht ausschließen kann, dass diese Systeme mit entsprechender Konfiguration eine noch bessere UTF–8–Unterstützung bieten).

Das größere Problem in der Praxis liegt außerhalb der Forth–Systeme: Wie bekommt man den Terminal–Emulator bzw. die Console, in der das Forth–System läuft, dazu, UTF–8–Ausgaben auch wie gewünscht anzuzeigen, und wie gibt man solche Zeichen ein? Dafür gibt es zwar Lösungen, aber zumindest ich muss da noch öfters herumfrickeln.

Records

Man kann zwar auch in Forth–94 relativ leicht Records/Strukturen selbst definieren, aber da das ein häufig benötigtes Feature ist, haben wir es in Forth–2012 standardisiert. Ein Beispiel ist:

```
begin-structure flist
  field:  flist-next
  ffield: flist-val
end-structure

falign here flist allot constant flist1
0 flist1 flist-next !
1.23e flist1 flist-val f!
```

In diesem Beispiel haben wir eine Struktur `flist` (für einen Knoten einer verketteten Liste von Gleitkommazahlen) mit zwei Feldern: Einem Zellen–Feld `flist-next`, und einem Gleitkommazahlen–Feld `flist-val`.

Danach kommt die Definition eines Exemplars `field1` dieser Struktur.[4] Schließlich werden die Felder von `field1` initialisiert.

Es gibt weitere Wörter zur Definition von Feldern mit verschiedenen Größen und Ausrichtungen: `cfield:`, `sffield:`, `dffield:` und auch ein Basiswort `+field`. Außerdem kann man Strukturen auch ohne `begin-structure` und `end-structure` definieren. Man kann die obige Definition von `flist` daher auch ausschließlich mit `+field` definieren:

```
0
  1 cells +field flist-next
  faligned 1 floats +field flist-val
constant flist
```

Zahlen mit expliziter Basis

Statt `base` zu ändern, kann man die Basis einer Zahl jetzt einfach durch einen Prefix angeben:

```
$ff \ hex
#99 \ Dezimal
%11 \ Binaer
'a' \ Zeichen
$-ff. \ hex double
```

Wie man an den Beispielen sieht, kann man neben Zahlen auch Zeichen so eingeben; die Forth–94–Methode mit `char` bzw. `[char]` funktioniert weiterhin.

Funktionstasten

Schon Forth–94 stellt `ekey` zur Verfügung, um Sondertasten und andere Nicht–Zeichen–Eingaben verarbeiten zu können. Allerdings kam bei `ekey` etwas nicht weiter Spezifiziertes heraus, das man daher in einem portablen Programm nicht verwenden konnte. Forth–2012 fügt daher Wörter für Funktionstasten und Cursortasten hinzu, die man auch noch mit Modifikationstasten (Shift, Ctrl, Alt) kombinieren kann. Beispiel:

```
... ekey ekey>fkey if
 case
  k-up                    of ... endof
  k-f1                    of ... endof
  k-left k-shift-mask or
         k-ctrl-mask  or of ... endof
  ...
 endcase
else
 ...
then
```

Das Ergebnis von `ekey` wird zunächst mit `ekey>fkey` als Sondertaste identifiziert (oder nicht, deswegen das `if`), und wenn es eine ist, wird aus dem `ekey`-code ein fkey-code[5] erzeugt. Der kann dann mit Tastencodes wie `k-up` (Cursor–up–Taste) oder `k-f1` verglichen werden. Man kann auch mit Tastenkombinationen wie Shift–Ctrl–Left vergleichen.

Allerdings gibt es keine Garantie, dass jedes System einen bestimmten Tastencode auch mit `ekey ekey>fkey` erzeugen kann, daher sollte man das Programm so schreiben, dass jede Funktion auch ohne die Sondertasten erreichbar ist.

Lokale Variablen

In Forth–94 gibt es eine Syntax für lokale Variablen (locals), die aber von vielen Programmierern nicht akzeptiert wurde, weil die Reihenfolge der locals in dieser Syntax im Vergleich zu Stack–Kommentaren verkehrt herum ist.

Stattdessen verwendeten viele die Syntax `{ a b c }`, wobei `c` mit dem top–of–stack initialisiert wird. Allerdings wird in SwiftForth `{...}` für Kommentare verwendet, sodass letztendlich `{: a b c :}` als Syntax standardisiert wurde. Ein kleines Beispiel, bei dem die Reihenfolge der Elemente eine Rolle spielt:

```
: swap ( a b -- b a ) {: a b :} b a ;
```

Da locals oft am Anfang einer Colon–Definition definiert werden, und dann oft alle Stack–Elemente in locals verfrachtet werden, wie im `swap`–Beispiel oben, wurde die locals–Definition so erweitert, dass sie den Stack–Effekt–Kommentar ersetzen kann:

```
: swap {: a b -- b a :} b a ;
```

Dabei wird der Bereich von `--` bis (exklusive) `:}` ignoriert und ist nur Kommentar.

Einige Systeme erlauben nur eine locals–Definition pro Colon–Definition. Daher will man auf diesen Systemen u.U. locals definieren, bevor man weiß, mit welchem Wert man sie initialisiert. Daher sieht der Standard auch die Möglichkeit vor, uninitialisierte locals zu definieren:

```
: foo {: a b | c d -- e f :} ... ;
```

Die Namen zwischen `|` und `--` bzw. `:}`, hier also `c d`, sind uninitialisierte locals, die am Anfang einen beliebigen Wert haben können.

Das Wort `locals|` für die alte locals–Syntax wird in Forth–2012 abgekündigt und wird voraussichtlich im nächsten Standard nicht mehr vorhanden sein.

Laden von Dateien

Mit `require` *`file`* bzw. `required ( c-addr u -- )` kann man jetzt eine Forth–Quellcode–Datei einmalig laden. Das ist sinnvoll, wenn z.B. die Bibliotheken `a.4th` und `b.4th` unabhängig voneinander die Bibliothek `c.4th` laden, und ein Programm `app.4th` dann sowohl `a.4th` als auch `b.4th` lädt. Dann lädt `require c.4th` nur beim

[4] Falls sich jemand über die Konstruktion mit `here...constant` statt `create` wundert: `create` garantiert keine Ausrichtung für Gleitkommazahlen (`falign`) und kann daher hier nicht sinnvoll benutzt werden.

[5] Diese Komplikation ist für Systeme mit komplizierteren `ekey`-Implementierungen gedacht

ersten Mal, während `include c.4th` mehrmals laden würde.

Bei der Anwendung von `require` sollte das Laden der Datei allerdings einen neutralen Stack–Effekt haben, z.B. `( -- )` oder `( x y -- x y )`, da sonst der Stack–Effekt beim Nicht–Laden von dem beim Laden abweicht.

Weiters wurde das weit verbreitete Wort `include` endlich standardisiert.

Defer

Mit

```
defer foo ( n -- )
```

kann man einen Platzhalter `foo` definieren, und ihn dann mit dem *execution token (xt)* eines beliebigen Wortes füllen, z.B.:

```
' . is foo
```

Wenn man `foo` aufruft, wird bis auf weiteres `.` ausgeführt. Mit weiteren Aufrufen von `is` kann man das Verhalten von `foo` auch wieder ändern. Sinnvollerweise überlegt man sich für jeden Platzhalter einen Stack–Effekt und dokumentiert ihn, und füllt den Platzhalter nur mit Wörtern, die diesen Stack–Effekt haben.

