Vierte Dimension

2/2016

Das Forth–Magazin

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

In dieser Ausgabe:

Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische
Informatik

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

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

LEGO RCX–Verleih

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

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

Inhalt:

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

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

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

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

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

Ingenieurbüro
Klaus Kohl–Schöpe

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

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

KIMA Echtzeitsysteme GmbH

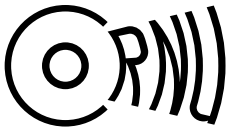
Güstener Strasse 72 — 52428 Jülich
Tel.: 02463/9967-0 — Fax: 02463/9967–99
`www.kimaE.de` — info@kimaE.de

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

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

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

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



Cornu GmbH
Ingenieurdienstleistungen
Elektrotechnik

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

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

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,

Das diesjährige Forthtreffen liegt hinter uns. Wir tagten Mitte April ja in Augsburg. Ein überschaubarer Kreis erfahrener Forth–Leute war in die Hochschule gekommen. Das Institut für Informatik hatte für uns zwei technisch gut ausgestattete Räume bereitgestellt. Einen, um sich zu versammeln, und einen für Vorträge. Vielen Dank den Organisatoren, Prof. Torsten Schöler und Erich Wälde. Zeitgleich fanden dort auch der lokale Linuxtag und die RETROpulsiv statt. Querverbindungen hat es dann aber nicht so, wie gehofft, gegeben, die Zirkel blieben doch alle unter sich. Schade.

Da dieses Heft Mitte April, schon recht gut fortgeschritten war, findet ihr hier noch keinen Tagungsbericht und nur wenig Bilder davon. Immerhin hat Martins Kursbericht es noch ins Heft geschafft, und ein hübsches Foto der Gruppe, SWAP–Drachen inclusive, neben dem diesjährigen Drachenträger, Thomas Prinz. Was der SWAP bis dahin bei dem vorherigen Drachenhüter, Karsten, alles erfahren hat, steht auch geschrieben. Aber nun mal der Reihe nach.

Dass die N.I.G.E.–Maschine aus China in der EU angekommen ist, hattet ihr ja neulich hier schon lesen können. Nun hat sie eine eingebaute Ethernet–Schnittstelle bekommen. Andrew gewährt uns dankenswerterweise wieder Einblick in diese Entwicklung. Wie gern würde ich das mal ausprobieren. Es wird Zeit, dass auch für solche Experimente wirklich sehr preiswerte kleine Platinen kommen. Matthias kennt sich damit schon aus und zeigt daher mal auf, was es da so alles schon gibt, rund um Mecrisp und um den J1a.

Kommerzieller Anwender des Forth und Vorreiter in der Entwicklung großer Systeme ist bekanntlich MPE. Stephen Pelcs Konzept, Module in Forth bilden zu können, hat Ulli aufgegriffen und in Geralds theForth.net gestellt. Spannend das. Kommen nun die Bibliotheken für MCUs in Schwung?

Was die elektronischen Netze und auch das Internet der Dinge neben all den öffentlich zugänglichen Objekten und Systemen dennoch brauchen, ist eine Privatsphäre[1]. Und so wundert es mich nicht, dass auch noch auf dem keinsten Microcontroller, der Daten verschicken kann, eine Verschlüsselung gemacht wird. Rafael zeigt, wie man damit angefangen hat. Und da DES historisch ist, sei schon verraten, dass er die anderen Verfahren im nächsten Heft zeigt.

Klaus hatte einen ganzen Koffer voll Platinen mit. Komplett alles, was an Evaluation Boards so auf dem Markt ist derzeit. Und etliche zum Mitnehmen! Und für diesen ganzen MCU–Zoo gilt: Keine MCU ohne Forth–System! Klaus hat dafür gesorgt, dass ihr da aus dem Vollen schöpfen könnt. Sein neues mcForth, hervorgegangen aus seinem bewährten KK–Forth, hat den Anspruch, nun auf all diesen MCUs verfügbar zu sein.

Und im Herbst ist dann die EuroForth–Tagung. Sie ist zurück auf dem Kontinent, im Süden des Landes, und auf einer Insel, sehr reizvoll gelegen. Ich wollt', ich könnt' dort auch dabei sein. Mal sehen.

