Vierte Dimension

2/2015

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,

endlich liegt das Heft 2015/2 unseres Forth-Magazins vor Euch. Fast könnte man denken, es tut sich in Sachen Forth nicht mehr viel. Aber weit gefehlt. Zwar erscheint das Heft erst jetzt, weil wir tatsächlich akuten Mangel an Artikeln zu verzeichnen haben — und man kann ja nicht alles selber schreiben. Da müsst Ihr schon helfen: Wenn Ihr nichts schreibt, gibt's auch hier nichts zu lesen! Aber es tut sich viel rund um Forth und das hat sich schon auf unserer Forth-Tagung im April in Hannover angedeutet.

Es gab dort eine Net2o-Party, auf der Bernd Paysans Net2o-System (Internet-2.0) auf verschiedensten Rechnern installiert und zum Zusammenspiel bewegt wurde. Matthias Koch hat auf der Tagung über sein Forth-System Mecrisp [1] und seine Interna berichtet, das ja auch schon im Forth Magazin besprochen wurde (siehe Heft 2013/1). Insbesondere gab es interessante Diskussionen um die von ihm eingesetzte Technik der Konstantenfaltung.

Matthias war über den Sommer sehr produktiv und hat eine Mecrisp-Version für den Raspberry Pi und eine experimentelle Version mit Registerallokator realisiert, der, wenn möglich, Register statt der obersten Stack-Elemente verwendet. Mit der Veröffentlichung des J1a-Forth-Prozessors [2] für die IceStorm-Open-Source-Toolchain [3] für Lattice FPGAs haben auch die Entwicklungen rund um Forth-Maschinen noch einmal Schwung bekommen. Matthias hat sein Mecrisp auch noch für den J1a angepasst, so dass auch dort ein native-code kompilierendes Forth mit Optimierer verfügbar ist. Matthias schreibt das ja auch mit eigenen Worten in seinem Leserbrief. Für dieses Jahr ist noch ein ARM-Forth-Sonderheft mit weiteren Einzelheiten geplant.

Vor 10 Tagen war die EuroForth-Konferenz in Bath und wieder sind viele spannende Dinge dort diskutiert worden. Wieder Forth-Maschinen, viel über Parallelisierung und die Frage, wie Forth — auch Schülern — sinnvoll vermittelt werden kann. Den Forth-Umfang didaktisch einzuschränken und Schritt für Schritt erfahrbar zu machen, war eine Idee, die unter dem Stichwort *Minimal Forth* intensiv diskutiert wurde. Ein Aspekt, den in diesem Heft auch Fred in seinem Artikel über Aussagen-Logik-Primitives anschneidet.

Auch gute Forth-Beispiele, die man sinnvoller Weise an einer zentralen Anlaufstelle finden kann, sind für die Vermittlung von Forth wichtig. Deswegen gibt es nun theForthNet [4], ein Paket-Repository für Forth. Anmelden und mitmachen.

Nicht alle Erfahrungen, die Programmierer mit Forth machen, sind durchweg positiv. Yossis Kreinin schreibt in seinem Artikel *My History With Forth & Stack Machines* in diesem Heft, welche Höhen und Tiefen er dabei durchlaufen hat. Insbesondere den Gedanken, dass die Problemlösungsstrategie von Forth ist, nicht das generelle Problem zu lösen, sondern genau das aktuelle Problem und dabei auch radikal Vereinfachungen anzuwenden, finde ich sehr bemerkenswert.

Viel Freude auch beim Lesen aller anderen spannenden Artikel dieser Ausgabe.

May the Forth be with you,

Ulrich Hoffmann

[1] http://mecrisp.sourceforge.net/
[2] http://excamera.com/sphinx/article-j1a-swapforth.html#j1aswapforth
[3] http://www.clifford.at/icestorm/
[4] http://theforth.net/

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-2015-02

Liebe Maker, Tüftler, Bastler, Designer, Wissenschaftler und Erfinder

Die erste Maker Faire Ruhr geht an den Start! Am Wochenende 12.-13. März 2016 findet das kreative Erfinderfestival in der DASA Arbeitswelt Ausstellung in Dortmund statt. Ab sofort können sich alle interessierten Maker anmelden, der „Call for Makers“ ist eröffnet! `http://www.makerfaire-ruhr.com/` Michael Richter, DASA, email an FG, 08.10.2014




Mecrisp - Neues aus diesem Jahr

Auch in dem Eckchen von Mecrisp ist dieses Jahr viel geschehen: Multitasking fand Einzug beim MSP430, außerdem werden jetzt alle aktuellen Launchpads unterstützt, ganz gleich, ob mit Flash oder mit FRAM. Auch die MSP432-Reihe fand Einzug, jedoch in Mecrisp-Stellaris, da diese Chips eine Mischung eines ARM-Prozessors mit der vom MSP430 bekannten Peripherie darstellen. In Mecrisp-Stellaris sind es vor allem zwei neue Portierungen, die für viel Aufsehen gesorgt haben: Mit dem LPC1114FN28 ist ein für Basteleien sehr interessanter, steckplatinenfreundlicher Chip im DIP-28 Gehäuse hinzugekommen, außerdem ist nach viel Ermunterung bei der Forth-Tagung und mit Hilfe von Ulrich Hoffmann eine Portierung für ARM-Linux und Android entstanden, die auf kreative Einsatzideen wartet.

Ganz neu hinzugekommen ist Mecrisp-Ice: Es läuft auf einem FPGA ! Nachdem in diesem Jahr mit dem "Project IceStorm"die allererste komplett freie Toolchain von Verilog bis hin zum Bitstream entstanden ist, hat James Bowman seinen für das elegante Design bekannten Stackprozessor J1 für den iCEstick von Lattice angepasst. Zu dem darauf laufenden Swapforth, dessen Hauptaugenmerk mehr auf Standardkonformität, Eleganz und Minimalismus liegt, konnte ich die von Mecrisp bekannten Optimierungen hinzufügen, den Komfort für den Programmierer verbessern und IO-Ports nach Art des MSP430 implementieren. Außerdem ist es in Mecrisp-Ice möglich, die eigenen Forth-Programme im aufgelöteten SPI-Flash zu speichern, von wo aus sie beim Start automatisch geladen und ausgeführt werden können, was Elektronik-Projekten mit einem FPGA Tür und Tor öffnet. Für den Kontakt zur Außenwelt stehen Ledcomm und IrDA zur Verfügung, und da Mecrisp-Ice ganz bestimmt auch viele Neugierige anziehen wird, die einmal einen Stackprozessor näher kennenlernen möchten, gehört ein komfortabler Disassembler zur Grundausstattung. Matthias Koch

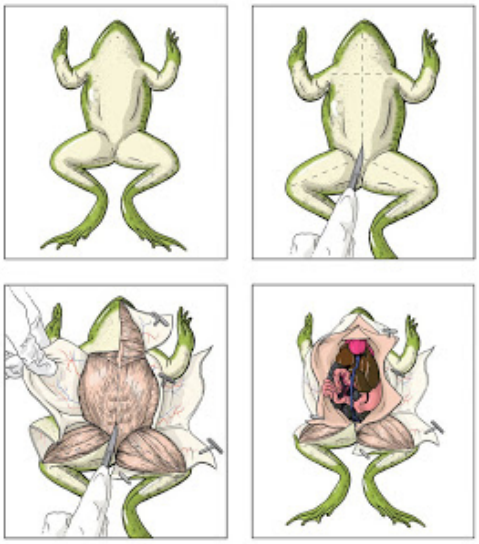
moonforth

This is a detailed interactive guide to building a Forth for an unusual architecture, the DCPU-16. Ever wanted to learn how the system on the Philae comet lander works, ported to an imaginary vintage computer used to control a spaceship in a futuristic game by the creator of Minecraft? Then this is the guide for you! cas
https://moonforth.github.io/

PDP-10

A simple bare-metal forth for the PDP-10. cas
https://github.com/aap/tenth

Frösche sezieren



...
Den Frosch, so denk ich, gilt's zu ehren.
Das tapf're Kerlchen dekapitiert,
Hat selbst im Tod nicht protestiert.
Man sollt auf's Denkmal ihn erheben
Denn er gab sein kostbar Leben
Millionenfach der Forschung hin.
Ohne den Frosch die Medizin,
Das sage ich als Internist,
Wär längst nicht, was sie heute ist.
Ich denke, es ist meine Pflicht,
Auch darauf einmal hinzuweisen.
Des Frosches Verdienst
gilt es zu preisen.
Ihr Seelen all, nehmt mein Gedicht
Als Laudatio für den kühnen Frosch,
Den uns noch immer grünen.
Er war, das weiß heut jedes Kind,
Der Menschheit immer wohlgesinnt.
Hoch leb die Gattung der Ranae;
Hoch leb der Frosch Exploratae.
...

Was das mit Forth zu tun hat? Weiterlessen! ... mk
Zitat aus: Froschmedizin und Krötengift Teil II, von R.W. Aritoquakes, 17. Mai 2012, irgendwo aus den Tiefen von: http://froschmaeusekrieg.blogspot.de/

My History With Forth & Stack Machines

Yossi Kreinin

My VLSI tools take a chip from conception through testing. Perhaps 500 lines of source code. Cadence, Mentor Graphics do the same, more or less. With how much source/object code? – Chuck Moore, the inventor of Forth

This is a personal account of my experience implementing and using the Forth programming language and the stack machine architecture. "Implementing and using" – in that order, pretty much; a somewhat typical order, as will become apparent.

It will also become clear why, having defined the instruction set of a processor designed to run Forth that went into production, I don't consider myself a competent Forth programmer (now is the time to warn that my understanding of Forth is just that – my own understanding; wouldn't count on it too much.) [1]

Why the epigraph about Chuck Moore's VLSI tools? Because Forth is very radical. Black Square kind of radical. An approach to programming seemingly leaving out most if not all of programming:

> ...Forth does it differently. There is no syntax, no redundancy, no typing. There are no errors that can be detected. ...there are no parentheses. No indentation. No hooks, no compatibility. ...No files. No operating system.

Figure 1: Black Square kind of radical

I've never been a huge fan of suprematism or modernism in general. However, a particular modernist can easily get my attention if he's a genius in a traditional sense, with superpowers. Say, he memorizes note sheets upon the first brief glance like Shostakovich did.

Now, I've seen chip design tools by the likes of Cadence and Mentor Graphics. Astronomically costly licenses. Geological run times. And nobody quite knows what they do. To me, VLSI tools in 500 lines qualify as a superpower, enough to grab my attention.

So, here goes.

***

I was intrigued with Forth ever since I read about it in Bruce Eckel's book on C++, a 198-something edition; he said there that "extensibility got a bad reputation due to languages like Forth, where a programmer could change everything and effectively create a programming language of his own". WANT!

A couple of years later, I looked for info on the net, which seemed somewhat scarce. An unusually looking language. Parameters and results passed implicitly on a stack. 2 3 + instead of 2+3. Case-insensitive. Nothing about the extensibility business though.

I thought of nothing better than to dive into the source of an implementation, pForth – and I didn't need anything better, as my mind was immediately blown away by the following passage right at the top of system.fth, the part of pForth implemented in Forth on top of the C interpreter:

```
: ( 41 word drop ; immediate
( That was the definition for the comment word. )
( Now we can add comments to what we are doing! )
```

Now. we. can. add. comments. to. what. we. are. doing.

What this does is define a word (Forth's name for a function) called "(". "(" is executed at compile time (as directed by `IMMEDIATE`). It tells the compiler to read bytes from the source file (that's what the word called, um, `WORD` is doing), until a ")" – ASCII 41 – is found. Those bytes are then ignored (the pointer to them is removed from the stack with `DROP`). So effectively, everything inside "(...)" becomes a comment.

Wow. Yeah, you definitely can't do that in C++. (You can in Lisp but they don't teach you those parts at school. They teach the pure functional parts, where you can't do things that you can in C++. Bastards.)