Eine Anwendung davon ist indirekte Rekursion:

```
defer x ( n1 -- n2 )

: y ( n1 n2 -- n3 )
  ... x ... ;

: real-x ( n1 -- n2 )
  ... y ... ;

' real-x is x
```

Eine andere Anwendung sind Hooks, die man umdefinieren kann. So ist z.B. `type` in Gforth und VFX Forth ein deferred word, sodass man z.B. die Ausgabe umleiten oder abdrehen kann.

Man kann das xt aus einem deferred word mit `action-of` wieder herausbekommen, z.B. um `type` zeitweise umzuleiten, und danach den ursprünglichen Zustand wiederherzustellen:

```
action-of type ( old-xt )
' 2drop is type
... \ code mit abgeschalteter Ausgabe
( old-xt ) is type \ Wiederherstellung
```

Weiters gibt es `defer! ( xt1 xt2 -- )`, eine nicht–parsende Variante von `is`, und `defer@`, eine nicht-parsende Variante von `action-of`.

Reflexion über Wordlists

In Forth konnte man von Anfang an in das System Einblick nehmen und auf diverse System–Datenstrukturen zugreifen. Diese Fähigkeiten wurden aber nie in Form von Wörtern standardisiert, sondern bestenfalls die internen Datenstrukturen spezifiziert (z.B. threaded code in Forth–83).

Forth–94 hatte explizit das Ziel, die Implementierung nicht zu spezifizieren, wodurch die Möglichkeiten, die man davor z.B. durch das Wissen über threaded code hatte, für Standard–Programme wegfielen; nur für wenige von diesen Möglichkeiten wurden Wörter eingeführt, z.B. `compile,` als Ersatz für das compilieren mit `,` in Forth–83.

In der Zwischenzeit haben die Entwickler anderer Programmiersprachen die Vorteile von Reflexion erkannt[6], und stellen dafür standardisierte Schnittstellen zur Verfügung, und haben damit Standard–Forth in diesem Bereich überholt.

In Forth–2012 bekamen wir wenigstens ein bisschen Reflexion standardisiert: Wir können jetzt die einzelnen Wörter einer Wordlist herausbekommen, und wichtige Informationen über Wörter.

Dazu haben wir einen neuen zellengroßen abstrakten Datentypen, der ein benanntes Wort identifiziert, das *name token (nt)*. In einem traditionellen System kann z.B. die NFA oder die CFA als nt dienen.

Mit `traverse-wordlist` kann man über alle Wörter einer Wordlist iterieren, und bekommt dabei die nts der einzelnen Wörter heraus. Im folgenden Beispiel werden die Wörter nur gezählt und die nts nicht weiter verwendet:

```
: helper1 ( n1 nt -- n2 f ) drop 1+ true ;
0 ' helper1 forth-wordlist traverse-wordlist .
```

Man muss `traverse-wordlist` das xt eines Wortes übergeben, das für jedes Wort in der Wordlist einmal aufgerufen wird, hier `helper1`. Davor wird noch mit `0` der anfängliche Zählerstand auf den Stack gelegt. `Helper1` wirft das nt weg, erhöht den Zählerstand `n1` mit `1+`, und legt noch `true` auf den Stack (bei `false` würde `traverse-wordlist` nicht weiteriterieren). Am Ende bleibt der endgültige Zähler auf dem Stack und wird mit `.` ausgegeben.

Mit dem nt kann man den Namen des Wortes (mit `name>string ( nt -- c-addr u )`), die interpretation semantics (`name>interpret ( nt -- xt )`) und die compilation semantics (`name>compile ( nt -- x xt )`, wobei `name>compile execute` die compilation semantics des Wortes ausführt) herausfinden. Ein Beispiel, wie diese Wörter mit `traverse-wordlist` zusammenspielen:

```
: helper2 ( nt -- f ) name>string cr type true ;
' helper2 get-current traverse-wordlist
```

[6] https://de.wikipedia.org/wiki/Reflexion_%28Programmierung%29

Dieses kleine Programm ist ähnlich `words`, gibt aber die `current`-Wordliste aus. Ein größeres Beispiel ist in dem Artikel über `mwords` zu finden.

Parse-name

`Parse-name ( -- c-addr u )` wird üblicherweise statt `bl word count` verwendet. Die von `Parse-name` zurückgegebene String–Beschreibung beschreibt den String im Input–Buffer, die Beschreibung kann also bis zum nächsten `refill` verwendet werden:

```
parse-name foo parse-name bar type type
```

gibt `barfoo` aus.

IORs werfen

Viele Wörter (z.B. `open-file`) geben als Fehlerindikator einen I/O Result (IOR) zurück. Es ist üblich, diese IORs als `throw`–code zu benutzen, aber Forth–94 garantiert keinen besonderen Zusammenhang zwischen IORs und `throw`, abgesehen davon, dass bei fehlerfreier Ausführung IOR=0 geliefert wird und `0 throw` nichts bewirkt. In Forth–2012 werden die IORs jetzt explizit als gültige `throw`–codes genannt, und Forth–Systeme sollten jetzt sinnvolle Fehlermeldungen ausgeben, wenn sie solche Fehler–`throw`s fangen; z.B. gibt Gforth bei `open-file throw` im Fehlerfall die Betriebssystemmeldung aus, z.B. "No such file or directory".

Steuerzeichen in Strings

Mit `s"` kann man keine Strings mit Steuerzeichen oder `"` definieren. Deshalb gibt es jetzt das Wort `s\"`, das das erlaubt. Steuerzeichen werden ähnlich wie in C spezifiziert:

```
s\" abc \"def\nghi\tjkl"
```

Synonyme

Synonym erlaubt es, einen neuen Namen (bzw. auch den gleichen Namen in einer anderen Wordlist) für ein existierendes Wort zu definieren, z.B.:

```
synonym endif then
```

Warum nicht `alias`? Weil das besonders für immediate–Wörter auf verschiedenen Systemen Verschiedenes macht.

Fvalue 2value

Es gibt jetzt eine double– und eine FP–Variante von `value`; in allen Fällen wird mit `to` der Wert verändert:

```
5. 2value d 6. to d
5e fvalue f 6e to f
```

Buffer:

Mit `buffer:` kann man einen uninitialisierten Puffer definieren:

```
500 cells buffer: b
b 500 cells erase
```

N>r nr>

Mit `n>r` und `nr>` kann man n Zellen auf den Return–Stack legen bzw. von dort holen; diese Wörter können im Zusammenhang mit Wörtern wie `get-order` sinnvoll sein, die n Zellen und den Zähler n auf den Datenstack legen:

```
get-order n>r
nr> set-order
```

Textersetzung

Es gibt jetzt eine Makro–Textersetzung mit `replaces` und `substitute`, gedacht besonders für Internationalisierung:

```
: datum1 s" 2015-04-11" ;
datum1 s" date" replaces
: zeit1 s" 21:00" ;
zeit1 s" time" replaces
s" Um %time% am %date% ..."
  pad 100 substitute throw type
\ ergibt "Um 21:00 am 2015-04-11 ..."
s" On %date% at %time% ..."
  pad 100 substitute throw type
\ ergibt "On 2015-04-11 at 21:00 ..."
```

Damit man bei einem übergebenen String, der zufällig eine Makro–Sequenz enthält, die man nicht ersetzt haben möchte, auch keine Ersetzung bekommt, kann man den Text zuerst durch `unescape` entschärfen lassen:

```
500 buffer: buf
s" zufälliges %date% im Text"
  buf unescape ( c-addr u )
pad 100 substitute throw type
\ ergibt "zufälliges %date% im Text"
```

Gleitkomma–Arithmetik

In Forth–94 konnten Gleitkommazahlen (floating point numbers, FP) auf dem Datenstack oder auf einem eigenen Gleitkomma–Stack abgelegt werden, und Standard–Programme mussten so geschrieben werden, dass sie für beides funktionieren, was so umständlich ist, dass es kaum jemand gemacht hat, und die allermeisten Programmierer für einen separaten FP–Stack geschrieben haben. In Forth–2012 ist jetzt der separate Gleitkommastack standardisiert.