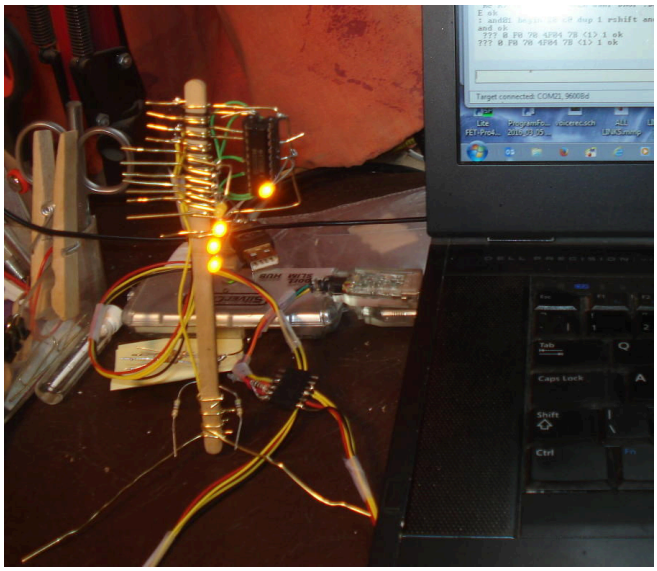
Bis dann, Euer Michael

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
http://fossil.forth-ev.de/vd-2016-02

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

[1] z.B.: Glenn Greenwald, Why privacy matters; TedTalk oct2014

Urchin — der Igel



Braucht man wirklich eine gedruckte Schaltung (PCB), um den *MicroBox*–Schaltplan aufzubauen? Natürlich nicht. Hol dir einen Kaffee im Büro, heb den Rührstab auf, nimm ein paar Büroklammern, einen 33K–Widerstand, den TI–MSP430G2553 mit Forth drin, einen Sockel dafür und 5 LEDs. Ein bisschen Kabel und einen 4–6 Pin–Stecker zum USB–TTL–Interface, oder die relevanten Pins vom TI LaunchPad. So, ein bisschen biegen und löten — dann noch 2 zusätzliche Widerstände, die als Schalter an IN0 und IN1 und gegen Masse fungieren. Das ist schon alles. Lade den uMMT–Code mit Hilfe der 4e4TH–IDE in den Chip und fertig. Getestet und geht. Ein schönes Projekt für einen verregneten Sonntag. JP

FIG UK 2016

Es hatte in Großbritannien eine rege Forth–Interest–Gruppe (FIG) gegeben, aber es wurde etwa 2006 sehr ruhig. Nach Gesprächen mit Paul Bennet , Jan Coombs und Stephen Pelc gab es nun wieder genug Interesse an einem neuen Versuch. Die Kommunikation ist viel einfacher heute — skype und google hangout z.B.

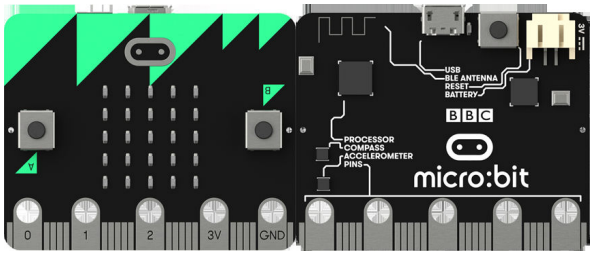


Abbildung 1: Micro:Bit

Und das *BBC–MicroBit–Board*, das jetzt an Schulen in Großbritannien geliefert wird — es gab heute[2] einige Videos darüber bei der BBC — könnte dieser Gruppe helfen, das ein bisschen vor Ort mit Forth zu unterstützen.

Unser eigenes Projekt *MicroBox* vom vergangenen Jahr mit den Pfadfindern ist auch in Richtung MicroBit gezielt — als MicroBox IO, die als Schutz zwischen dem MicroBit und externer HW sitzt. Senkt das Risiko, denn bei der Microbox kann man den Chip ersetzen. MicroBit in SMT hingegen ist wahrscheinlich nicht einfach reparierbar.

Abbildung 2: MicroBox

Wer aus dem deutschsprachigen Raum würde an FIGUK2016 interessiert sein? Bitte eine kurze E–Mail an **juergen@exemark.com**. Und bitte weiterverteilen[3]. Vielen Dank im Voraus. JP

`http://www.bbc.co.uk/news/technology-35824446`
`https://www.microbit.co.uk/about#front`
`http://www.forth-ev.de/wiki/doku.php/en:projects:microbox:start`

Frühlingsblüher

Alle drei Architekturen von MECRISP treiben im Frühling frische Knospen, so ist diesmal gleich eine ganze Handvoll neuer Portierungen anzukündigen.