Read some more and...

```
conditional primitives
: IF ( -- f orig )
   ?comp compile 0branch
   conditional_key >mark ; immediate
: THEN ( f orig -- )
   swap ?condition >resolve ; immediate
: BEGIN ( -- f dest )
   ?comp conditional_key <mark ; immediate
: AGAIN ( f dest -- )
   compile branch
   swap ?condition <resolve ; immediate
: UNTIL ( f dest -- )
   compile 0branch
   swap ?condition <resolve ; immediate
: AHEAD ( -- f orig )
   compile branch
   conditional_key >mark ; immediate
```

Conditional primitives?! Looks like conditional primitives aren't – they define them here. This `COMPILE BRANCH` business modifies the code of a function that uses `IF` or `THEN`, at compile time. `THEN` – one part of the conditional – writes (`RESOLVEs`) a branch offset to a

[1] Beitrag uebernommen aus Yossi Kreinin's blog vom 10. September 2010; mit freundlicher Genehmigung des Authors.

point in code saved (MARKed) by IF, the other part of the conditional.
It's as if a conventional program modified the assembly instructions generated from it at compile time. What? How? Who? How do I wrap my mind around this?
Shocked, I read the source of pForth.
Sort of understood how Forth code was represented and interpreted. Code is this array of "execution tokens" – function pointers, numbers and a few built-ins like branches, basically. A Forth interpreter keeps an instruction pointer into this array (ip), a data stack (ds), and a return stack (rs), and does this:

```
while(true) {
 switch(*ip) {
  //arithmetics (+,-,*...):
  case PLUS: ds.push(ds.pop() + ds.pop());
  ++ip;
  //stack manipulation (drop,swap,rot...):
  case DROP: ds.pop();
  ++ip;
  //literal numbers (1,2,3...):
  case LITERAL: ds.push(ip[1]);
  ip+=2;
  //control flow:
  case COND_BRANCH: if(!ds.pop()) ip+=ip[1];
                    else ip+=2;
  case RETURN: ip = rs.pop();
  //user-defined words:
  //save return address & jump
  default: rs.push(ip+1);
  ip = *ip;
 }
}
```

That's it, pretty much. Similar, say, to the virtual stack machine used to implement Java. One difference is that compiling a Forth program is basically writing to the code array in a WYSIWYG fashion. COMPILE SOMETHING simply appends the address of the word SOMETHING to the end of the code array. So does plain SOMETHING when Forth is compiling rather than interpreting, as it is between a colon and a semicolon, that is, when a word is defined.
So

```
: DRAW-RECTANGLE 2DUP UP RIGHT DOWN LEFT ;
```

simply appends
{&2dup,&up,&right,&down,&left,RETURN}
to the code array. Very straightforward. There are no parameters or declaration/expression syntax as in...

```
void drawRectangle(int width, int height) {
  up(height);
  right(width);
  down(height);
  left(width);
}
```

...to make it less than absolutely clear how the source code maps to executable code. "C maps straightforwardly to assembly"? Ha! Forth maps straightforwardly to assembly. Well, to the assembly language of a virtual stack machine, but still. So one can understand how self-modifying code like IF and THEN works.
On the other hand, compared to drawRectangle, it is somewhat unclear what DRAW-RECTANGLE does. What are those 2 values on the top of the stack that 2DUP duplicates before meaningful English names appear in DRAW-RECTANGLE's definition? This is supposed to be ameliorated by stack comments:

```
: DRAW-RECTANGLE ( width height -- ) ... ;
```

...tells us that DRAW-RECTANGLE expects to find height at the top of the stack, and width right below it.
I went on to sort of understand CREATE/DOES> – a further extension of this compile-time self-modifying code business that you use to "define defining words" (say, CONSTANT, VARIABLE, or CLASS). The CREATE part says what should be done when words (say, class names) are defined by your new defining word. The DOES> part says what should be done when those words are used. For example:

```
: CONSTANT
   CREATE ,
   DOES> @
;
\ usage example:
7 CONSTANT DAYS-IN-WEEK
DAYS-IN-WEEK 2 + . \ should print 9
```

CREATE means that every time CONSTANT is called, a name is read from the source file (similarly to what WORD would have done). Then a new word is created with that name (as a colon would have done). This word records the value of HERE – something like sbrk(0), a pointer past the last allocated data item. When the word is executed, it pushes the saved address onto the data stack, then calls the code after DOES>. The code after CREATE can put some data after HERE, making it available later to the DOES> part.
With CONSTANT, the CREATE part just saves its input (in our example, 7) – the comma word does this: *HERE++ = ds.pop(); The DOES> part then fetches the saved number – the @ sign is the fetch word: ds.push(*ds.pop());
CONSTANT works somewhat similarly to a class, CREATE defining its constructor and DOES> its single method:

```
class Constant
  def initialize(x) @x=x end
  def does() @x end
end
daysInWeek = Constant.new(7)
print daysInWeek.does() + 2
```

...But it's much more compact on all levels.
Another example is defining C-like structs. Stripped down to their bare essentials (and in Forth things tend to be stripped down to their bare essentials), you can say that:

```
struct Rectangle {
  int width;
  int height;
};
```

... simply gives 8 (the structure size) a new name Rectangle, and gives 0 and 4 (the members' offsets) new names, width and height. Here's one way to implement structs in Forth:

```
struct
  cell field width
  cell field height
constant rectangle

\ usage example:
\ here CREATE is used just for allocation
create r1 rectangle allot
  \ r1=HERE; HERE+=8
2 r1 width !
3 r1 height !
: area dup width @ swap height @ * ;
  r1 area .
  \ should print 6
```

CELL is the size of a word; we could say "4 field width" instead of "cell field width" on 32b machines. Here's the definition of FIELD:

```
: field
  ( struct-size field-size -- new-struct-size )
   create over , +
   does> @ +
;
```

Again, pretty compact. The CREATE part stores the offset, a.k.a current struct size (OVER does ds.push(ds[1]), comma does *HERE++=ds.pop()), then adds the field size to the struct size, updating it for the next call to FIELD. The DOES> part fetches the offset, and adds it to the top of the stack, supposedly containing the object base pointer, so that "rect width" or "rect height" compute &rect.width or &rect.height, respectively. Then you can access this address with @ or ! (fetch/store). STRUCT simply pushes 0 to the top of the data stack (initial size value), and at the end, CONSTANT consumes the struct size:

```
struct \ data stack: 0
  cell ( ds: 0 4 ) field width ( ds: 4 )
  cell ( ds: 4 4 ) field height ( ds: 8 )
constant rectangle ( ds: as before STRUCT )
```

You can further extend this to support polymorphic methods – METHOD would work similarly to FIELD, fetching a function pointer ("execution token") through a vtable pointer and an offset kept in the CREATEd part. A basic object system in Forth can thus be implemented in one screen (a Forth code size unit – 16 lines x 64 characters).

To this day, I find it shocking that you can define defining words like CONSTANT, FIELD, CLASS, METHOD – something reserved to built-in keywords and syntactic conventions in most languages – and you can do it so compactly using such crude facilities so trivial to implement. Back when I first saw this, I didn't know about DEFMACRO and how it could be used to implement the defining words of CLOS such as DEFCLASS and DEFMETHOD (another thing about Lisp they don't teach in schools). So Forth was completely mind-blowing.

And then I put Forth aside.

It seemed more suited for number crunching/"systems programming" than text processing/"scripting", whereas it is scripting that is the best trojan horse for pushing a language into an organization. Scripting is usually mission-critical without being acknowledged as such, and many scripts are small and standalone. Look how many popular "scripting languages" there are as opposed to "systems programming languages". Then normalize it by the amount of corporate backing a language got on its way to popularity. Clearly scripting is the best trojan horse.

In short, there were few opportunities to play with Forth at work, so I didn't. I fiddled with the interpreter and with the metaprogramming and then left it at that without doing any real programming.

Here's what Jeff Fox, a prominent member of the Forth community who've worked with Chuck Moore for years, has to say about people like me:

> Forth seems to mean programming applications to some and porting Forth or dissecting Forth to others. And these groups don't seem to have much in common.
>
> ... One learns one set of things about frogs from studying them in their natural environment or by getting a doctorate in zoology and specializing in frogs. And people who spend an hour dissecting a dead frog in a pan of formaldehyde in a biology class learn something else about frogs.
>
> ... One of my favorite examples was that one notable colorforth [a Forth dialect] enthusiast who had spent years studying it, disassembling it, reassembling it and modifying it, and made a lot of public comments about it, but had never bothered running it and in two years of 'study' had not been able to figure out how to do something in colorforth as simple as:
>
> ```
> 1 dup +
> ```
>
> ... [such Forth users] seem to have little interest in what it does, how it is used, or what people using it do with it. But some spend years doing an autopsy on dead code that they don't even run.
>
> Live frogs are just very different than dead frogs.

Ouch. Quite an assault not just on a fraction of a particular community, but on language geeks in general.

> I guess I feel that I could say that if it isn't solving a significant real problem in the real world it isn't really Forth.

True, I guess, and equally true from the viewpoint of someone extensively using any non-mainstream language and claiming enormous productivity gains for experts. Especially true for the core (hard core?) of the Forth community, Forth being their only weapon. They actually live in Forth; it's DIY taken to the extreme, something probably unparalleled in the history of computing, except, perhaps, the case of Lisp environments and Lisp machines (again).

Code running on Forth chips. Chips designed with Forth CAD tools. Tools developed in a Forth environment running on the bare metal of the desktop machine. No standard OS, file system or editor. All in recent years when absolutely nobody else would attempt anything like it. They claim to be 10x to 100x more productive than C programmers (a generic pejorative term for non-Forth programmers; Jeff Fox is careful to put "C" in quotes, presumably either to make the term more generic or more pejorative).

> ... people water down the Forth they do by not exercising most of the freedom it offers... by using Forth only as debugger or a yet another inefficient scripting language to be used 1% of the time.
> Forth is about the freedom to change the language, the compiler, the OS or even the hardware design and is very different than programming languages that are about fitting things to a fixed language syntax in a narrow work context.

What can be said of this? If, in order to "really" enter a programming culture, I need to both "be solving a significant real problem in the real world" and exercising "the freedom to change the language, the compiler, the OS or even the hardware design", then there are very few options for entering this culture indeed. The requirement for "real world work" is almost by definition incompatible with "the freedom to change the language, the compiler, the OS and the hardware design".

And then it so happened that I started working on a real-world project about as close to Forth-level DIY as possible. It was our own hardware, with our own OS, our own compilers, designed to be running our own application. We did use standard CAD tools, desktop operating systems and editors, and had standard RISC cores in the chip and standard C++ cross compilers for them. Well, everyone has weaknesses. Still, the system was custom-tailored, embedded, optimized, standalone, with lots of freedom to exercise – pretty close to the Forth way, in one way.

One part of the system was an image processing co-processor, a variation on the VLIW theme. Its memory access and control flow was weird and limited, to the effect that you could neither access nor jump to an arbitrary memory address. It worked fine for the processing-intensive parts of our image processing programs.

We actually intended to glue those parts together with a few "control instructions" setting up the plentiful control registers of this machine. When I tried, it quickly turned out, as was to be expected, that those "control instructions" must be able to do, well, everything – arithmetic, conditions, loops. In short, we needed a CPU.

We thought about buying a CPU, but it was unclear how we could use an off-the-shelf product. We needed to dispatch VLIW instructions from the same instruction stream. We also needed a weird mixture of features. No caches, no interrupts, no need for more than 16 address bits, but for accessing 64 data bits, and 32-bit arithmetic.

We thought about making our own CPU. The person with the overall responsibility for the hardware design gently told me that I was out of my mind. CPUs have register files and pipeline and pipeline stalls and dependency detection to avoid those stalls and it's too complicated.

And then I asked, how about a stack machine? No register file. Just a 3-stage pipeline – fetch, decode, execute. No problem with register dependencies, always pop inputs from the top of the stack, push the result.

He said it sounded easy enough alright, we could do that. "It's just like my RPN calculator. How would you program it?" "In Forth!"

I defined the instruction set in a couple of hours. It mapped to Forth words as straightforwardly as possible, plus it had a few things Forth doesn't have that C might need, as a kind of insurance (say, access to 16-bit values in memory).

This got approved and implemented; not that it became the schedule bottleneck, but it was harder than we thought. Presumably that was partly the result of not reading "Stack Computers: the new wave", and not studying the chip designs of Forth's creator Chuck Moore, either. I have a feeling that knowledgable people would have sneered at this machine: it was trivial to compile Forth to it, but at the cost of complicating the hardware.

But I was satisfied – I got a general-purpose CPU for setting up my config regs at various times through my programs, and as a side effect, I got a Forth target. And even if it wasn't the most cost-effective Forth target imaginable, it was definitely a time to start using Forth at work.

(Another area of prior art on stack machines that I failed to study in depth was 4stack – an actual VLIW stack machine, with 4 data stacks as suggested by its name. I was very interested in it, especially during the time when we feared implementation problems with the multi-port register file feeding our multiple execution units. I didn't quite figure out how programs would map to 4stack and what the efficiency drop would be when one had to spill stuff from the data stacks to other memory because of data flow complications. So we just went for a standard register file and it worked out.)

The first thing I did was write a Forth cross-compiler for the machine – a 700-line C++ file (and for reasons unknown, the slowest-compiling C++ code that I have ever seen).

I left out all of the metaprogramming stuff. For instance, none of the Forth examples above, the ones that drove me to Forth, could be made to work in my own Forth. No `WORD`, no `COMPILE`, no `IMMEDIATE`, no `CREATE/DOES>`, no nothing. Just colon definitions, RPN syntax, flow control words built into the compiler. "Optimizations" – trivial constant folding so that `1 2 +` becomes 3, and inlining – `: INLINE 1 + ;` works just like `: 1 + ;` but is inlined into the code of the caller. (I was working on the bottlenecks so saving a `CALL` and a `RETURN` was a big deal.) So I had that, plus inline assembly for the VLIW instructions. Pretty basic.

I figured I didn't need the more interesting metaprogramming stuff for my first prototype programs, and I could

add it later if it turned out that I was wrong. It was wierd to throw away everything I originally liked the most, but I was all set to start writing real programs. Solving real problems in the real world. It was among the most painful programming experiences in my life.

All kinds of attempts at libraries and small test programs aside, my biggest program was about 700 lines long (that's 1 line of compiler code for 1 line of application code). Here's a sample function:

```
: mean_std ( sum2 sum inv_len -- mean std )
  \ precise_mean = sum * inv_len;
  tuck u*
   \ sum2 inv_len precise_mean
   \ mean = precise_mean >> FRAC;
  dup FRAC rshift -rot3
   \ mean sum2 inv_len precise_mean
   \ var = (((unsigned long long)sum2
     \     * inv_len) >> FRAC)
     \     - (precise_mean * precise_mean
     \     >> (FRAC*2));
  dup um* nip FRAC 2 * 32 - rshift -rot
   \ mean precise_mean^2 sum2 inv_len
  um* 32 FRAC - lshift swap FRAC rshift or
   \ mean precise_mean^2 sum*inv_len
  swap - isqrt
   \ mean std
;
```

`Tuck u*`.