Es gibt jetzt `s>f f>s` zum Konvertieren zwischen einfachgenauen ganzen Zahlen und Gleitkommazahlen. Mit `ftrunc` kann man jetzt zur nächsten ganzen Zahl in Richtung 0 runden (das Ergebnis ist aber immer noch eine

Gleitkomma–Zahl). Weiters wurden die Wörter `fasinh` `fatan2` genauer spezifiziert.

Ausblick

Auch wenn wir mit Forth–2012 einen Meilenstein geschafft haben, geht die Arbeit am Standard weiter.

In Forth–2012 wurden `locals|` und `[compile]` abgekündigt und werden voraussichtlich mit der nächsten Version aus dem Standard verschwinden. Stattdessen sollte man `{:` bzw. `postpone` benutzen.

Bei der letzten Standardisierungssitzung 2015 haben wir beschlossen, dass Forth–Systeme negative ganze Zahlen als 2er–Komplement darstellen und für den Überlauf die übliche Modulo–Arithmetik verwenden, sodass jetzt z.B. übliche Hash–Funktionen effizient in Standard–Forth ausgedrückt werden können.

In eine ähnliche Richtung geht der Vorschlag, dass das Ergebnis von `1 chars` gleich 1 sein soll; aber das Komitee konnte sich zu diesem Thema noch zu keinem Konsens durchringen.

Es liegen noch weitere Vorschläge vor, die allerdings kontroverser sind und wohl einige Jahre benötigen werden, bis es einen Konsens für sie gibt:

Quotations erlauben es, Hilfsdefinitionen wie sie für `catch` oder `traverse-wordlist` benötigt werden, direkt und namenlos innerhalb einer Definition zu schreiben. Beispiel:

```
: count-words ( wid -- n )
  0 [: drop 1+ true ;] rot traverse-wordlist ;
```

Recognizer erweitern den Text–Interpreter um die Erkennung beliebiger Wörter. Beispiele für Recognizer sind die Zahlenerkennung und –konversion in aktuellen Forth–Systemen; diese sind aber derzeit nicht portabel erweiterbar, sondern nur über systemspezifische Hooks. Standardisierte Recognizer würden das ändern.

Weitere geplante Themen sind Wörter für 16–bit–Speicherzugriffe u.ä., multi–tasking und multi–threading, und das C–Interface.

Link

`http://www.complang.tuwien.ac.at/anton/home.html`

Macht mit!

Zu viele Köche verderben den Brei und oft auch die Software, aber bei Standards ist das anders: Der Standard soll ja das widerspiegeln, was in der Community üblich ist, und um das herauszufinden, brauchen wir Input von Euch, der Forth–Community. Auch die Probleme, die Ihr habt, und für die der aktuelle Standard keine Lösungen bietet, aber Eurer Meinung nach bieten sollte, solltet Ihr an uns herantragen (auch wenn dann in vielen Fällen eine Erklärung zurückkommen wird, wie das jetzt schon geht, oder warum wir das jetzt nicht angehen).

Wie könnt Ihr beitragen? Ihr könnt in der Newsgroup `comp.lang.forth` oder in der Mailingliste[7] über Vorschläge (RfDs) mitdiskutieren und Euch bei der Umfrage über die fertigen Vorschläge (CfVs) beteiligen.

Wenn wir einen Release Candidate des Standard–Dokuments haben (bisher einmal in 11 Jahren), könnt Ihr Euch an der öffentlichen Begutachtung beteiligen. Dabei sind das letzte Mal Fehler gefunden und beseitigt worden; für inhaltliche Änderungen ist es aber sinnvoller, sich schon in der RfD/CfV–Phase zu beteiligen.

Wenn Ihr Euch stärker beteiligen wollt, könnt Ihr selbst Vorschläge (RfDs) machen. Seid darauf gefasst, dass dabei immer von irgendjemandem Gegenwind kommt, egal wie üblich das Vorgeschlagene ist, und lasst Euch davon nicht entmutigen; wenn der Vorschlag umgekehrt gar keine Unterstützung erfährt, dürft Ihr Euch schon entmutigen lassen. In jedem Fall ist mit dem Formulieren des Vorschlags und seiner Revisionen, und der Behandlung der Kommentare ein gewisser Zeitaufwand verbunden.

Wenn Ihr Euch noch stärker beteiligen wollt, kommt zum Standardisierungstreffen (in den letzten Jahren die zwei Tage vor der EuroForth). Ihr erlebt dann intensive Diskussionen über Vorschläge, Verfahrensfragen, Kleinigkeiten, und künftige Entwicklungen, und könnt auch mitreden. Bei Eurer zweiten Teilnahme hintereinander dürft Ihr auch mit abstimmen, mit Zustimmung des Komitees auch beim ersten Mal.

[7] `https://groups.yahoo.com/neo/groups/forth200x/info`

Ausführbare Ketten

Matthias Trute

Win32Forth hat ein nettes Konzept, genannt Ketten. Das sind Listen von Execution Tokens, die en block ausgeführt werden. Gewissermaßen werden Ideen von DEFER und Listen nahe beieinandergelegt. Nachfolgend sollen beide Ansätze vereint werden: Der Name einer Kette wird wie bei einem Defer ein ausführbares Kommando. Ruft man es auf, wird die gesamte Kette aufgerufen, indem alle Elemente der Reihe nach ausgeführt werden.

Motivation

Bei den Recherchen zu den Recognizern fand ich so manches Schöne und Nützliche. Ein Beispiel sind Chains, die in Win32Forth eine wichtige Rolle spielen.

Grundlagen

Zunächst, was ist eine Chain. Deutlich wird dies am Quellcode der Kernroutine

```
: do-chain ( chain_address -- )
    begin   @ ?dup
    while   dup>r
            cell+ perform
            r>
    repeat  ;
```

Irgendwo, adressiert durch den Parameter `chain_address`, fängt eine Liste von Execution Tokens an. Diese werden der Reihe nach ausgeführt, bis ein Null-Token erreicht wird. Diese Liste wird im Dictionary verankert und besteht aus zwei Angaben pro Element: Dem XT und einem Link zum Folgeelement. Die üblichen Methoden zur Listenverwaltung sind entsprechend vorhanden, folgen jedoch nicht so ganz den zum Beispiel Wirth'schen Konventionen[1], Forth halt.

Innerhalb von Win32Forth gibt es eine Reihe von derartigen Ketten, die verschiedene Aufgaben erfüllen: `initialization-chain`, `reset-stack-chain`, `forth-msg-chain` oder auch `forth-io-chain`. Da wir es mit einem Windowssystem zu tun haben, ist die `mouse-chain` nicht unerwartet. Aufgerufen werden die Chains über ihren Namen und eine Aktion, die ausgeführt werden soll; exemplarisch die `initialization-chain`:

```
: Temp-HELLO
  (boot)
  only forth also definitions
  initialization-chain do-chain
;

' Temp-HELLO is boot
```

Das heißt, alles, was in dieser Kette verankert ist, wird beim Start des Programms ausgeführt. Es finden sich knapp zwei Dutzend Stellen im Quellcode, die ihr etwas hinzufügen. Es gibt auch eine `number?-chain`, die auf Strings losgelassen wird, um sie in eine Zahl zu verwandeln.

Allen Ketten ist gemeinsam, dass sie entweder komplett durchlaufen werden oder eine Exception werfen, die dann irgendwer auffangen kann.

Implementierung

Die Umsetzung im Win32Forth ist über viele Dateien verteilt und nicht unbedingt einfach nachvollziehbar. Außerdem habe ich, vielleicht auch bedingt durch anfängliches Fehlverständnis, einige Dinge anders haben wollen. Der erste Punkt ist, dass es keinen einfach wiederzuverwendenden Code für eine verlinkte Liste im Kernsystem von amforth gibt. Eine Wordlist ist zwar nichts anderes, aber der Programmcode ist doch schon sehr spezialisiert. Zum anderen wollte ich die Ketten ähnlich zu den Defer haben, so dass sie über ihren Namen auch gleich ausführbar sind. Bei Win32Forth sind Worte wie `do-chain` oder verwandte Befehle wie `add-chain` immer dabei.

Während die Recognizer zunächst wichtiger waren, geisterte die Idee der Ketten immer irgendwie im Hintergrund mit. Überraschenderweise gab es letztendlich zwei Wege, wie sie umgesetzt wurden: Als Wordlist und als Stack. Beide haben Vor– und Nachteile, so dass es durchaus sinnvoll erscheint, auch beide zu haben. Gemeinsam ist, dass sie die auf den vorhandenen Möglichkeiten aufbauen und dadurch sehr kompakt sind.

Geblieben sind die Exceptions, falls etwas schief läuft. Amforth hat die ohnehin schon immer gehabt, da erschien es sinnvoll, dies beizubehalten. Die verwendeten Implementierungen erlauben es jedoch, auf Exceptions zu verzichten und mit Flags zu arbeiten.

Zuerst die Stackvariante. Sie basiert auf den gleichen Bauteilen wie die Search–Order– oder Recognizer–Stacks. Kernpunkte sind, dass die Daten der Kette im EEPROM liegen[2] und dass die maximale Größe der Kette a priori festgelegt wird.

```
\ #require quotations.frt
\ #require builds.frt
\ #require eallot.frt

: chain ( n -- )
  ehere swap
  1+ cells eallot ;
```

[1] Algorithmen und Datenstrukturen, Niklaus Wirth 1975. Viel einfacher als TAOCP von Donald Knuth.

[2] Beim MSP430 ist hier der Info–Flash im Einsatz.

```
: chain.run ( chainid -- i*x )
  [: ( i*x XT -- j*y 0 )
    execute 0
  ;] swap map-stack ( -- 0 )
  drop ;

: chain: ( n "name" -- )
  <builds  ( or create )
    0 chain dup , !e
  does>
   @i chain.run ;

: chain>id ( "name" -- chainid )
  ' >body @i ;
```

Warum `<builds` und nicht `create`? Die MSP430 Systeme sind beim Flash wirklich write-once und da `Create` den Inhalt des Execution Tokens immer überschreibt, ist der Rückgriff auf `<builds` einfach nur pragmatisch.

Für die Kette selbst ist der Einsatz geradlinig: Zuerst wird eine Liste von Execution Tokens auf dem Datenstack vorbereitet und dann auf einen Schlag der Kette zugewiesen. Ganz so wie bei SET-ORDER.

```
> 4 chain: kette
ok
>  ' ver ' cr ' ver 3 chain>id kette set-stack
ok
> kette
amforth 6.3 ATmega328P
amforth 6.3 ATmega328P ok
```

Zweimal die Versionsangabe dazwischen der Zeilenvorschub.

Die Wortlistenvariante geht einen anderen Weg. Sie baut auf dem Wort `traverse-wordlist` aus dem Forth–2012–Standard auf.

```
\ #require name2interpret.frt
\ #require builds.frt
: chain wordlist ; \ trivial

: chain.run ( chainid -- i*x )
  [: name>interpret execute true ;]
  swap traverse-wordlist ;

: chain: <builds ( or create )
    0 chain dup , !e
  does>
    @i chain.run ;

: chain>id ( "name" -- chainid )
  ' >body @i ;
```

Bei der Nutzung sind die Unterschiede zur obigen Stackvariante zu sehen.

```
> chain: kette
ok
> get-current
ok
> chain>id kette set-current
ok
> : s1 ." eins " ;
ok
> : s2 ." zwei " ;
ok
> set-current
ok
> chain>id kette show-wordlist
s2 s1 ok
> kette
zwei eins  ok
```

Was sofort auffällt, ist, dass es diesmal komplette Worte sind und nicht nur die Execution Tokens. Da kann man natürlich auch Synonyme benutzen. Ebenso wird CURRENT auf eine Wortliste gesetzt, die es nie in die Search Order bringt und trotzdem Worte enthält, die ausgeführt werden.

Was unterscheidet die beiden Ansätze grundlegend? Als Erstes: Die Wortlistenvariante kann beliebig viele Einträge haben. Dafür steht die Reihenfolge bei der Ausführung fest: Das zuletzt definierte Wort wird als erstes ausgeführt. Dafür ist sehr einfach nachzuprüfen, welche Elemente in einer Kette enthalten sind. Die Stackvariante hat hingegen eine feststehende maximale Größe, dafür kann man die Reihenfolge der Elemente auch nachträglich frei ändern. Herauszufinden, welche Elemente enthalten sind, ist hingegen etwas komplexer. Da braucht es ein `SEE`.

Auf dem MSP430 sind einige Details anders. So gibt es keinen EEPROM, es wird einfach in das RAM geschrieben. Aus `eallot` wird damit einfach nur `allot`, aus `!e` wird `!`. Dafür muss man am Ende auch `SAVE` sagen, damit alles den nächsten Reset übersteht.

Ausblick

Der naheliegende Einsatz ist, wie bei Win32Forth, die Startsequenz. Amforth hat hier `turnkey` als simplen Defer, wo man etwas aufpassen muss, wenn man seine eigene Startroutinen hinzufügen will.

Daneben sind durchaus Alternativen zu den vorgestellten Worten vorstellbar. So ist `chain>id` ein parsing word. Das ist in aller Regel ok, wird aber zum Problem, wenn compiliert werden soll. Ist der Ansatz

```
> ' kette chain>id
```

vielleicht doch der Bessere? Das wäre auch deutlich näher am Vorbild aus Win32Forth.

Und natürlich: Lohnt es sich, Aufwand in das Thema Wortlisten zu stecken? Bislang kann man nur Anhängen und die Suchrichtung geht ausschließlich vom Neuesten zum Ältesten. Mit Kenntnis der internen Headerstruktur sollte es eigentlich ein Leichtes sein, die Wortreihenfolge zu ändern. Sofern das genutzte Speichermedium ein Überschreiben erlaubt. Was haben andere Programme davon?

Für eine komplette Anbindung an das DEFER–Konzept braucht es noch ein passendes IS. Das gibt der Standard jedoch nicht her, der will ausschließlich und genau ein Execution Token. Wenn man bereit ist, das erst mal zu ignorieren, kann man durchaus aufbauend auf den Ideen der Value–Manufaktur auch hier kreative Erweiterungen schaffen.

Config–Dateien parsen

Bernd Paysan

Config–Dateien sollen für Menschen lesbar sein, also eine einfache, bekannte Syntax haben, und wirklich nur Config–Variablen modifizieren können. Die Implementierung ist eine kreative Verwendung von Recognizern, nicht um den existierenden Interpreter zu erweitern, sondern um mit existierenden Recognizern einen neuen Interpreter zu bauen. Auch sonst werden Gforth–spezifische Features verwendet, die nur in der aktuellen Entwicklerversion verfügbar sind.