This computes the mean and the standard deviation of a vector given the sum of its elements, the sum of their squares, and the inverse of its length. It uses scaled integer arithmetic: inv_len is an integer keeping (1<<FRAC)/length. How it arranges the data on the stack is beyond me. It was beyond me at the time when I wrote this function, as indicated by the plentiful comments documenting the stack state, amended by wimpy C-like comments ("C"-like comments) explaining the meaning of the postfix expressions.

This nip/tuck business in the code? Rather than a reference to the drama series on plastic surgery, these are names of Forth stack manipulation words. You can look them up in the standard. I forgot what they do, but it's, like, ds.insert(2,ds.top()), ds.remove(1), this kind of thing.

Good Forth programmers reportedly don't use much of those. Good Forth programmers arrange things so that they flow on the stack. Or they use local variables. My `DRAW-RECTANGLE` definition above, with a `2DUP`, was reasonably flowing by my standards: you get width and height, duplicate both, and have all 4 data items – width,height,width,height – consumed by the next 4 words. Compact, efficient – little stack manipulation. Alternatively we could write:

```
: DRAW-RECTANGLE { width height }
  height UP
  width RIGHT
  height DOWN
  width LEFT
;
```

Less compact, but very readable – not really, if you think about it, since nobody knows how much stuff `UP` leaves on the stack and what share of that stuff `RIGHT` consumes, but readable enough if you assume the obvious. One reason not to use locals is that Chuck Moore hates them:

> I remain adamant that local variables are not only useless, they are harmful.
>
> If you are writing code that needs them you are writing non-optimal code. Don't use local variables. Don't come up with new syntax for describing them and new schemes for implementing them. You can make local variables very efficient especially if you have local registers to store them in, but don't. It's bad. It's wrong.
>
> It is necessary to have [global] variables. ... I don't see any use for [local] variables which are accessed instantaneously.

Another reason not to use locals is that it takes time to store and fetch them. If you have two items on a data stack on a hardware stack machine, `+` will add them in one cycle. If you use a local, then it will take a cycle to store its value with { local_name }, and a cycle to fetch its value every time you mention local_name. On the first version of our machine, it was worse as fetching took 2 cycles. So when I wrote my Forth code, I had to make it "flow" for it to be fast.

The abundance of `DUP`, `SWAP`, `-ROT` and `-ROT3` in my code shows that making it flow wasn't very easy. One problem is that every stack manipulation instruction also costs a cycle, so I started wondering whether I was already past the point where I had a net gain. The other problem was that I couldn't quite follow this flow.

Another feature of good Forth code, which supposedly helps achieve the first good feature ("flow" on the stack), is factoring. Many small definitions.

> Forth is highly factored code. I don't know anything else to say except that Forth is definitions.
>
> If you have a lot of small definitions you are writing Forth. In order to write a lot of small definitions you have to have a stack.

In order to have really small definitions, you do need a stack, I guess – or some other implicit way of passing parameters around; if you do that explicitly, definitions get bigger, right? That's how you can get somewhat Forth-y with Perl – passing things through the implicit variable $_: call chop without arguments, and you will have chopped $_.

Anyway, I tried many small definitions:

```
:inline num_rects   params @ ;
:inline sum         3 lshift gray_sums + ;
:inline sum2        3 lshift gray_sums 4 + + ;
:inline rect_offset 4 lshift ;
:inline inv_area    rect_offset rects 8 + + @ ;
:inline mean_std_stat ( lo hi -- stat )
  FRAC lshift swap 32 FRAC - rshift or
;
: mean_std_loop
  \ inv_global_std =
```

```
  \ (1LL << 32) / MAX(global_std, 1);
  dup 1 max 1 swap u/mod-fx32 drop
  \ 32 frac bits

  num_rects \ start countdown
  begin
   1 - \ rects--
   dup sum2 @
   over sum @
   pick2 inv_area
   mean_std
    \ global_mean global_std inv_global_std
    \ rectind mean std
   rot dup { rectind } 2 NUM_STATS * *
   stats_arr OFT 2 * + + { stats }
    \ stats[OFT+0] =
    \ (short)( ((mean - global_mean)
    \  * inv_global_std) >> (32 - FRAC) );
    \ stats[OFT+1] =
    \ (short)( std * inv_global_std
    \ >> (32 - FRAC) );
   pick2       um* mean_std_stat stats 2 + h!
    \ global_mean global_std inv_global_std mean
   pick3 - over m* mean_std_stat stats h!
   rectind ?dup 0 = \ quit at rect 0
  until
  drop 2drop
;
```

I had a bunch of those short definitions, and yet I couldn't get rid of heavy functions with `DUP` and `OVER` and `PICK` and "C" comments to make any sense of it. This stack business just wasn't for me.

> Stacks are not popular. It's strange to me that they are not. There is just a lot of pressure from vested interests that don't like stacks, they like registers.

But I actually had a vested interest in stacks, and I began to like registers more and more. The thing is, expression trees map perfectly to stacks: (a+b)*(c-d) becomes a b + c d – *. Expression graphs, however, start to get messy: (a+b)*a becomes a dup b + *, and this dup cluttering things up is a moderate example. And an "expression graph" simply means that you use something more than once. How come this clutters up my code? This is reuse. A kind of factoring, if you like. Isn't factoring good?

In fact, now that I thought of it, I didn't understand why stacks were so popular. Vested interests, perhaps? Why is the JVM bytecode and the .NET bytecode and even CPython's bytecode all target stack VMs? Why not use registers the way LLVM does?

Speaking of which. I started to miss a C compiler. I downloaded LLVM. (7000 files plus a huge precompiled gcc binary. 1 hour to build from scratch. So?) I wrote a working back-end for the stack machine within a week. Generating horrible code. Someone else wrote an optimizing back-end in about two months.

After a while, the optimizing back-end's code wasn't any worse than my hand-coded Forth. Its job was somewhat easier than mine since by the time it arrived, it only took 1 cycle to load a local. On the other hand, loads were fast as long as they weren't interleaved with stores – some pipeline thing. So the back-end was careful to reorder things so that huge sequences of loads went first and then huge sequences of stores. Would be a pity to have to do that manually in Forth.

You have no idea how much fun it is to just splatter named variables all over the place, use them in expressions in whatever order you want, and have the compiler schedule things. Although you do it all day. And that was pretty much the end of Forth on that machine; we wrote everything in C.

What does this say about Forth? Not much except that it isn't for me. Take Prolog. I know few things more insulting than having to code in Prolog. Whereas Armstrong developed Erlang in Prolog and liked it much better than reimplementing Erlang in C for speed. I can't imagine how this could be, but this is how it was. People are different.

Would a good Forth programmer do better than me? Yes, but not just at the level of writing the code differently. Rather, at the level of doing everything differently. Remember the freedom quote? "Forth is about the freedom to change the language, the compiler, the OS or even the hardware design".

... And the freedom to change the problem.

Those computations I was doing? In Forth, they wouldn't just write it differently. They wouldn't implement them at all. In fact, we didn't implement them after all, either. The algorithms which made it into production code were very different – in our case, more complicated. In the Forth case, they would have been less complicated. Much less.

Would less complicated algorithms work? I don't know. Probably. Depends on the problem though. Depends on how you define "work", too.

The tiny VLSI toolchain from the epigraph? I showed Chuck Moore's description of that to an ASIC hacker. He said it was very simplistic – no way you could do with that what people are doing with standard tools.

But Chuck Moore isn't doing that, under the assumption that you need not to. Look at the chips he's making. 144-core, but the cores (nodes) are tiny – why would you want them big, if you feel that you can do anything with almost no resources? And they use 18-bit words. Presumably under the assumption that 18 bits is a good quantity, not too small, not too large. Then they write an application note about imlpementing the MD5 hash function:

> MD5 presents a few problems for programming a Green Arrays device. For one thing it depends on modulo 32 bit addition and rotation. Green Arrays chips deal in 18 bit quantities. For another, MD5 is complicated enough that neither the code nor the set of constants required to implement the algorithm will fit into one or even two or three nodes of a Green Arrays computer.

Then they solve these problems by manually implementing 32b addition and splitting the code across nodes. But

if MD5 weren't a standard, you could implement your own hash function without going to all this trouble.

In his chip design tools, Chuck Moore naturally did not use the standard equations:

> Chuck showed me the equations he was using for transistor models in OKAD and compared them to the SPICE equations that required solving several differential equations. He also showed how he scaled the values to simplify the calculation. It is pretty obvious that he has sped up the inner loop a hundred times by simplifying the calculation. He adds that his calculation is not only faster but more accurate than the standard SPICE equation. ... He said, "I originally chose mV for internal units. But using 6400 mV = 4096 units replaces a divide with a shift and requires only 2 multiplies per transistor. ... Even the multiplies are optimized to only step through as many bits of precision as needed.

This is Forth. Seriously. Forth is not the language. Forth the language captures nothing, it's a moving target. Chuck Moore constantly tweaks the language and largely dismisses the ANS standard as rooted in the past and bloated. Forth is the approach to engineering aiming to produce as small, simple and optimal system as possible, by shaving off as many requirements of every imaginable kind as you can.

That's why its metaprogramming is so amazingly compact. It's similar to Lisp's metaprogramming in much the same way bacterial genetic code is similar to that of humans – both reproduce. Humans also do many other things that bacteria can't (...No compatibility. No files. No operating system). And have a ton of useless junk in their DNA, their bodies and their habitat.

Bacteria have no junk in their DNA. Junk slows down the copying of the DNA which creates a reproduction bottleneck so junk mutations can't compete. If it can be eliminated, it should. Bacteria are small, simple, optimal systems, with as many requirements shaved off as possible. They won't conquer space, but they'll survive a nuclear war.

This stack business? Just a tiny aspect of the matter. You have complicated expression graphs? Why do you have complicated expression graphs? The reason Forth the language doesn't have variables is because you can eliminate them, therefore they are junk, therefore you should eliminate them. What about those expressions in your Forth program? Junk, most likely. Delete!

I can't do that.

I can't face people and tell them that they have to use 18b words. In fact I take pride in the support for all the data types people are used to from C in our VLIW machine. You can add signed bytes, and unsigned shorts, and you even have instructions adding bytes to shorts. Why? Do I believe that people actually need all those combinations? Do I believe that they can't force their 16b unsigned shorts into 15b signed shorts to save hardware the trouble?

OF COURSE NOT.

They just don't want to. They want their 16 bits. They whine about their 16th bit. Why do they want 16 and not 18? Because they grew up on C. "C". It's completely ridiculous, but nevertheless, people are like that. And I'm not going to fight that, because I am not responsible for algorithms, other people are, and I want them happy, at least to a reasonable extent, and if they can be made happier at a reasonable cost, I gladly pay it. (I'm not saying you can't market a machine with a limited data type support, just using this as an example of the kind of junk I'm willing to carry that in Forth it is not recommended to carry.)

Why pay this cost? Because I don't do algorithms, other people do, so I have to trust them and respect their judgment to a large extent. Because you need superhuman abilities to work without layers. My minimal stack of layers is – problem, software, hardware. People working on the problem (algorithms, UI, whatever) can't do software, not really. People doing software can't do hardware, not really. And people doing hardware can't do software, etc.

The Forth way of focusing on just the problem you need to solve seems to more or less require that the same person or a very tightly united group focus on all three of these things, and pick the right algorithms, the right computer architecture, the right language, the right word size, etc. I don't know how to make this work.

My experience is, you try to compress the 3 absolutely necessary layers to 2, you get a disaster. Have your algorithms people talk directly to your hardware people, without going through software people, and you'll get a disaster. Because neither understands software very well, and you'll end up with an unusable machine. Something with elaborate computational capabilities that can't be put together into anything meaningful. Because gluing it together, dispatching, that's the software part.

So you need at least 3 teams, or people, or hats, that are to an extent ignorant about each other's work. Even if you're doing everything in-house, which, according to Jeff Fox, was essentially a precondition to "doing Forth". So there's another precondtion – having people being able to do what at least 3 people in their respective areas normally do, and concentrating on those 3 things at the same time. Doing the cross-layer global optimization.

It's not how I work. I don't have the brain nor the knowledge nor the love for prolonged meditation. I compensate with, um, people skills. I find out what people need, that is, what they think they need, and I negotiate, and I find reasonable compromises, and I include my narrow understanding of my part – software tools and such – into those compromises. This drags junk in. I live with that.

> I wish I knew what to tell you that would lead you to write good Forth. I can demonstrate. I have demonstrated in the past, ad nauseam, applications where I can reduce the amount of code by 90% and in some cases 99%. It can be done, but in a case by case basis. The general principle still eludes me.

And I think he can, especially when compatibility isn't a must. But not me.

I still find Forth amazing, and I'd gladly hack on it upon any opportunity. It still gives you the most bang for the buck – it implements the most functionality in the least space. So still a great fit for tiny targets, and unlikely to be surpassed. Both because it's optimized so well and because the times when only bacteria survived in the amounts of RAM available are largely gone so there's little competition.

As to using Forth as a source of ideas on programming and language design in general – not me. I find that those ideas grow out of an approach to problem solving that I could never apply.

Update (July 2014):

Jeff Fox's "dead frog dissector" explained his view of the matter in a comment[2] to this article, telling us why the frog (colorForth) died in his hands in the first place... A rather enlightening incident, this little story.

—

Comment of Albert van der Horst on 07.16.14

I'm the person who was dissecting colorforth, a dead frog. The comment from Jeff Fox is totally unfair. The only reason I did it was THAT COLORFORTH JUST DIDN'T RUN ON THE THREE MACHINES I HAD AVAILABLE. And he nor Chuck Moore had any consideration with people like me or would waste their time helping me get colorforth up and running. I never managed to run colorforth, until such time as others made an emulation environment to boot a floppy image under windows. (Did you know that colorforth is just a floppy image? Of late Chuck complained that he lost work because his notebook cum floppy died. Backups? Source control?)

I'm a huge fan of Forth and an implementor, see my website. I thank you for a balanced view of Forth, showing the other reality. It is sobering and instructive and few people bother to write down negative experiences.

For what it is my disassembly of colorforth is impressive. It regenerates assembly labels from hints in the source which is hard enough. But the names in the source are all but encrypted! So indeed Forth works wonders, for the right people, which is certainly not everybody.

Then the 18 bits of the GA144. Don't be impressed. It is a stupid design error. At the time 18 bits static memory chips where in fashion. They didn't think further than just interfacing those chip at the time. The chip has an unbalance between processing and communication power/data availability. Numerous discussions by Forth experts who know about hardware have not found a typical area where you could use those chips. If they where 40 cents instead of 40 dollar it might be a different matter, but no. And the pinout! Only hobbyist would try those chips out. There are no pins„ just solderareas, with a stride of .4 mm (not mils, 400 micrometer). Hobbyist trying those out would be probably my age (over sixty).

Links

- Yossi.Kreinin@gmail.com
- http://yosefk.com/blog/my-history-with-forth-stack-machines.html
- http://home.hccnet.nl/a.w.m.van.der.horst/

And another (not so good) news:

Lua boot loader

- URL: https://svnweb.freebsd.org/base/projects/lua-bootloader/
- Contact: Rui Paulo <rpaulo@FreeBSD.org>
- Contact: Pedro Souza <pedrosouza@FreeBSD.org>
- Contact: Wojciech Koszek <wkoszek@FreeBSD.org>

The Lua boot loader project is in its final stage and it can be used on x86 already. The aim of this project is to replace the Forth boot loader with a Lua boot loader. All the scripts were re-written in Lua and are available in `sys/boot/lua`. Once all the Forth features have been tested and the boot menus look exactly like in Forth, we will start merging this project to FreeBSD HEAD. Both loaders can co-exist in the source tree with no problems because a pluggable loader was introduced for this purpose.

The project was initially started by Wojciech Koszek, and Pedro Souza wrote most of the Lua code last year in his Google Summer of Code project.

To build a Lua boot loader just use:

```
WITH_LUA=y WITHOUT_FORTH=y
```

Open tasks:

1. Feature/appearance parity with Forth.
2. Investigate use of floating point by Lua.
3. Test the EFI Lua loader.
4. Test the U-Boot Lua loader.
5. Test the serial console. (cas)

[2] Albert van der Horst on 07.16.14 at 3:50 pm

Aussagen-Logik: Ein PC-Forth-Teilsystem mit nur 6 Primitives

Fred Behringer

Wie viele Primitives braucht ein vernünftiges PC-Forth? Ich bin nahe daran, mit 7 hinzukommen. Hier aber beschränke ich mich auf die (zweiwertige) Aussagen-Logik und löse diese mir selbst gestellte Teilaufgabe mit 6 Primitives. Es läuft alles sowohl unter Turbo-Forth als auch unter ZF (beide in 16-Bit-Ausführung). Aufrufen tu ich über Windows-XP.(!)

Bei Logik auf dem Computer denkt man in erster Linie an die allgegenwärtigen Maschinen-Befehle in Form von Assembler-Mnemonics. Low-Level in Forth ist gleichbedeutend mit *Primitives*. Ich möchte hier mal schnell die 16 ein- und zweistelligen Grundfunktionen der zweiwertigen Aussagenlogik in unmittelbar ausführbarer Weise sammeln und dabei möglichst wenige *Primitives* (sprich Code-Definitionen) verwenden. Ich schaffe das mit den 6 *Primitives* `r> >r @ ! and invert` . Bekanntlich kann man die zweiwertige Aussagen-Logik mit den beiden Funktionen *and* (bitweise sowohl als auch) und *invert* (Umkehrung aller Bits) aufbauen. Irgendwie müsste man aber noch die Konstanten erzeugen (ich verwende 0 für falsch und -1 für wahr). Und ganz ohne Operationen auf dem Datenstack geht es auch nicht. Ob das alles schon zu den Aufgaben eines Colon-Compilers gehört, bleibe dahingestellt. Ich verwende einfach das Gesamtsystem (Turbo-Forth oder ZF) fürs Compilieren und bin sicher, dass der eigentliche Colon-Compiler dort enthalten ist. Die Logik-Worte, die ich hier besprechen möchte, lege ich in ein Vokabular namens `logik`. Und für die Überprüfung der Worte der Aussagen-Logik schalte ich das Vokabular `forth` dann ab und lasse nur noch das Vokabular `logik` zu.

Eine Schwierigkeit bleibt: Ich kann nicht erwarten, dass Ausgaben auf dem Bildschirm schon zum Aufgabenbereich des Colon-Compilers gehören. (0 und -1 sind hier bei mir Forth-Worte. Von anderen Ganzzahlen ist im Vokabular `logik` keine Rede - sollte dort keine Rede sein. Andererseits ist die Überprüfung beispielsweise von

```
: xor ( n1 n2 -- n3 )
  over over invert and >r
  swap invert and r> or ;
```

ohne . (Punkt) oder ohne `.s` (Stack-Inhalt) kaum möglich. Ich darf also schnell mogeln und mir `.s`, genau wie die 6 *Primitives*, vom Vokabular `forth` her mit ins Vokabular `logik` ziehen. Für die Verwendung von . oder `.s` in `logik` wird `forth` dann hinterher abgeschaltet.

```
only forth also logik definitions : .s .s ;
only forth logik
```

Interessant vielleicht noch die Bemerkung, dass ZF per `.s` nur die letzten 4 Stack-Inhalte ausgibt. Das tut aber meinem Vorhaben keinen Abbruch.

Zum Abschluss des Textteils noch eine Frage an die Wissenden: Ich spreche im Listing zweimal von "Mogeln" (bei .s und bei .). Wie kann ich es aber prinzipiell verhindern, dass sich beim Aufbau des Vokabulars `logik` Worte einschleichen, die zwar dem Colon-Compiler entstammen, die aber in `logik` noch nicht definiert sind (Ausnahmen: . (Punkt) `.s` und die 6 *Primitives*)? Im vorliegenden Artikel habe ich einfach auf meine Disziplin beim Schema "Jedes Wort in `logik` verwendet nur Worte, die schon in `logik` definiert sind" vertraut. Heißt das Zauberwort der Lösung dieser Frage *Meta-Compilation*? Ich werde mal bei *noForth* (von Albert Nijhof und Willem Ouwerkerk) im vorigen Heft (siehe Interview unter redaktioneller Führung von Ulrich Hoffmann) nachforschen.

Listing

```
1  \ Fuer Turbo-Forth (TF) einladen ueber: include list.txt attributs off [ret]
2  \ Fuer              ZF  einladen ueber:   fload list.txt               [ret]
3
4  \ Der Colon-Compiler erzeugt daraus beim Einladen ein Compilat, in dem es nur
5  \ die hier besprochenen Forth-Worte gibt. Sie stehen im urspruenglichen System
6  \ im Vokabular logik zur Verfuegung und koennen per words aufgerufen werden.
7  \ Will man (zum Beispiel, um das Erreichte zu ueberpruefen) ins zugrunde
8  \ liegende Forth-System zurueck, so kann man das durch Aufrufen von only forth
9  \ also logik erreichen.
10
11 only forth also definitions
12 vocabulary logik
13 logik definitions
14
15 hex         \ Alle Zahlenangaben hexadezimal !
16
17 \ Zunaechst die 6 Primitives, mit denen ich im Vorliegenden auskomme.
18 \ Mehrfachdefinitionen seien, wie ueblich, erlaubt. Die juengste gilt.
19 \ invert gibt es in TF und ZF nicht und wird hier ergaenzt.
20 \ Ausser invert sind diese Primitives sowohl in TF wie auch in ZF alle
21 \ schon enthalten. Aber doppelt haelt (sicherheitshalber) besser ;-)
```

```
22
23  code r>     ( -- n )          0 [rp] ax mov rp inc rp inc        1push end-code
24  code >r     ( n -- )          ax pop rp dec rp dec ax 0 [rp] mov  next end-code
25  code and    ( n1 n2 -- n3 )   bx pop ax pop bx ax and            1push end-code
26  code invert ( n -- n' )       ax pop ax not                      1push end-code
27  code @      ( ad -- n )       bx pop 0 [bx] push                  next end-code
28  code !      ( n ad -- )       bx pop 0 [bx]  pop                  next end-code
29
30  \ Es wurde alles Folgende sowohl in TF wie auch in ZF ausprobiert.
31  \ Das Ausprobieren geschah unter XP (kein Extra-Dos-System noetig).
32  \ TF oder/und ZF werden in der 16-Bit-Version vorausgesetzt (Internet).
33  \ TF und ZF liegen bei mir auf einem FAT32-formatierten USB-2.0-Stick,
34
35  : (r>) ( -- r-ad )  r>  ;
36
37  :  v1  ( --   ad ) (r>) ; ;
38  :  v2  ( --   ad ) (r>) ; ;
39
40  \ Achtung: Hier ist im Quelltext das doppelte Semikolon wichtig! Es
41  \ schafft Platz fuer den Wert der 'Variablen' v1 und v2 (ergaenzbar).
42
43  \ Es folgen ein paar Stack-Operationen und der Returnstack-Operator r@.
44
45  \ Alle besprochenen Highlevel-Operationen benoetigen nur die 6 Primitives
46  \ r> >r and invert @ ! . Die Immediate-Eigenschaft wird nicht gebraucht.
47
48  : drop ( n -- )     v1 !                        ;
49  : dup  ( n -- n n ) v1 ! v1 @ v1 @              ;
50  : swap ( n1 n2 -- n2 n1 ) v1 ! v2 ! v1 @ v2 @   ;
51  : r@ ( -- n ) r> r> dup >r swap >r              ;
52  : rot  ( n1 n2 n3 -- n2 n3 n1 ) >r swap r> swap  ;
53  : over ( n1 n2    -- n1 n2 n1 ) swap dup rot rot ;
54
55  \ Ohne Stackausgabe kann nicht vernuenftig gearbeitet werden. Ich
56  \ beschaffe mir dazu .s auf demselben Weg wie bei den Primitives.
57
58  : .s   ( -- stackwerte ) .s                     ;
59
60  \ Man sehe mir diese kleine Mogelei nach! Mein Thema sind hier die
61  \ 16 Grundfunktionen der Aussagen-Logik. Um beispielsweise bei der
62  \ Ausgabe von 0 -1 seq1 . nur die Wahrheiswerte 0 und -1 zu
63  \ beruecksichtigen, noch eine weitere Mogelei:
64
65  : . ( -- 0/-1 ) dup 0= if 30 emit else ascii - emit 31 emit then ;
66
67  \ Diese Mogelei laesst, man moege sich darueber nicht wundern, auch
68  \ beispielsweise 4711 als Tastatur-Eingabe zu. Das Wort . (Punkt)
69  \ macht im Vokabular logik daraus -1 :
70
71  \ Frage an den interessierten Leser:
72  \ 4711 4713 equ . [ret] -1 ok
73  \ Warum?
74
75  \ Jetzt die Grundfunktionen der zweiwertien Aussagen-Logik. Als
76  \ Wahrheitswerte dienen, wie ueblich, 0 und -1 (-1, nicht +1 !):
77  \ 0 = 0000000000000000b ; -1 = 1111111111111111b.
78  \ 0 und -1 werden ohne Verwendung des Zahlen-Interpreters erzeugt.
79
80  :  or   ( n1 n2 -- n3 ) invert swap invert and invert               ;
81  : nor   ( n1 n2 -- n3 ) invert swap invert and                      ;
82  : xor   ( n1 n2 -- n3 ) over over invert and >r swap invert and r> or ;
83  :   0   (       --  0 )   r@ dup xor                                ;
84  :  -1   (       -- -1 )    0 invert                                 ;
85  \ and   ( n1 n2 -- n3 )  ist Primitive!
86  : nand  ( n1 n2 -- n3 )  and invert                                 ;
87  : seq1  ( n1 n2 -- n3 ) swap invert or                              ;
88  : nseq1 ( n1 n2 -- n3 ) seq1 invert                                 ;
89  : seq2  ( n1 n2 -- n3 )      invert or                              ;
90  : nseq2 ( n1 n2 -- n3 ) seq2 invert                                 ;
91  : equ   ( n1 n2 -- n3 )  xor invert                                 ;
92
93
94  \ Wahrheitstafel (a' := a invert)
95  \ -------------------------------
96  \                       a:b = 0:0 -1:0 0:-1 -1:-1
97  \ a b or                    =  0   -1   -1   -1
```

```
98  \ a b nor                  = -1    0    0    0
99  \ a b xor                  =  0   -1   -1    0
100 \ a b and                  =  0    0    0   -1
101 \ a b nand                 = -1   -1   -1    0
102 \ a b seq1  <-> a' b    or = -1    0   -1   -1   ex a=0 quodlibet
103 \ a b nseq1 <-> a' b   nor =  0   -1    0    0
104 \ a b seq2  <-> a  b'   or = -1   -1    0   -1
105 \ a b nseq2 <-> a  b'  nor =  0    0   -1    0
106 \ a b equ                  = -1    0    0   -1
107
108 \ Die Bezeichnung seq fuer die materiale Implikation ist aus der
109 \ Logik bekannt. Es gilt: ex falso quodlibet und ex quodlibet verum.
110 \ Dabei geht man unbewusst von a (in der obigen Colon-Definition: n1)
111 \ als Praemisse aus (siehe seq1). Nimmt man b (also n2 statt n1) als
112 \ Praemisse, dann erhaelt man die oben mit seq2 bezeichnete
113 \ Logik-Funktion. (Schwierigkeiten mit der fragwuerdigen "Logik" der
114 \ Logik der Implikation werden in der Logik-Literatur diskutiert. In
115 \ der Schaltalgebra (Boolesche Algebra) gibt es keine Schwierigkeiten.
116 \ Wir tun uns in Forth sicher leichter, wenn wir die Diskussionen der
117 \ Logiker ueber die Paradoxien der Implikation vergessen und stattdessen
118 \ an die Interpretations-Moeglichkeiten der Schaltalgebra denken.)
119 \ Mit nseq1 und nseq2 lehne ich mich an nand und an nor an.
120
121 \ Man kann den Ausdruck "a b seq1" auch als "aus a folgt b" lesen.
122 \ Das gilt genau dann, wenn "aus nicht b folgt nicht a" gilt. Das
123 \ wiederum gilt genau dann, wenn man a invert b or ersetzt durch
124 \ a b invert nand. Letzteres erhaelt man aus dem De-Morgan-Theorem.
125
126 \ Zum Arbeiten innerhalb von logik, sprich: zur Ueberpruefung, darf
127 \ ich jetzt das Vokabular forth abschalten (unzugaenglich machen).
128 \ (Das wirkt bis hin zum Unvermoegen, das System in diesem Zustand
129 \ per bye zu verlassen.) ;-)
130
131 only forth logik
```