Einleitung

Recognizer wurden hier schon öfter diskutiert [1, 2, 3, 4]. Das Prinzip ist einfach: Ein Recognizer guckt sich ein Wort im Eingabestrom an, und entscheidet dann, ob er sich zuständig fühlt, oder nicht. Bei Literals ist der Recognizer auch für das Konvertieren in Daten auf dem Stack zuständig. Am Ende landet dann eine Typ–spezifische Tabelle auf dem Stack, und eben die Daten, die für diesen Typ gebraucht werden. Der Interpreter oder Compiler greift dann in dieser Tabelle auf die für ihn relevante Methode zu, und mit der wird dann compiliert oder interpretiert.

Was ich hier machen will, ist aber etwas anders. Config–Files kennt man von allerlei Programmen, sie sind üblicherweise ⟨*variable*⟩ = ⟨*wert*⟩–Paare. Die Variablen haben einen Typ, und der Wert darf nur von diesem Typ sein, also String, Integer oder Float. Anders als in normalem Forth soll der Typ auch wirklich geprüft werden, denn die Config–Datei wird ja von einem unbedarften User editiert. Config–Dateien können sich auch während des Programmlaufs verändern, und müssen dann wieder rausgeschrieben werden. Alle Information über die Inhalte der Config–Datei muss auch dem Programm selbst bekannt sein.

Die Benutzung soll natürlich einfach sein. Es gibt ein vordefiniertes Vokabular `config`, für die Default–Config, man kann natürlich auch andere Vokabulare verwenden, um Konfigurationen abzulegen. So sieht etwa das Config–Vokabular für net2o zur Zeit aus:

```
scope{ config
Variable rootdirs$
Variable prio#
Variable host$
Variable date#
Variable objects$
Variable chats$
Variable keys$
Variable .net2o$
}scope
```

`scope{` und `}scope` sind Kürzel für `get-current also` *`<vocabulary>`* bzw. `previous set-current`. Die Suffixe an den Variablennamen kennzeichnen ihren Typ, '$' für String, '#' für zahl, '&' für doppelt genaue Zahl, und '%' für Fließkomma. Die letzten beiden brauchen wir hier nicht.

Eine gültige Config–Datei sieht etwa so aus:

```
.net2o="~/.net2o"
keys="~/.net2o/keys"
chats="~/.net2o/chats"
objects="~/.net2o/objects"
date=2
host="daiyu"
prio=10
rootdirs="/home/bernd/net2o"
```

Was brauche ich zum Lesen dieser Datei? Eine Read–Loop, die jede Zeile interpretiert. Kern ist hier das Wort `config-line`:

```
: config-line ( -- )
  '=' parse 2>r
  parse-name config-recognizer map-recognizer
  2r> eval-config
  postpone \ ;
```

Was macht das? Es guckt erst mal nach dem '='–Zeichen, das davor ist der Variablen–Name. Alles, was nach dem '='–Zeichen kommt, wird dem Config–Recognizer–Stack zum Parsen vorgelegt. Dieser enthält die drei Recognizer `rec:num`, `rec:float` und `rec:string`. Zusammen mit dem Namen der Variable kann dann `eval-config` den Rest der Arbeit übernehmen. Der Rest der Zeile interessiert uns nicht, hier können also Kommentare stehen.

```
: eval-config ( .. rec addr u -- )  rot
  case
    r:string of
      '$' ['] $! [: drop free throw ;]
      exec-config
    endof
    r:num    of
      '#' ['] !  ['] drop exec-config
    endof
    r:dnum   of
      '&' ['] 2! ['] 2drop exec-config
    endof
    r:float  of
      '%' ['] f! ['] fdrop exec-config
    endof
    2drop ." can't parse config line: '"
    source type ." '" cr
  endcase ;
```

Da es keinen Config–Parser–Eintrag in der vom Recognizer zurückgegebenen Tabelle gibt, nutze ich ein Case–Statement, das für jeden der vier erwarteten Datentypen `exec-config` mit passenden Parametern aufruft. Dieses Wort braucht das Zeichen für den Typ–Hint am

Variablennamen, ein xt zum Speichern und eins zum Verwerfen des Werts; letzteres für den Fehlerfall. Ich will im Fehlerfall nur eine Warnung ausgeben, und diese Zeile nicht parsen. Stattdessen wird dann der Default genommen, das sollte gutmütig sein.

```
: exec-config ( .. addr u char xt1 xt2 -- )
  >r >r
  [: >r type r> emit ;] $tmp
  config-wl find-name-in
  ?dup-IF  execute r> execute rdrop
  ELSE rdrop r> execute .config-err THEN ;
```

`$tmp` baut mit Hilfe der Quotation einen konkatenierten String, vorne der Variablenname, hinten der Typ–Hint, und das suchen wir im Vokabular `config`, dessen WID im Value `config-wl` gespeichert ist (kann man dort auch ändern, wenn man ein anderes Vokabular nutzen möchte). Finden wir was, speichern wir die vom Recognizer gewandelten Daten in der Variable, finden wir nichts, verwerfen wir die Daten.

Das war's schon, zum Schreiben muss man nur mit `map-wordlist` durch das Vokabular laufen, und alle Variablen–Inhalte dem Typ entsprechend ausgeben.

Referenzen

[1] Matthias Trute, *Recognizer — Interpreter dynamisch verändern,* VD 2/2011

[2] Bernd Paysan, *Recognizer,* VD 2/2012

[3] Matthias Trute, *Recognizer — Interpreter dynamisch verändern,* VD 3–4/2014

[4] Matthias Trute, *Suchen durch Erkennen,* VD 1/2015

Listings

```
1   \ config file reader/writer
2
3   Vocabulary config     ' config >body Value config-wl
4   Variable config-recognizer
5
6   ' rec:string ' rec:float ' rec:num 3 config-recognizer deque!
7
8   : .config-err ( -- )
9       ." unknown config variable: '" source type ." '" cr ;
10  : exec-config ( .. addr u char xt1 xt2 -- ) >r >r
11      [: >r type r> emit ;] $tmp config-wl find-name-in
12      ?dup-IF  execute r> execute rdrop
13      ELSE rdrop r> execute .config-err THEN ;
14
15  : eval-config ( .. rec addr u -- )  rot
16      case
17          r:string of  '$' ['] $! [: drop free throw ;] exec-config  endof
18          r:num    of  '#' ['] !  ['] drop exec-config               endof
19          r:dnum   of  '&' ['] 2! ['] 2drop exec-config              endof
20          r:float  of  '%' ['] f! ['] fdrop exec-config              endof
21          2drop ." can't parse config line: '" source type ." '" cr
22      endcase ;
23
24  : config-line ( -- )
25      '=' parse 2>r
26      parse-name config-recognizer map-recognizer 2r> eval-config
27      postpone \ ;
28
29  : read-config-loop ( -- )
30      BEGIN  refill  WHILE  config-line  REPEAT ;
31
32  : read-config ( addr u wid -- )  to config-wl
33      >included throw ['] read-config-loop execute-parsing-named-file ;
34
35  : write-config ( addr u wid -- )  to config-wl
36      r/w create-file throw >r
37      [: config-wl
38          [: dup name>string 1- 2dup + c@ >r type .\" ="
39              execute r>
40              case
41                  '$' of  $@ [: '"' emit [ also see-voc ] c-\type [ previous ]
42                                '"' emit ;] $tmp type cr  endof
43                  '#' of  @ 0 .r cr  endof
44                  '&' of  '#' emit 2@ 0 d.r '.' emit cr  endof
45                  '%' of  f@ fe. cr  endof
46                  drop
47              endcase
48          ;] map-wordlist ;] r@ outfile-execute
49      r> close-file throw ;
```

Protokoll der Mitgliederversammlung 2016

Erich Wälde

Moderation Martin Bitter

Protokollant Erich Wälde

Teilnehmer
Direktorium: Ewald Rieger, Ulli Hoffmann, Bernd Paysan
insgesamt 17 stimmberechtigte Mitglieder (Anzahl der ausgegebenen Stimmkarten)

Sitzungsdatum 17.04.2016

Sitzungsbeginn 09:15 h

Sitzungsende 12:20 h

Sitzungsort Hochschule Augsburg, Fakultät für Informatik Gebäude J, Friedberger Straße 2a, 86161 Augsburg

Begrüßung

Ewald Rieger begrüßt im Namen des Direktoriums die anwesenden Mitglieder.