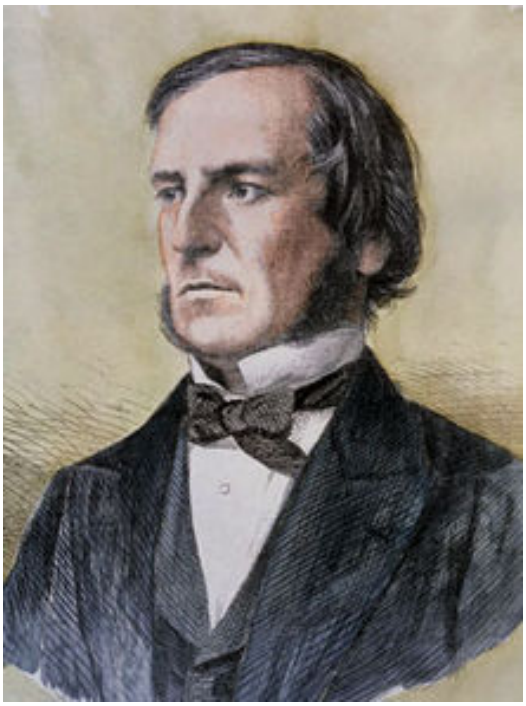


Abbildung 1: George Boole (um 1860)

George Boole (* 2. November 1815 in Lincoln, England; † 8. Dezember 1864 in Ballintemple, in der Grafschaft Cork, Irland) war ein englischer Mathematiker (Autodidakt), Logiker und Philosoph. Er gilt als der Begründer der Aussagenlogik.

Boole schuf in seiner Schrift *The Mathematical Analysis of Logic* von 1847 den ersten algebraischen Logikkalkül und begründete damit die moderne mathematische Logik, die sich von der bis dato üblichen philosophischen Logik durch eine konsequente Formalisierung abhebt. Er formalisierte die klassische Logik und Aussagenlogik und entwickelte ein Entscheidungsverfahren für die wahren Formeln über eine disjunktive Normalform.[1][2] Boole nahm damit – da aus der Entscheidbarkeit der klassischen Logik ihre Vollständigkeit und Widerspruchsfreiheit folgt – schon gut 70 Jahre vor Hilberts Programm für ein zentrales Logikgebiet die Lösung der von David Hilbert gestellten Probleme vorweg. Als Verallgemeinerungen von Booles Logikkalkül wurden später die sogenannte *boolesche Algebra* und der *boolesche Ring* nach ihm benannt. (Quelle: Wikipedia)

Wave Engine (9)

Hannes Teich

Im VD-Heft 4/2013 wurde ein kleiner in Gforth programmierter Synthesizer besprochen, der über eine abspielbare Wave-Datei eine Reihe vierstimmiger Akkorde erklingen lässt. Da die Berechnung der Datei recht lange dauert, wird nun versucht, über eine Sinustabelle Zeit zu gewinnen.

Kleine Überraschung

Um es gleich zu verraten: Die Funktion `fsin` ist unschlagbar tüchtig. Bei der Verwendung einer Sinustabelle (360 Grad, Float-Zahlen) wurde 16 % mehr Zeit als mit `fsin` gemessen. Legt man eine Tabelle mit nur 90 Grad an, dauert das Auslesen noch bedeutend länger. Sicherlich würde mit Integer-Zahlen Zeit gewonnen, aber das wollte ich mir nicht antun, zumal es sich hier nur um die Zeit der Berechnung handelt und nicht etwa um die Abspielgeschwindigkeit der Wave-Datei.

Warum es sich dennoch lohnt

Bei diesen Forschungen habe ich immer den geplanten Vollausbau im Sinn. In der Version aus Heft 4/2013 gab es 2 Manuale zu je 2 Stimmen, jede Stimme aus zwei Teiltönen bestehend (Grundton und ein Oberton). Damit ist das Prinzip des Vollausbaus gegeben: 4 Manuale zu je 6 Stimmen mit 16 Teiltönen werden ebenso berechnet.

Der Begriff „Manual“ ist für die Syntax der besonderen Notenschrift durchaus passend. Im Heft 2/2011 ist ein Beispiel zu sehen (und zu hören). Damals gab es bereits einen funktionierenden Vollausbau der Wave-Engine, aber – wie berichtet – das Projekt soll neu und ordentlicher hochgezogen werden. Fein wäre es, wenn das schneller ginge.

Also, weshalb soll sich eine Sinustabelle dennoch lohnen? Man bedenke: Bei einer Abtastrate der Wave-Datei von 44,1 kHz sind für alle 23 Mikrosekunden (Abspielzeit) 4 * 6 * 16 = 384 Sinuswerte zu berechnen. Da kann man für ein kleines Musikstück locker Kaffee trinken gehen. Eine Beschleunigung um einen Faktor Zehn wäre sehr willkommen.

Und das lässt sich durch einen Trick bewerkstelligen. Die maximal 16 Teiltöne brauchen nicht jedesmal berechnet zu werden, wenn man sie in der Sinus-Tabelle mit ablegt. Dann geht es alle 23 Mikrosekunden nur noch um 4 * 6 = 24 Werte aus der Tabelle. Normalerweise werden nicht alle 16 Teiltöne bemüht, darum spreche ich nur von einem Faktor 10, ungefähr.

Programmtechnisches

Das in erwähnter Weise funktionierende Programm ist an bekannter[1] Stelle, aber auch beim Autor abrufbar. Es geht immer noch um zwei Manuale zu zwei Stimmen, ausreichend für vierstimmige Akkorde, aber statt zweier Teiltöne gibt es jetzt deren 16 Stück, was sich deutlich im Klangbild zeigt.

Das Programm besteht aus sechs Dateien. *WAVER3-04.fs* inkludiert *goodies.fs* (Vordefinitionen), *freqtab.fs* (Frequenztabellen), *wavgen.fs* (Wave-Generator), *partitur.fs* (Notenblatt) und *singen.fs* (Sinus-Generator). Die hauptsächlichen Veränderungen betreffen die zuletzt genannte Datei.

```
: filltab ( --)
  dots 0 do
   resol i s>f f* fsin .7e f* \ Grundton
   resol i 2* s>f f* fsin .1e f* f+
   resol i 3 * s>f f* fsin .3e f* f+
   resol i 4 * s>f f* fsin .1e f* f+
   resol i 5 * s>f f* fsin .2e f* f+
   resol i 6 * s>f f* fsin .3e f* f+
   resol i 7 * s>f f* fsin .05e f* f+
   resol i 8 * s>f f* fsin .1e f* f+
   resol i 9 * s>f f* fsin .1e f* f+
   resol i 10 * s>f f* fsin .2e f* f+
   resol i 11 * s>f f* fsin .05e f* f+
   resol i 12 * s>f f* fsin .3e f* f+
   resol i 13 * s>f f* fsin .05e f* f+
   resol i 14 * s>f f* fsin .05e f* f+
   resol i 15 * s>f f* fsin .1e f* f+
   resol i 16 * s>f f* fsin .1e f* f+
   sintab i 8 * + f!
  loop ;
```

`filltab` zeigt das Füllen der Sinustabelle mit 16 Teiltönen. Wobei `resol` die gewählte Auflösung ist (2pi / Anzahl Stützpunkte). 360 Stützpunkte pro voller Periode des Grundtons scheinen klanglich zu reichen. Somit läuft die Schleife 360 mal und legt jedesmal einen Wert aus dem Gemisch der Sinuslinien nieder. Abb. 1 zeigt das Ergebnis.

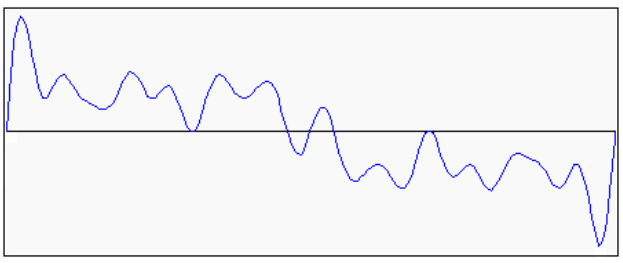


Abbildung 1: So sieht die abgelegte Kurve aus.

[1] Auf www.forth-ev.de im Menue *Download* und dann *Wave Engine* anklicken.

```
: loops ( --)
   durat 2* 0       \ Tondauer (17 bit)
   ?DO
     0e accu f!     \ Akku leeren
     #manuals 0
     ?DO
       i to manu#   \ Manual-Nummer
       #voices 0
         ?DO
         i to lvox# \ Stimmen-Nummer
         i j >voice
         to gvox#   \ dito fortlaufend
         generate   \ Teilton bearbeiten
       LOOP
     LOOP
   accu f@ .5e f+
   f>s w>buf        \ in die Wave-Datei
   LOOP ;
```

In `loops` sind die ineinander verschachtelten Schleifen zu erkennen. Hier war früher im Inneren noch die Teilton-Schleife vorhanden, die jetzt wegfallen konnte. Es werden nacheinander die Stimmen aller Manuale ausgewählt und der Funktion `generate` übergeben.

```
: generate ( --)
   gvox# 24 * gtable +
   dup f@ fsin2   \ Sinuswert aus Phase
   dup 16 + f@ f* \ Sinuswert mal Huellkurve
   accu f+!       \ auf Akku aufaddieren
   dup 8 + f@     \ Phasenschritt
   dup f+!        \ auf Phase aufaddieren
   16 + dup f@
   damp f* f! ;   \ Huellkurve daempfen
```

In `generate` wird die Funktion `fsin2` verwendet, die sich ähnlich wie `fsin` verhält, allerdings die Werte aus der Sinus-Tabelle holt. Wird `fsin2` durch `fsin` ersetzt, so ertönt jeweils nur der Grundton, da die Teiltöne nicht mehr direkt berechnet werden, sondern in der Tabelle vorliegen.

```
: fsin2 ( arc -- sin) \ 16 % langsamer als fsin
  2pi fmod      \ ( --) ( arc)
  resol f/      \ ( f1) ( dots)
  fround f>s    \ ( f1 dots)
  8 * sintab +  \ ( f1 loc)
  f@ ;
```

`fsin2` ist der Ersatz für `fsin` . Zunächst wird der Winkel *modulo 2pi* eingegrenzt. Die Float-Konstante `resol` bestimmt die Auflösung, hier 360 Stützpunkte. Zur Adressierung des Tabellenortes ist eine Rundung zur nächstgelegenen Ganzzahl angebracht, damit Sinus pi/6 auch wirklich 0.5 ergibt.

Der Aufruf des Programms

Wie in Abb. 2 zu sehen, verwende ich (mit Windows 8.1) die PowerShell (PS). Es wird zunächst der Desktop gewählt, denn dort befindet sich der Ordner *Waver3-04* , der die sechs Dateien des Wave-Engine-Programms enthält. Durch Eingabe von `go` läuft das Programm ab und erzeugt die Wave-Datei *WE-001.wav* auf dem Desktop. Die Meldung `<0> <0>` besagt, dass Daten- und Float-Stack am Ende leer sind (nützlich beim Experimentieren).

Mit Linux habe ich's diesmal nicht getestet. Der Aufruf sollte einfacher sein als mit Windows.

Ausblick

Immer noch geplant ist das Aufbohren auf 4 Manuale zu je 6 Stimmen und 16 Teiltönen pro Stimme. Dies alles soll dreifach vorhanden sein, zweimal zum „Verlebendigen" der Einschwingvorgänge. Auch weicher Toneinsatz ist geplant. Das Abklingen der Töne wurde hier bereits verwirklicht. Ein neuer Ton beendet den alten. (All dies lief früher schon.)

Aufwändig, aber wichtig (und einst auch funktionierend gewesen) ist die Interpretation der speziellen Noten-Syntax, wie sie früher beschrieben worden ist. Die Partitur (das „Notenblatt") im aktuellen Fall ist ein Witz[2], das ist klar. Die Auswahl der Teiltöne (die Klangfarbe) soll mit ins Notenblatt, versteht sich.

Eine Bemerkung zur Wave-Datei: Der Windows Media Player weigert sich, sie abzuspielen. Ich verwende stattdessen Audacity (Windows und Linux), das die Kurven anzeigt und mit dem auch eine MP3-Datei erzeugt werden kann[3].

Und damit für diesmal genug. Happy testing!