Wahl des Schriftführers

Als Protokollant wird Erich Wälde gewählt.

Wahl des Versammlungsleiters

Zum Sitzungsleiter wird Martin Bitter gewählt. Der Sitzungsleiter stellt fest, dass die Versammlung fristgerecht einberufen wurde. Es wurden 17 von 105 Stimmkarten ausgegeben. Damit sind mehr als 10 % der Mitglieder anwesend und die Versammlung ist beschlussfähig.

Ergänzungen zur Tagesordnung

Anträge zur Ergänzung der Tagesordnung liegen keine vor, die Tagesordnung wird einstimmig angenommen.

Bericht des Direktoriums

a. Bericht der Verwaltung (Ewald Rieger)

Mitgliederentwicklung Im Jahr 2015 gab es *keine* Neuzugänge und 2 Austritte (beide durch Ableben der Mitglieder). Der Verein hatte zum Jahresende 2015 104 Mitglieder. In 2016 sind 2 Austritte und 1 Zugang zu verzeichnen. Es stehen noch 6 Mitgliedsbeiträge aus.

Finanzen Ewald Rieger erläuterte im Detail die Einnahmen und Ausgaben, getrennt nach Verein und Zweckbetrieb (Vierte Dimension). Es ergibt sich ein Vermögen des Vereins von 6530 € zum Jahresende. Die vor 8 Jahren vom Finanzamt beanstandeten Rücklagen von ca. 18000 € wurden damit weisungsgemäß abgebaut. Die letzte Prüfung der Gemeinnützigkeit fand 2015 statt, ein Freistellungsbescheid für die Jahre 2012 bis 2015 zur Körperschaftssteuer und Gewerbesteuer wurde erteilt. Gleichzeitig wurde festgestellt, dass die Formulierung des Vereinszwecks in unserer Satzung nicht den geltenden Vorschriften entspricht und zur Verlängerung der Gemeinnützigkeit entsprechend den Vorgaben angepasst werden muss.

Für das Jahr 2016 sind Einnahmen von etwa 3750 € und Ausgaben in Höhe von 5135 €zu erwarten. Damit würde das Vermögen des Vereins auf 5144 € abschmelzen. Es sind ab sofort *alle* Ausgaben bei Ewald Rieger zu beantragen (gilt v.a. für die gleichberechtigt handelnden Vorstandsmitglieder). Das Vermögen würde ohne weitere Maßnahmen im Jahr 2020 auf null fallen, was die Auflösung des Vereins zur Folge hätte.

Bei der Erläuterung der Kosten wurde festgestellt, dass vier Ausgaben der Vierten Dimension inclusive Versand derzeit mit 22.83 € (Inland) bzw. 31.03 € (Ausland) zu Buche schlagen, was durch die vorgesehenen 16 € Mitgliedsbeitrag nicht mehr getragen wird.

b. Rund um das Forth–Magazin (Ulrich Hoffmann)

Die Vierte Dimension erscheint immer noch, sie ist die letzte gedruckte Veröffentlichung in Sachen Forth auf dem Planeten. Das ist bemerkenswert und gut, allerdings ist die Artikellage derzeit so schlecht wie nie. Ohne die unermüdliche Zuarbeit von Michael Kalus, Dirk Brühl und anderen, gäbe es kein Heft mehr.

James Bowman von der *Silicon Valley FIG* hat angefragt, ob er (und Mitstreiter) die Vierte Dimension denn übersetzen und veröffentlichen dürften. Dabei würde er die vorhandenen Werkzeuge (LaTeX–Template, fossil, etc.) komplett übernehmen. Erste Versuche haben bereits stattgefunden.

Die Autoren werden daher gebeten, ihre Zustimmung zur Übersetzung und zur weiteren Veröffentlichung zu geben.

Das Sonderheft ARM wurde quasi im Alleingang von Matthias Koch erstellt, mit tatkräftiger Unterstützung in Sachen Hardware von Klaus Kohl, sowie von Michael Kalus und Bernd Paysan.

Es wurde gefragt, ob man denn das Heft mit Werbung verzieren könnte und damit zusätzliche Einnahmen generieren. Es gibt selbstverständlich die Möglichkeit, als Firmenmitglied einen Platz auf der Werbeseite zu füllen.

Für weitergehende Werbung gab es keine ausdrückliche Zustimmung.

Es wird hiermit auch ausdrücklich gewünscht, dass die Vorträge zur Tagung in Textform für die Vierte Dimension eingereicht werden.

c. Internet–Präsenz (Ulrich Hoffmann)

Die Umstellung der Webseite von `geeklog` auf ein neues System geht leider nur sehr schleppend voran. Bei dem Umzug ist wohl auch das `subversion`–Repository auf `git` umzustellen. Leider ist damit auch verbunden, dass wir derzeit zwei angemietete Server bezahlen und noch nichts davon haben, dass der neue Server günstiger ist als der alte.

d. Außendarstellung und Projekte (Bernd Paysan)

Die Forth–Gesellschaft war 2015 an mehreren Veranstaltungen präsent: Maker Faire (Hannover, 2 Tage), 32C3 (Hamburg, 4 Tage, incl. Triceps, Bitkanone, Vortrag von Matthias Koch und Beteiligung von net2o/Bernd Paysan am Netzwerk Track `#ybti` bzw. `#wefixthenet`). Ein Workshop für Studierende an der Hochschule Augsburg wurde von Erich Wälde durchgeführt. Darüberhinaus fanden Einzelaktivitäten statt, wie z.B. die Portierung von `mecrisp` auf weitere boards/controller, sowie die Portierung von `AmForth` auf die MSP430 Familie.

Für 2016 ist die Beteiligung an der Maker Faire in Hannover sowie auf dem 33C3 geplant.

e. Bericht des Kassenprüfers (Thomas Prinz)

Die Prüfung der Kasse fand am 15.04.2016 durch Thomas Prinz statt. Er berichtet, dass alle Buchungen belegt und nachvollziehbar sind. Die Buchhaltung ist vorbildlich.

Entlastung des Direktoriums

Thomas Prinz stellt den Antrag das Direktorium zu entlasten.

Der Antrag wird mit 16 Ja–Stimmen und einer Enthaltung angenommen, das Direktorium ist damit entlastet.

Wahl des Direktoriums

Die bislang amtierenden Mitglieder des Direktoriums sind bereit, für eine weitere Amtszeit zu kandidieren.

Das Direktorium (Ewald Rieger, Ulli Hoffmann, Bernd Paysan) wird einstimmig für eine weitere Amtszeit gewählt. Alle Gewählten nehmen die Wahl an.