Listing

Der umfangreiche Source-Code wurde schon im Heft 4/2013 vorgestellt. Die derzeitige Fassung hat rund 18 Seiten Source-Code, und da die aktuellen Änderungen in diesem Beitrag vorgestellt wurden, haben wir diesmal in Absprache mit dem Autor darauf verzichtet erneut alles abzudrucken. Die Quelle steht zum Herunterladen bereit: http://tinyurl.com/Waver3-04 oder http://www.forth-ev.de/filemgmt/viewcat.php?cid=54

[2] Damit kann ich aber leben, um das Ganze in Forth erstmal zu erproben (mk)

[3] WE-001.wav ließ sich auf dem guten alten PowerBook G4, OSX, mit Quicktime anstandslos abspielen. OpenSuse kennt die "Rhythmbox", ein Musikspieler. Dort bekommt man die Meldung, es gäbe keine "playback URI" für die Datei, und darum würde die Datei nicht abgespielt, es wird mit der Fehlermeldung "Error transferring track" abgebrochen. OpenSuse kennt aber auch das Paket *sox*. Damit erklang von der Konsole aus mit dem Kommando *play WE-001.wav* das Stück auch ganz wunderbar. Und in Win7 gings mit Audacity, und mit Winamp.

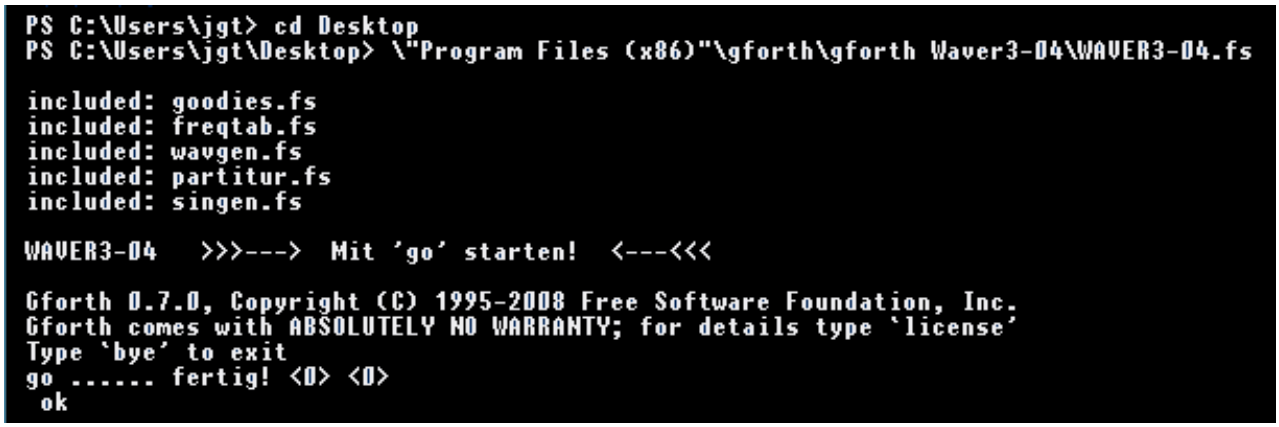


Abbildung 2: Aufruf und Ablauf des Programms Wave Engine

GATE - Eine Art GOTO in Forth

Albert Nijhof

Die GATE-Konstruktion ist eine selbstständige flache Struktur ohne Verschachtelung, die sich wie ein Forthwort verhält. Sie ist also innerhalb von Forth-Programmen verwendbar und ähnelt einem endliche Automaten[1]*. Man kann damit auf übersichtliche Weise den an sich verpönten GOTO-Spaghetti-Code, der von dem einen oder anderen Problem verlangt wird, compilieren. Endliche Automaten sind eine gängige Art, Steuerungen zu programmieren. Eine Unterstützung in Forth ist sicher wünschenswert.*[2]

Einleitung

GATE ist ein geeignetes Hilfsmittel beim Spezifizieren oder Beschreiben einer bestimmten Art von Problemen. Es hat was an sich vom Ausfüllen von Formularen, mit dem willkommenen Zusatz, dass es ausführbaren Code erzeugt. Man wird auf die Spur einer guten Faktorisierung gesetzt, mit deren Hilfe man das Problem besser in den Griff bekommt, auch mit üblichem Forthcode.

Zuerst kommt ein Anwendungsbeispiel anhand von INTERPRET, das durch sein Vertrautsein wahrscheinlich auch ohne Kommentar gut lesbar ist. Die GATE-Worte darin sind gekennzeichnet. Um nicht durcheinanderzukommen, schreibe ich INTERPRED mit einem D.

Danach folgt eine Erklärung der GATE-Worte und am Ende steht das GATE-Programm, das man zusammen mit dem INTERPRED-Beispiel herunterladen kann. Es sollte unter ANS-Forth laufen.

Der vollständige Code steht in den Dateien, deren URLs am Ende des Beitrags angegeben sind. In diesen Dateien ist der Kommentar kürzer, und in Englisch.

Zuerst das Beispiel

- In diesem Beispiel wurden die GATE-Worte mit == markiert.
- Man tippe KWIT ein und die Sache läuft.
- Man kann KWIT wieder verlassen, indem man einen Fehler macht.
- Einfachheitshalber werden doppeltgenaue Zahlen nicht erlaubt.

Das Beispielprogramm ist weiter unten aufgelistet (Listing1). Es handelt sich dabei um den berühmten Forthinterpreter namens QUIT – und damit es keine Verwechselungen mit dem Original gibt, wurde KWIT daraus.

Um KWIT zu verstehen, ist es erforderlich, mit dem innersten Kern des klassischen Forth-Interpreters und Compilers bereits vertraut zu sein. Deshalb wurde noch ein ganz einfaches Beispiel angefügt (Listing0), das kaum Kenntnisse über die Innereien des Forth voraussetzt. Es ist die übliche simple Testroutine: Auf einen Tastendruck warten und die dazu passende Zeichenkette ausgeben. Interessant dabei ist halt die einfache Art der Fehlerbehandlung. Gibt man nicht die richtige Ziffer ein, kommt die passende Fehlermeldung, und alles geht von vorne los. So etwas lässt sich mit mit Hilfe von GATE eben wirklich

[1] *finite state machine*

[2] *Der Beitrag wurde übersetzt von Fred Behringer. Fred war seit geraumer Zeit von den hier besprochenen Zeilen auf Alberts Homepage, die dieser schon 2003 verfasste, beeindruckt. Albert hat seinen ursprünglichen Text nun überarbeitet und dem heutigen Standard angepasst.*

[3] Die Beispiele wurden übrigens für diesen Beitrag mit gforth getestet. Hat alles funktioniert. mk

[4] Ich habe den Namen *GATE* für das zustands-definierende Wort gewählt, weil *STATE* in Forth schon eine andere Bedeutung hat, das hätte Verwirrung stiften können.

einfach schreiben, ohne dass man lange über die passende Faktorisierung der Abfragen nachdenken muss[3].

Erklärung der GATE-Worte

GATE[4] ccc

Eröffne einen Zustand für den Automaten mit dem Namen *ccc*. *ccc* ist dann ein ausführbares Forthwort. Die Reihenfolge, in welcher Gates definiert wurden, tut nichts zur Sache. Sobald ein Gate eröffnet wurde, kann man es hinter *GOTO* verwenden.

== ccc

Definiere die Vorschriften (rules) für ein zuvor eröffnetes Gate *ccc*. Ausführung eines undefinierten Gates erzeugt eine Fehlermeldung. Ein Gate besteht aus mehreren Vorschriften. Es gibt vier Sorten von Vorschriften:

1. ... -IF- ... GOTO ccc
2. ... -IF- ... READY
3. ... GOTO ccc ...
4. READY

Eine Vorschrift mit *-IF-* (1 oder 2) kann niemals als Letztes auftreten. Eine Vorschrift ohne *-IF-* (3 oder 4) ist automatisch Schluss-Vorschrift.

-IF-

Der Code für *-IF-* muss natürlich ein Flag erzeugen. Der Code nach *-IF-* (inklusiv abschließendes *GOTO ccc* oder *READY*) wird bedingt ausgeführt. Ist die Bedingung nicht erfüllt, springt die Maschine zur nächsten Vorschrift (nach dem Code hinter *GOTO ccc* oder *READY*). In der Schluss-Vorschrift muss *-IF-* fehlen. Automatisches Durchspringen auf eine folgende Vorschrift, die ja nicht mehr besteht, ist damit ausgeschlossen.

GOTO ccc

ccc muss ein Gate sein.

READY

READY anstelle von *GOTO ccc* dient dazu, die Struktur zu verlassen.

GATE implementieren

Zunächst werden dort einige Hilfsworte definiert (Listing2). SYNTAX und ?SYNTAX sind nicht wesentlich, sie dienen nur der Compilersicherheit.
Im GATE-Programm müssen wir dann zwei kleine Probleme lösen, die Vorwärtsreferenzen und die Rückkehradresse.

Vorwärtsreferenzen

In der Definition von INTERPRED kommen COMPILING und EXECUTING vor, aber die sind zu diesem Zeitpunkt noch nicht definiert[5]. Das lösen wir dadurch auf, dass wir GATEs schon im Voraus öffnen mit *GATE ccc*. Sicherheitshalber wird im Body von *ccc* einstweilen das Token von *0-GATE* platziert. Später wird mit *== ccc ...* *;* eine noname:-Aktion definiert. Deren Token wird dann in den Body von *ccc* gelegt.

Rückkehradresse

GOTO ccc beinhaltet keine Rückkehradresse. Es darf also keine Verschachtelung auftreten. *R> DROP* würde in vielen Forths funktionieren, um zum vorhergehenden Wort zurückzukehren. Wir wählen aber eine Lösung, die in allen Standard-Forthsystemen arbeitet: Was tut *GOTO ccc* während des Compilierens? - Die Body-Adresse von *ccc* wird als Literal compiliert. - Danach wird *EXIT* compiliert. Das steckt in *IFORNOIF*. Während der Ausführung erscheint also die Body-Adresse auf dem Stack. Sodann beseitigt *EXIT* die Rückkehradresse auf saubere Art. Nun müssen wir noch dafür sorgen, dass danach ein *@ EXECUTE* ausgeführt wird. Motor dafür ist die *does*-Routine von *GATE*. Man wird vielleicht

```
: GATE ... DOES> @ EXECUTE ;
```

erwarten. Es ist aber

```
: GATE ... DOES> BEGIN @ EXECUTE ?DUP 0= UNTIL
;
```

Warum dieses *0= UNTIL* ? Das Wort *READY* legt anstelle einer Body-Adresse eine Null auf den Stack. Damit wird der Does-Motor abgeschaltet.

Listings

```
1   \ Listing1 - Das Beispielprogramm
2
3   : snumber ( a n -- x )  \ String -> einfachgenaue Zahl
4     over c@ [char] - =
5     over 1 >      and >r  \ Minuszeichen?
6     r@ if 1 /string then
7     0 0 2swap >number
8     nip or abort"  What's this? "
9     r> if negate then ;
10
11  : ?stack
12    depth 0< if -4 throw then ;
13
14  gate interpred
15  gate compiling
16  gate executing
17
18  == interpred
19  state @ -if- goto compiling
20  goto executing
21
22  == executing
23  bl word dup c@ 0= -if- drop ?stack ."  ok!" ready
24  find              -if- execute        goto interpred
25  count snumber                         goto executing
26
27  == compiling
28  bl word dup c@ 0= -if- drop ?stack ."  ok?" ready
29  find s>d          -if- drop compile,  goto compiling
30  ( xt imm? )       -if- execute        goto interpred
31  count snumber postpone literal        goto compiling
32
33  : kwit
34    begin interpred refill 0= until ;
35
36
```

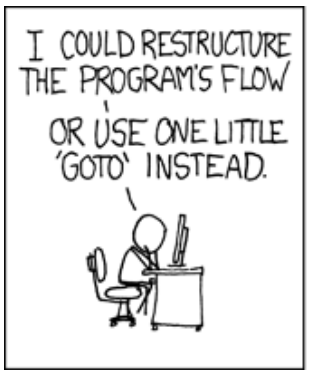
[5] Dieser Text stammt aus dem Jahre 2003 als DEFER noch kein Standardwort war. (AN, 2015)

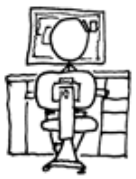
```
\ Listing2 - Das GATE-Programm


\ Zuerst ein paar Hilfsworte:

forth definitions decimal
: }
  postpone exit postpone then ; immediate

vocabulary fsa   fsa also definitions
here abs   constant syntax          \ Erkennungszahl

: ?syntax  ( x -- )
  syntax <> abort" syntax error " ;
: 0-gate
  cr .s true abort"  Undefined gate " ;

\ Bestimme die Body-Adresse von Gate ccc
: gatea    ( ccc -- body )
  ' >body dup cell+ @ ?syntax ;

: ifornoif ( pos/neg -- )
  0< if postpone } syntax } postpone ; swap ! ;




\ Und nun die GATE-Worte:

forth definitions
\ Eröffne Gate ccc
: gate  ( ccc -- )
  create ['] 0-gate , syntax ,
  does>  begin @ execute ?dup 0= until ;

\ Definiere die 'Rules' für Gate ccc
: ==    ( ccc -- gatea xt syntax )
  gatea :noname syntax ;

: -if-  ( syntax -- -syntax )
  ?syntax postpone if syntax negate ; immediate


: goto  ( gatea xt -+syntax ccc -- )
  dup abs ?syntax
  gatea postpone literal ifornoif ; immediate

: ready ( gatea xt -+syntax -- )
  dup abs ?syntax
  postpone false ifornoif ; immediate

previous forth
\ --- Ende ---
```

```
\ Listing0 - simple example (UHO 20150507)

  Gate one
  Gate two
  Gate three

  == one
  ." press 1 "  key [char] 1 = -if- ." fine " goto two
  ." no! "
  goto one

  == two
  ." press 2 " key [char] 2 = -if- ." fine " goto three
  ." no! "
  goto one

  == three
  ." press 3 "  key [char] 3 = -if- ." done " ready
  ." no! "
  goto one

\ Type ONE to start program
\ Type 1 2 3 to exit program

\ tested ok with gforth , mk 14.05.2015

cr .( done )
```

Links

http://home.hccnet.nl/anij/c/c213-gate.frt
http://home.hccnet.nl/anij/c/c213-gate-example.frt





ZF, Turbo-Forth und der RetroPie

Fred Behringer

Ich bringe einen schnellen vorzeitigen Bericht über meine allerersten Erfahrungen mit der Raspi-Mini-PC-Emulator-Komplettlösung der Firma Watterott [8] und mache ein paar Vorschläge zur Verwirklichung eines meiner Ziele (mit 6 RPi-emulierten x86-Maschinenbefehlen Willi Strickers High-Level-Basisliste [7] auf dem Raspberry Pi zum Laufen zu bringen).