An dieser Stelle wurde die Sitzung unterbrochen. Der Tradition folgend hatte der Drachenrat getagt. Karsten Roederer übergab den Drachen an den neuen Drachenhüter Thomas Prinz. Die erste Amtshandlung des neuen Drachenhüters war die ebenso traditionelle Aufstellung zum Gruppenfoto.

Abstimmung: Satzungsänderung

Auf Anregung des Sachbearbeiters beim zuständigen Finanzamt wurde beantragt, den Wortlaut in § 2 Satz 2 geringfügig zu ändern (siehe auch Einladung).

Bisher: *Der Zweck des Vereins ist die Förderung der Kenntnis und Verbreitung der Programmiersprache Forth ...*

Neu: *Der Zweck des Vereins ist die **Förderung der Bildung**. Der Satzungszweck wird insbesondere verwirklicht durch Förderung der Kenntnis und Verbreitung der Programmiersprache Forth ...*

Der Antrag wurde einstimming angenommen.

Abstimmung: Mitgliedsbeiträge

Wie auf der Einladung erläutert, wurde der Antrag gestellt, die Mitgliedsbeiträge ab dem 1.1.2017 anzupassen:

Bisher: 16 € (ermäßigt) bzw. 32 € / Jahr

Neu: 20 € (ermäßigt) bzw. 40 € / Jahr

Der Beitrag der Fördermitglieder von 88 €/Jahr bleibt unverändert.

Diese Zahlen wurden auf der Jahresversammlung 2015 in Hannover von der Versammlung gebilligt.

Die Anpassung gleicht die Kosten des Zweckbetriebs aus. Es wird ausdrücklich darauf hingewiesen, dass die Beiträge von den Mitgliedern freiwillig erhöht werden dürfen (als Spende).

Wer keine gedruckte Version des Hefts beziehen möchte (die .pdf–Version wird ja gleichzeitig auf dem Webserver angeboten), kann dies an Ewald Rieger melden. Eine entsprechende Bemerkung wird auch in das Impressum des Hefts übernommen.

Der Antrag wurde einstimmig angenommen, die neuen Beträge gelten ab dem 1.1.2017.

Projekte

Die **Maker Faire 2016** in Hannover findet nun drei Tage (Fr.–So.) statt. Der Freitag ist explizit für Schülerbesuche vorgesehen. Wer mitmachen will (Standbesatzung), meldet sich bei Ulrich Hoffmann. Möglicherweise kann Matthias Koch eines seiner Fluoreszenz–Experimente mit Algen ausstellen.

Der Jahreskongress des Chaos Computer Clubs **33C3** findet vom 27.–30.12.2016 in Hamburg statt. Auch hier könnte man noch eine Verstärkung der Mannschaft brauchen, bitte bei Bernd Paysan melden.

Gerald Wodni hat ein neues Design der **Webseite** realisiert. Wiki und E–Mail funktionieren schon. Er bietet an, dass der Verein das übernehmen kann.

Daniel Ciesinger berichtet, dass er an seiner örtlichen **Grundschule** ein Angebot für Schüler realisieren will.

Martin Bitter berichtet, dass die Portierung von Forth auf den **Lego EV3** fortgeschritten ist und einen benutzbaren Stand erreicht hat. Er hat dieses Thema auf der Tagung als Workshop angeboten.

Verschiedenes

Für die **Tagung 2017** haben die Damen (Claudia Wodni, Elisabeth und Andrea Bitter) vorgeschlagen, die Organisation zu übernehmen. Als Veranstaltungsort wird dann wohl die Gegend um Hamminkeln (bei Wesel) in Frage kommen.

Erich Wälde fragt die Versammlung, ob man das ohne Diskussion so machen will. Die Tagung 2016 war ein Experiment, um durch die raum–zeitliche Nähe zu anderen Veranstaltungen (Linux Infotag, RETROpulsiv) an einer Hochschule möglicherweise *Laufkundschaft* zu erhalten. Daher waren alle Vorträge auch öffentlich.

Allerdings kann das Experiment als gescheitert gelten. Für den Workshop von Martin Bitter hatte sich genau ein Student angemeldet. In den Vorträgen waren zeitweise zwei Herren von der RETROpulsiv anwesend. Es wurde aber andererseits von Mitgliedern der FG etwas Zeit bei den anderen Veranstaltungen verbracht.

Es folgte eine lange Diskussion.

Möglicherweise sind andere Veranstaltungen (Easterhegg, OHM, 33C3, Chaoscamp) deswegen besser geeignet, weil das Publikum schon sehr interessiert ist. Vielleicht kommt auch der Linux Presentation Day (LPD) in Frage, der am 30.04. stattfand und danach am 22.10.2016 an vielen Orten in Europa gleichzeitig stattfindet — siehe auch die Links.

Andere Vorschläge zur Tagung 2017 wurden nicht gemacht.

Die **EuroForth 2016** findet vom 7. bis 11. September 2016 in Konstanz statt.

Schluss

Die Jahresversammlung endet um 12:20 h.

Hechingen, den 29.05.2016

gez. Erich Wälde

Links

- Easterhegg, über Ostern, wie der Name sagt, zuletzt 2016 in Salzburg, jährlich.
  `http://www.easterhegg.eu/`
- OHM: Observe Hack Make, leider unklar ob/wann die wieder stattfindet.
  `https://ohm2013.org/`
- Chaos Communication Camp, zuletzt 2015 in Mildenberg, alle 4 Jahre.
  `https://de.wikipedia.org/wiki/Chaos_Communication_Camp`
- Linux Presentation Day
  `http://www.linux-presentation-day.de/`
- Chaos Communication Congress, jährlich.
  `https://de.wikipedia.org/wiki/Chaos_Communication_Congress`,
  `https://events.ccc.de/2015/11/04/32c3-gated-communities/`

Wie die niederländische Forth–Benutzer–Gruppe zu ihrem Maskottchen kam.

Im Jahre 1997 verfassten Mitglieder der damaligen Forth–gebruikersgroup (der heutige HCC!) eine Arbeitsanleitung für den Umgang mit dem AT89C2051 Microcontroller. Vorbilder waren damals die *BASIC–Stamps*, kleine Platinen in Briefmarkengröße mit Mikroprozessoren drauf, die in BASIC programmierbar waren. Das sollte natürlich auch in Forth gehen. Das Buch mit der Arbeitsanleitung dafür bekam den Spitznamen *Postzegelboek*, niederländisch für „Briefmarken–Heft“. Doch man fand das zu lang, und so wurde der Name auf *Egelboek* gekürzt. Was nachvollziehbar ist, wenn man „Postzegel“ holländisch ausspricht: „poss–say–ghull“. Und von „say–ghull“ zu „ay–ghull“, niederländisch *egel*, der Igel, ist es nicht mehr weit. Seither ist der Igel das Maskottchen der Gruppe und aus dem *Egel–Werkboek* wurde jetzt das *Egel–Projekt*, eine umfangreiche Experimentier–Sammlung, um Mikrocomputer kennen zu lernen. Das Projekt wurde neulich noch um eine eigene Platine erweitert, das *Egel–Kit*. Damit ist es noch einfacher, die Hardware der MCUs zu erkunden, weil steckbare Module für die Experimente schon dabei sind. Außerdem soll es dazu animieren, solche Boards selbst zu machen, nach dem Motto: „Keine Angst beim SMD löten“. Der Aufbau dieses Boards mit all seinen Pfostenleisten hat dabei durchaus was von einem Igel.