Der RetroPie

Es hat mich 95,– Euro gekostet, aber die Freude darüber war unbezahlbar: Eine Komplettlösung der Firma Watterott. Von meiner ersten Informations-Anfrage bis zur ersten Inbetriebnahme sind (nur) genau sieben Tage vergangen. Das DHL-Päckchen von Watterott enthielt:

- Raspberry Pi 2 Modell B - ARM Cortex-A7 Quad-Core 900MHz 1GB RAM
- RPi-Display B+ - 2.8" Touch-Display für Raspberry Pi B+ und 2
- TEK-BERRY B+ Gehäuse (mit Ausschnitt für RPi-Display)
- 5V/2A microUSB-Netzteil (HNP10I-MICRO-USB)
- 8GB microSD Karte mit RetroPie

Was ist RetroPie? Ein Emulator-Image, das den Gamemern erlaubt, Old-DOS-Games der Heimcomputerzeit auf dem Raspi laufen zu lassen. Zugegeben, ich habe ganz und gar nicht vor zu game-en. Aber der Emulator enthält auch beispielsweise rpix86 [1] mit dem auf DOS 5.0 [6] fußenden 4DOS 7.50 [4] - und da wurde ich hellhörig!

Eigentlich aufmerksam bin ich darauf geworden über das diesjährige c't-Wissen-Sonderheft "Raspberry Pi" [2]. Aus dessem Editorial: Was ich (der Editor) allerdings weiß: Das Basteln mit dem Raspi macht einen Heidenspaß. ... All das gibts quasi umsonst. ...

Gutes altes Forth auf dem Raspberry Pi laufen lassen

gForth z.B. auf dem Raspberry Pi ist eine gute Idee und hat meine volle Bewunderung. Ich möchte das Ganze aber gern eine Nummer kleiner haben. Unter Turbo-Forth oder ZF - jenen Systemen, mit denen ich von Anfang meiner außerberuflichen Beschäftigung mit der digitalen Welt an vertraut war und zu denen ich mir nötigenfalls x86-Maschinencode (oder ARM-Emulationen davon) schnell hinzukonstruieren kann.

Völlig losgelöst von den bisherigen Riesenkästen, die den Namen "PC" tragen. Mit deren bombastischen Festplatten, mit den Disketten-Laufwerken, mit den Grafik- und Ethernet-Einsteckkarten, mit dem unmöglich großen Netzteil - und mit all dem Pipapo und Strippengewirr. Aber auch losgelöst von den allgegenwärtigen "Handys". Der ganze Raspi passt in meine Jackentasche - und ist fast genauso handy wie jedes Handy. Die Mikro-SD-Karte (also die "Festplatte" des Raspis) ist so klein, dass ich sie ohne Pinzette (allerdings: Push-push-System) kaum wieder herausnehmen kann. Das soll aber nicht als Nachteil gesehen werden. Wäre ja auch genauso wenig vonnöten wie das "Herausnehmen" einer Festplatte aus dem PC. Und 8 Gigabyte auf der SD-Karte sind doch eigentlich eine ganze Menge (!) Und 32 Gigabyte würden auch nicht die Welt kosten.

Und der Clou von det Janze: Der auf den Raspi2 aufsteckbare Lowest-Power-Mini-Bildschirm (mit Aussparung im RPi-Gehäuse)! Zwar mit einer Diagonalen von nur etwa 7 cm (Euro-Card-Quadrat), aber mit gestochen scharfen Zeichen. Eigentlich war ich ja gegen die augenverderbenden Mini-Lesegeräte der jungen Leute in der S-Bahn - aber beim RPi-Display der Firma Watterott verliere auch ich meine Vorurteile.

Inbetriebnahme

"Ganz leichte Inbetriebnahme (Out-of-the-Box)"? Hm! Ganz egal, wie groß das gesammelte jahrzehntelange Vertrautsein mit elektronischen Dingen war, stand es auch hier nicht zu erwarten, dass die Inbetriebnahme auf Anhieb klappt: Ich hatte es zunächst unachtsamer Weise fertiggebracht, den auf der unteren Seite des RPi-Displays angebrachten Verbindungspfosten nur auf die obere Reihe des RPi-Pfostensteckers zu setzen. Von außen nicht sofort erkennbar. Zum Glück für den Augenblick war mir dieser Fehler vor langer Zeit bei einem Flachbandkabel eines PC-Disketten-Laufwerks schon einmal passiert. Und als der RPi-Bildschirm dunkel blieb, fiel mir mein damaliges Missgeschick schlagartig wieder ein: Fehler gefunden. Und dann war da beim Booten des Retropies noch das verflixste y im "werksseitig" eingestellten Password. Prompt habe ich dafür - noch ohne jegliche Konfiguration zu beachten - das y der deutschen (also nicht der amerikanischen) Tastatur eingegeben. Aber das fiel mir beim Durchgooglen der Fehler anderer dann allmählich auch selber wieder ein.

Ausblick

Kurzum, ich stehe jetzt auf der Höhe des Eingabe-Prompts des Retropie-Systems, bin überaus begeistert und bin sehr zuversichtlich, was mein allernächstes Ziel betrifft: Ich möchte auf dem Faszinationsobjekt Raspberrypi2 mit 6 ARM-Forth-Primitives (ARM-Maschinenbefehle [3, 5]) und sonst nur emulierten x86-ZF-Colon-Definitionen die High-Level-Basis-Liste unseres vorjährigen Drachenpreisträgers Willi Stricker [7] nachvollziehen.

Natürlich hat mein Vorhaben, statt der 26 Forth-Primitives aus [7] mit nur 6 Primitives auszukommen, nicht unbedingt etwas mit ARM-oder-nicht-ARM zu tun.

Das Problem "26 oder nur 6" ist aber eine ganze Stange komplexer als mein vorliegender kurzer Freudenausbruch über den Watterott-Emulator. Ich habe aber die berechtigte Hoffnung, das 6statt26-Problem in nächster Zeit auch noch zu einem guten Abschluss zu bringen.
Das RPi-Display von Watterott ist ein Touch-Display. Ich habe - nicht, weil ich es dringend brauche, sondern nur so aus Neugierde - auf den Retropie-Prompt hin den Befehl *startx* eingegeben und ein paar Mal auf den Minibildschirm getatscht. Da tut sich tatsächlich etwas! Aber das interessiert mich im jetzigen Moment nicht ganz so stark.
Weiter bin ich mit meinen Experimenten noch nicht ;-). Aber ich bleibe am Ball. Versprochen!

Referenzen

[1] Aalto, Patrick: rpix86. DOS-Emulator. Diverse Links siehe Internet.

[2] c't-Wissen-Sonderheft "Raspberry Pi", 2015.

[3] Li, Jin-Fu: ARM Instruction Sets. PDF-Datei, 30 Seiten. Einprägsame Darstellung. Siehe Internet.

[4] Meinhard, Klaus: http://www.4dos.info/v4dos.htm . Prima Zusammenstellung (fast) aller Entwicklungs-Stadien von 4DOS!

[5] Pfister, Rolf: Programmieren in Maschinensprache auf dem Raspberry Pi. Teil1, Teil2, Teil3, Teil4: Assemblerfreier From-Scratch-Zugang. Links siehe Internet.

[6] Softonic: http://en.softonic.com/s/ms-dos-5.0

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

[8] Watterott, Stefan: Eigentümer der gleichnamigen Firma. Link siehe Internet.

Protokoll der Mitgliederversammlung 2015

Erich Wälde

Die Mitgliederversammlung fand statt in Hannover im CongressCentrum Wienecke XI. am 12. April 2015.

Tagesordnung

- Begrüßung
- Wahl des Protokollführers
- Wahl des Versammlungsleiters
- Bericht des Direktoriums und Kassenbericht
- Wahl des neuen Direktoriums
- Übergabe des Drachens
- Verschiedenes

Begrüßung

Ewald Rieger eröffnet die Versammlung um 9:10 Uhr und begrüßt die anwesenden Mitglieder. Er stellt fest, dass die Versammlung satzungsgemäß und fristgerecht einberufen wurde.

Wahl des Protokollführers

Erich Wälde erklärt sich bereit, die Protokollführung zu übernehmen. Er wurde ohne Gegenstimmen von der Versammlung gewählt.

Wahl des Versammlungsleiters

Als Versammlungsleiter wird Heinz Schnitter vorgeschlagen und ohne Gegenstimmen gewählt. Ewald Rieger übergibt die Leitung der Versammlung an Heinz Schnitter. Heinz Schnitter stellt die Beschlussfähigkeit der Versammlung fest. Es sind 15 wahlberechtigte Vereinsmitglieder von 105 Mitgliedern anwesend, was mehr als 10% sind.
Die Tagesordnung wird verlesen und einstimmig angenommen. Änderungsanträge sind keine eingegangen. Der Verleih des Drachenpreises soll vor der Pause stattfinden, in der Pause wird zuerst das obligatorische Gruppenfoto gemacht.

Bericht des Direktoriums

Zum siebenten Mal in Folge trägt Ewald Rieger den Kassen- und Verwaltungsbericht vor.

Mitglieder

Ende 2014 zählte der Verein 107 Mitglieder. Im Jahr 2014 gab es 3 Eintritte und 4 Austritte. In diesem Jahr gab es bereits zwei Austritte.

Einnahmen ideeller Bereich	2607.00 €
Ausgaben ideeller Bereich	3948.48 €
zusammen	-1341.48 €
Vermögensverwaltung	-141.89 €
Einnahmen Zweckbetrieb	1967.00 €
Ausgaben Zweckbetrieb	1570.27 €
zusammen	+396.73 €
Vermögen	10429.30 €
im Vergleich zum Vorjahr	-1086 €

Die Ausgaben des Zweckbetriebs liegen zu niedrig, da nur 3 Ausgaben der Vierten Dimension zustande kamen und die Rechnung für die 3. Ausgabe leider erst im Jahr 2015 einging. Normalerweise wäre das ein Betrag von ca. 3000 €.
Das Vermögen wurde im Vergleich zum Vorjahr um 1086 € abgebaut.
Vermögensentwicklung: Von den ca. 18000 € im Jahr 2008 sind jetzt noch ca. 9000 € vorhanden (die VD 2014-03 schon abgezogen). In diesem Tempo wären wir im Jahr 2020 bei null angekommen. Die ursprüngliche Forderung des Finanzamts, nicht mehr als etwa 2 Jahresmitgliedsbeiträge an Vermögen zu haben, kann jetzt als erfüllt gelten. Es wäre so langsam an der Zeit, Ausgaben und Einnahmen wieder auszubalancieren.
Der Zweckbetrieb hat in den vergangenen Jahren kontinuierlich Verlust gemacht. Eine Ausgabe der VD kostet heute 4.26 €. Vier Ausgaben plus Versand belaufen sich auf 22.83 € für Empfänger im Inland bzw. 31.03 € für Empfänger im Ausland. Von 120 Exemplaren waren zuletzt 14 Belegexemplare, die meisten davon gingen ins Ausland.
Für das Jahr 2015 ist nach heutigem Wissen ein Verlust von ca. 2800 € zu erwarten. Es ist Zeit, Änderungen einzuleiten.

Forth-Magazin Vierte Dimension und Web-Präsenz

Ulrich Hoffmann
Von der *Vierten Dimension* sind 2014 nur 3 Ausgaben erschienen. Heft 2015-01 ist erschienen, Heft 2015-02 sei in Arbeit. Außerdem ist ein *Sonderheft ARM* in Arbeit. Die Vierte Dimension wird immer noch von einem Redaktionsteam erstellt. Rückmeldungen von den Lesern gibt es immer noch nicht. Die VD ist nach unserem Wissen das weltweit letzte *gedruckte* Forth-Magazin. Daher ja auch die Veröffentlichung von englischsprachigen Artikeln.
Es wäre falsch zu sagen, das Heft sei tot. Es ist aber dennoch so, dass derzeit zu wenig Artikel eingehen. Deshalb hier einmal mehr der Aufruf: Schreibt! Auch wenn ihr noch nie geschrieben habt, das Redaktionsteam bringt

das schon in Form. Auch kleine Meldungen und die Betreuung von Autoren ist eine große Hilfe. Dank geht an dieser Stelle an alle Mithelfer!
Die *Web-Präsenz* auf www.forth-ev.de ist weiterhin eine gut besuchte Seite. Jedenfalls werden Schwierigkeiten immer recht schnell per email gemeldet.
Die Webseite basiert immer noch auf der Software *geeklog*. Diese ist in die Jahre gekommen, die alte Version 1 wird auch nicht mehr gepflegt. Die Umstellung auf die Version 2 ist nicht trivial. Das Bilderplugin ist schon tot, Spam und fremde Inhalte zu beseitigen ist Arbeit.
Es gilt also, die Webseite technisch auf neue Füße zu stellen. Ein Umzug zu einem anderen Provider wird ebenfalls diskutiert.