Ich hab nun auch so ein Kit bekommen, einen Sack voll mit „Hühnerfutter“ und „Igelstacheln“, und bin gespannt, wie ich das alles zusammengebastelt kriege. mk

Kommentare über mehrer Zeilen

Ulrich Hoffmann

Forth kennt mit den Worten \ und (zwei Standard–Möglichkeiten, Teile des Quellcodes als Kommentare zu markieren und ihren Inhalt zu ignorieren. Was gibt es zu beachten, wenn sich Kommentare über mehrere Zeilen erstrecken sollen?

Das Standard–Forth–Wort \ erklärt den gesamten Rest der aktuellen Zeile zum Kommentar und sorgt dafür, dass die Verarbeitung des Quelltexte am Anfang der nächste Zeile fortgesetzt wird. Bei (wird der Quelltext bis zur nächsten schließenden Klammer übersprungen. Was aber, wenn es keine nächste Zeile oder keine schließende Klammer im Quelltext mehr gibt? Nun, dann ist eben der Rest des Quelltextes Kommentar. So weit, so gut.

Wenn Standard–Forth–Syteme Quelltext–Dateien interpretieren, dann ist es ihnen erlaubt, dies zeilenweise zu tun: Es wird jeweils eine Zeile in den Speicher geladen und diese dann verarbeitet. Das bedeutet aber, dass der jeweilig aktuelle Quelltext nur eben diese eine Zeile ist und der Rest des Quelltextes eben auch nur bis zum Ende der aktuellen Zeile reicht und nicht bis zum Ende der Datei.

Bei \ ist das nicht schlimm. Bei (fordert der Standard (Forth–94 [3] oder Forth–2012 [4]), dass weitere Zeilen geladen werden, um nach der schließenden Klammer zu suchen. Das kann mit dem Wort `REFILL ( -- flag )` geschehen, das versucht eine weitere Zeile Quellcode zu lesen und mit einem `true`–Flag angibt, dass das erfolgreich war. `false` heißt, dass das Ende der Datei erreicht ist. Die Implementierung von (in gforth [2] und auch VFX–Forth [1] macht genau dies in einer Schleife, um die schließende Klammer zu finden.

VFX–Forth definiert zusätzlich zu den Standard–Worten unter anderem auch ((als Mehrzeilenkommentar–Wort, das Quellcode bis zu einem)) als Kommentar auffasst. Hier wollen wir uns die Funktionsweise genauer ansehen. Eine Implementierung basierend auf dem Forth–94–Standard findet sich im Listing unten.

In Zeile 6 wird im Quellcode (d. h. in der aktuellen Quellcode–Zeile) nach einer schließenden Klammer gesucht. Wurde sie gefunden, stellt Zeile 7 fest, dass noch Quellcode übrig ist. Zeile 9 testet dann, ob eine weitere) folgt. Wenn ja, wird sie übersprungen und das Ende des Kommentars ist gefunden. Folgt kein) oder wurde überhaupt keine schließende Klammer im Quellcode gefunden, dann versucht Zeile 12 mit `REFILL` die nächste Quellzeile zu lesen. Bei Erfolg geht die Suche in Zeile 6 von vorne los. Gibt es keine weitere Quellzeile, liefert `REFILL` ein `false`–Flag und der Kommentar ist auch dann zu Ende.

Wir können ((testen:

```
1  include VFX-multi-line-coment_Forth-94.fs
2  (( wird bis zum Ende der Zeile ignoriert
3  wird ignoriert
4  )).( wird wieder interpretiert)
```

Lädt man diesen Quellcode (Datei `comment-test.fs`), dann wird nach dem `include` das Wort ((am Anfang von Zeile 2 interpretiert. Es sucht, wie eben beschrieben, nach)), findet es aber nicht in Zeile 2, nicht in Zeile 3, aber in Zeile 4. Die Verarbeitung des Quelltextes setzt also in Zeile 4 Spalte 3 wieder ein und es wird die Zeichenkette *wird wieder interpretiert.* ausgegeben: Der Mehrzeilenkommentar wurde erfolgreich übersprungen.

Referenzen

1. http://www.mpeforth.com/vfxcom.htm
2. https://www.gnu.org/software/gforth/
3. http://lars.nocrew.org/dpans/dpans.htm
4. https://forth-standard.org/

Listings

```
1  \ VFX multi line comments based on Forth-94                uh 2015-10-29
2  \ -----------------------
3  : (( ( ccc<double-paren> -- )
4      BEGIN
5        BEGIN
6          [CHAR] ) PARSE 2DROP
7          SOURCE >IN @ /STRING  0 MAX
8        WHILE ( c-addr )
9          C@ [CHAR] ) = IF >IN @ 1+ >IN ! EXIT THEN
10       REPEAT ( c-addr )
11       DROP
12       REFILL 0=
13     UNTIL ; immediate
```

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

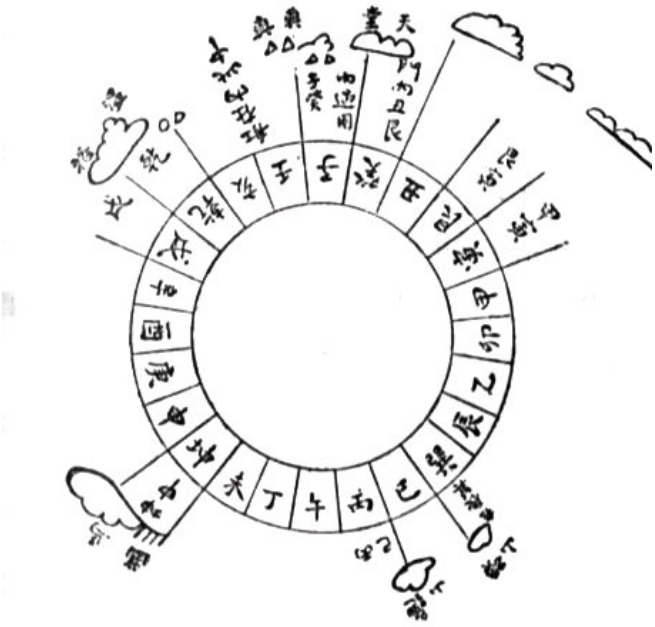
Donnerstags ab 20:00 Uhr
Forth–Chat net2o `forth@bernd` mit dem Key `keysearch kQusJ`

9.–11. September 2016
EuroForth 2016 in Konstanz
`http://www.euroforth.org`

27.–30. Dezember 2016
33c3 in Hamburg
`https://events.ccc.de/`

Und im kommenden Jahr:
25.–27. August 2017:
5. Maker Faire Hannover, im HCC
`http://www.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.

EuroForth — die internationale Forth–Konferenz

Hotel mein Inselglück, Insel Reichenau, Deutschland

9.–11. September 2016

Termine

July 1: Deadline for draft papers (academic stream)
July 28: Notification of acceptance of academic stream papers
August 31: Deadline for camera-ready paper submission
September 7–9: Forth200x meeting
September 9–11: EuroForth 2014 conference
September 11–12: optional 4th day

Weitere Deadlines (paper deadline usw.) werden noch angekündigt

Kosten

Expected cost:	**Delegate**	**Spouse**
Standards meeting:	360€	170€
EuroForth:	420€	190€
4th day:	100€	60€

Anreise

Züge fahren vom Züricher Flughafen (ZRH) halbstündig nach Konstanz und brauchen etwas mehr als eine Stunde, gefolgt von einer 50–minütigen Überfahrt mit der Fähre zur Insel Reichenau (oder mit der Regionalbahn Seehas und Bus 7372/Taxi). Mehr unter `http://www.reichenau-tourismus.de/Infos-Service/Anreise`.

Organisator

Dieses Jahr wird die EuroForth von Klaus Schleisiek organisiert.

`http://www.complang.tuwien.ac.at/anton/euroforth/ef16/`