Außendarstellung und Projekte

Bernd Paysan
Im Jahr 2014 hat der *Linuxtag* in Berlin nicht mehr in der alten Form stattgefunden, und es ist auch fraglich, ob es den nochmal gibt. Dafür war der Verein auf der *Maker Faire* in Hannover präsent, auf den *Vintage Computing Festivals* in München und Berlin, sowie auf dem Kongress des Chaos Computer Clubs *31C3* in Hamburg.
Für das Jahr 2015 wird der Verein im Juni auf der *Maker Faire* Hannover präsent sein. Das *Vintage Computing Festival* in München und die *32C3* im Dezember in Hamburg werden besucht. Außerdem wird Erich Wälde voraussichtlich nach Augsburg an die Hochschule fahren, um einen Workshop zu halten.
An *Projekten* laufen weiterhin:

- Lego Forth (Martin Bitter)
- Mecrisp (Matthias Koch, Bernd Paysan)
- Tiva Connected Launchpad (Bernd Paysan u.a.); der ethernet stack läuft bereits. net2o soll kommen. Das Ganze ist eine *Internet-of-Things*-Spielwiese.

Kassenprüfung

Die Kassenprüfung wurde am Freitag Abend (10.04.2015) von Klaus Zobawa vorgenommen. Er berichtet, dass alle Bewegungen belegt und nachvollziehbar sind. Er empfiehlt die Entlastung des Direktoriums.

Entlastung des Direktoriums

Das Direktorium wurde ohne Gegenstimme entlastet.

Wahl des Direktoriums

Das derzeitige Direktorium stellte sich zur Wiederwahl und wurde einstimmig von der Versammlung gewählt. Die Mitglieder Ulrich Hoffmann, Bernd Paysan und Ewald Rieger nehmen die Wahl an.
An dieser Stelle wird die Mitgliederversammlung unterbrochen.

Verleih des Drachenpreises

Der Drachenrat hat getagt. Der bisherige Preisträger Willi Stricker hält die Laudatio und übergibt den Drachen an den neuen Preisträger Karsten Roederer.
In der anschließenden Pause wird auch das Gruppenfoto gemacht.
Nach der Pause wird die Mitgliederversammlung weitergeführt.

Verschiedenes

Mitgliedsbeiträge Derzeit betragen die Mitgliedsbeiträge 16 € (ermäßigt) und 32 €. Die letzte Änderung war im Jahr 1998, damals wurden die Beiträge halbiert. Nach 17 Jahren wird also eine moderate Erhöhung angestrebt, um die Kosten des Zweckbetriebs zu decken. Zur Erinnerung: Ermäßigungen können beim Vorstand formlos beantragt werden. Es werden zunächst vier Vorschläge gemacht, nach einer zweistufigen Abstimmung bleibt folgender Vorschlag übrig: Die Mitgliedsbeiträge sollen von 32 (16) Euro um 8 (4) Euro auf 40 € bzw. 20 € (ermäßigt) angehoben werden. Dieser Vorschlag wird auf der Mitgliederversammlung 2016 zur Abstimmung gestellt werden.

Projekte Die o.g. Projekte (Lego Forth, Mecrisp, Tiva Connected Launchpad) laufen weiter. Neue Vorschläge: Kann Forth (*mecrisp*) auch nativ auf einem Raspberry Pi laufen? Da macht v.a. der boot loader mit der proprietären firmware für die Graphikkarte Schwierigkeiten. ONF revival? (ONF-NT). Das ist eine Wiederaufnahme der *Open Network Forth*-Implementierung von Heinz Schnitter am Beschleuniger in Garching. Die Netzwerk-Transport-Schicht funktioniert so langsam. Man kann darüber nachdenken, was man damit anfangen will. Es soll eine Mailing-Liste mit Archiv und eine Wiki-Seite geben. Das Projekt ist schon angelaufen, auch wenn es noch nicht sehr sichtbar ist. *J1-Forth*? Wird von Ulrich Hoffmann betrieben, Karsten Roederer will sich der Sache als Tester annehmen.

Tagungsbericht? Wer schreibt einen solchen?

Forthtagung2016 Carsten Strotmann macht den Vorschlag, die Tagung mit einer anderen Veranstaltung zusammenzulegen, z.B. mit dem Linux-Informationstag in Augsburg. Das würde bedeuten, dass alle Vorträge öffentlich sind, dass es einen Einführungsworkshop geben sollte, und dass wir das Programm nicht erst am Abend vorher zusammentragen. Erich und Carsten fragen Thorsten Schöler von der Hochschule Augsburg, ob das machbar ist. Heinz organisiert die Übernachtung, Bernd und Erich das Programm.

Um 12:05 Uhr wird die Versammlung von Heinz Schnitter geschlossen.

gez. Erich Wälde (Protokollführer)

Abbildung 1: Gut gelaunte Tagungsteilnehmer in Hannover 2015

Und so sah das Tagungsprogramm aus:

Freitag

- 14:00-15:00 Führung durch die c't-Redaktion
- 16:00-16:45 Gerald Wodni: Flink - Forth Uplink (Internet der Dinge)
- 16:45-17:30 Bernd Paysan: net2o command language
- 17:30-18:00 Manfred Mahlow: e4thcom-Terminal (in der ForthBox): Target-spezifische Plugins
- 19:00-20:00 Bernd Paysan: Workshop "net2o party"
- 20:00-20:30 Ulrich Hoffmann: Workshop "Neue Forth-Targets"
- 20:30-21:00 Bernd Paysan, Gerald Wodni: Workshop "Internet der Dinge"

Samstag

- 09:00-09:45 Matthias Koch: Mecrisp
- 09:45-10:05 Manfred Mahlow: e4thcom-Terminal (in der ForthBox): Cross-Assembler-Schnittstelle
- 10:30-11:15 Carsten Strotmann: So ein MIST
- 11:15-12:00 Ulrich Hoffmann: Forth im FPGA
- 13:00-13:45 Anton Ertl: Forth 2012 - der neue Standard
- 13:45-14:30 Martin Bitter: gforth - Lego EV3 - Aktoren und Sensoren

Sonntag

- 09:00-12:00 Vereinsversammlung der Forth-Gesellschaft e.V.

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 rowila@t-online.de

Gruppengründungen, Kontakte

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

μP–Controller Verleih

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

Spezielle Fachgebiete

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**$_p$

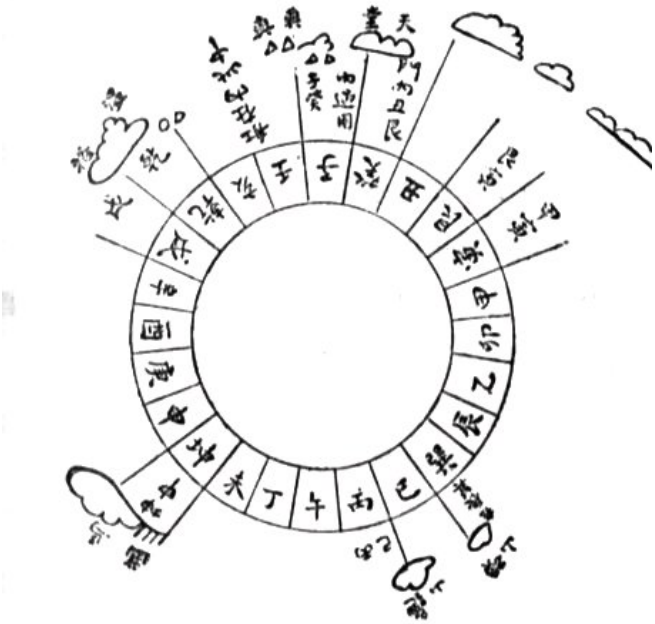
Termine

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

27.–30. Dezember 2015:
32C3 — 32. Chaos Communication Congress, Hamburg
https://events.ccc.de

12.–13. März 2016:
1. Maker Faire Ruhr
http://www.makerfaire-ruhr.com

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

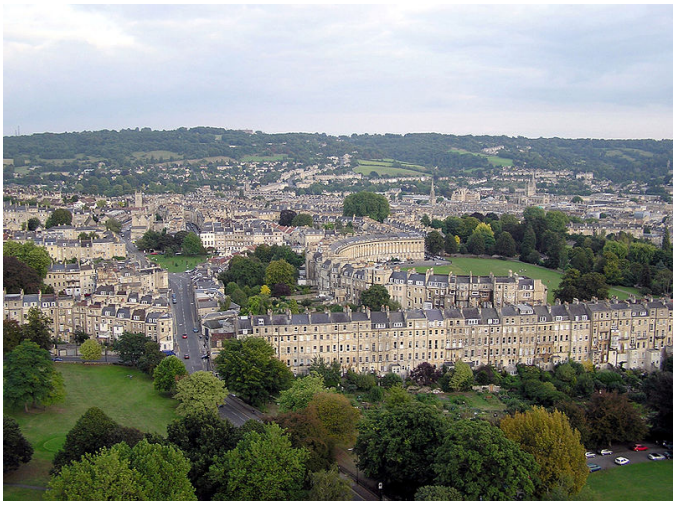


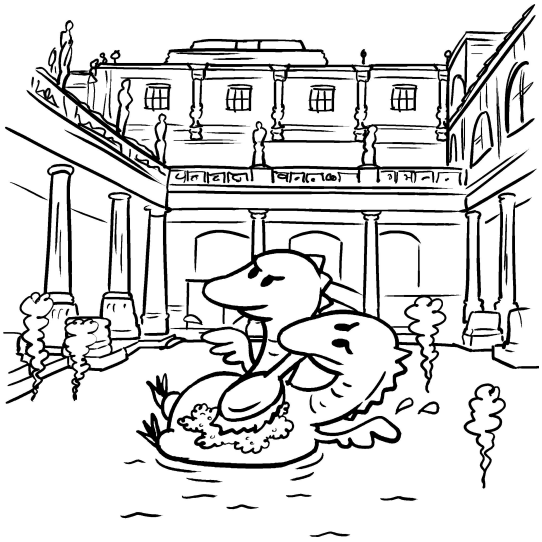
Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen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

In Pratts Hotel, Bath, England - October 2-4 2015 ...





... fand die diesjährige *EuroForth* statt — in Großbritannien! Und unser Forthmagazin lag danieder, keine Ankündigung. So hat sich das nur von Druidenmund zu Druidenohr herumgesprochen — nicht mal auf unserer Webseite war ein Hinweis. Angekündigt in `comp.lang.forth` konnte man sich jedoch auf `http://www.mpeforth.com/euroforth2015/euroforth2015.htm` und `http://www.complang.tuwien.ac.at/anton/euroforth/ef15/cfp.html` informieren und auf `http://www.rigwit.co.uk/EuroForth2015/` anmelden.

Inhalt der Konferenz (in Reihenfolge der Präsentationen): Paul Bennet: *Components of certification*, Daniele Peri: *Use of Forth to enable distributed processing on wireless sensor networks*, Nick Nelson: *A Forth programmer jumps into the Python pit*, Anton Ertl: *From exit to set-does>: A stroy of Gforth re-implementation* und *Another approach for specifying and implementing recognizers*, Peter Knaggs: *Minimal Forth*, Leon Wagner: *Forth in space*, Sergey Baranov: *A Forth-simulator of real time multi-task applications*, Andrew Read: *Hardware multitasking within a softwcore CPU*, Tobias Strauch: *Using system hyper pipelining for multi-threaded Forth compatible stack processor mapped on FPGA*, Klaus Schleisiek: *microCore and floating point numbers*, Ulrich Hoffmann: *Workshop: Forth in FPGA, Survey, multitasking and beyond*, Gerald Wodni: *FLink — The Forth Link* and *Forth-2012 Wiki*, Paul Bennett: *Forth in education — A progress report* mk/uho

Und 2016 wird die EuroForth in Konstanz am Bodensee sein ...

Wer hier lebt, hat wirklich Glück. Welche Stadt in Deutschland ist so schön gelegen, wie Konstanz? Von welchen Ufern erschließen sich so vielfältige Blicke und Perspektiven über das Wasser auf die Alpen, den Säntis, den Hegau, den Schweizer Seerücken, wie hier?

Mit der Universität - der 2007 der begehrte Titel einer Exzellenz–Universität zuerkannt wurde - und der Hochschule für Technik, Wirtschaft und Gestaltung (HTWG) verfügt Konstanz über zwei hochrangige Bildungseinrichtungen von gutem Ruf weit über die Grenzen der Region hinaus. Sie fungieren als Innovationsschmieden und Wissensvermittler für die Region. Das bewegt sich nicht nur im theoretischen Bereich. In vielen Projekten gelingt der Brückenschlag zwischen Hochschule und Lebenswelt. Viele Unternehmen in Konstanz und der Region konnten sich in der Vergangenheit nur so gut entwickeln, weil sie mit den Hochschulen kompetente Partner in der Forschung und Entwicklung, in der Ausbildung und der Nachwuchsförderung gefunden haben. Ein guter Platz für die Euroforth, der anziehende Wirkung entfalten kann. Ich jedenfalls freu mich schon darauf, im kommenden Jahr mehr darüber zu erfahren, wann das sein wird und was es dort dann alles Schönes zu bestaunen geben wird. mk