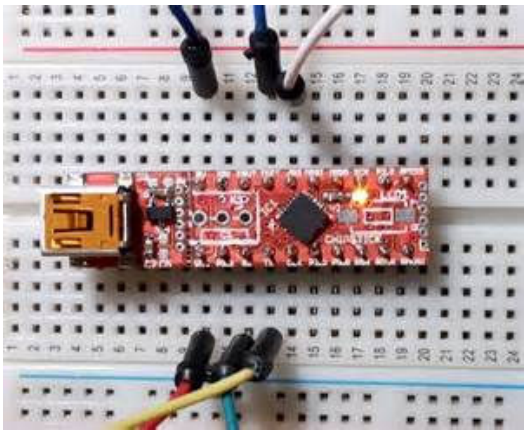


Abbildung 3: ChipStick, Ken Boak

[2] 22. März 2016

[3] Klar, da machen wir mit. mk

[4] Sustainable Suburbia, Building a „$5 Forth Computer“ — First Steps

Frisch dabei aus den Reihen der MSP430 ist der FR2433, welchen *Ken Boak* für seinen CHIPSTICK[4] gewählt hat.

In Mecrisp–Stellaris tummeln sich nun zwei Geckos: Die gut ausgestatteten und viel Programmierspaß versprechenden Silabs–Entwicklungskits GIANT GECKO und HAPPY GECKO fanden den Weg in meinen Postkasten.

Abbildung 4: Silabs Giant Gecko

Während der Giant Gecko mit einer segmentierten LCD–Anzeige und innerer Größe, sprich viel Speicher, daherkommt, bringt der Happy Gecko ein chices, reflektierendes 128x128 Grafikdisplay mit. Vielen Dank dafür — ich kann sie nur weiterempfehlen!

Wer das FREEDOM–KL25Z mag, wird sich sicherlich über die größeren, frisch unterstützten Boards von Freescale freuen. Das KL46Z ist wie ein großer Bruder des KL25Z, während das Freedom–K64F die Antwort von Freescale auf das Tiva–Connected–Launchpad zu sein scheint und Ethernet mitbringt.

Abbildung 5: Freescale–KL46Z

In England ist momentan viel Wirbel um den BBC–MICROBIT, der aus dem altbekannten Nordic–nRF51822 mit Kompass, Beschleunigungssensor und einer 5x5–LED–Matrix besteht. Dank der schon lange vorhandenen Nordic–Portierung von *John Huberts* war es ein Leichtes, auch den Microbit zu unterstützen. Der Chip an sich ist interessant, wer mag, kann mit dem Radio–Modem experimentieren und sich an Bluetooth versuchen — das Board ist allerdings schwer zu bekommen, das Fehlen eines Schaltplanes macht die Entwicklung mühselig und die ungewöhnliche Art der Kontaktierung ist vermutlich nicht jedermanns Sache.

Abbildung 6: Lattice–HX8K

Die größten Neuigkeiten jedoch sind in den jüngsten Spross, MECRISP–ICE, geflossen: Nachdem der proppenvolle HX1K–FPGA des Icesticks nun auch einen Timer–Interrupt besitzt und damit zum ersten Mal ein J1a mit Interrupts das Licht der Welt erblickt hat, ist für diejenigen, die gerne noch eigene Logik hinzufügen möchten, nun auch der HX8K mit Forth erobert. Kurz: Doppelt so viel Speicher, viel mehr IO–Leitungen, das Ganze gewürzt mit einem Barrel–Shifter und Multiplikation in einem Takt — dazu viele, viele freie Gatter für eigene Experimente. Genau der richtige Einstieg für alle, die sich in Verilog[5] austoben möchten.

Matthias Koch

Und hier seht ihr mal, wo das alles herkommt, heutzutage ...

[5] Verilog HDL ist neben VHDL die weltweit meistgenutzte Hardwarebeschreibungssprache.

Developing an Ethernet Controller for the N.I.G.E. Machine

Andrew Read (andrew81244@outlook.com)

In this article I would like to walk you through my project to develop Ethernet capability on the N.I.G.E. Machine[1]*. Many of you probably know a great deal more about Internet protocols and Ethernet standards than I do, and actually my project is still work–in–progress, so I am not trying to present a tutorial or a model Ethernet solution. Rather I would like to share the story of my development process, focusing on how I took advantage of an FPGA*[2]*, the N.I.G.E. Machine, and of course FORTH, as tools for rapid prototyping.*

Introduction

The Digilent Nexys 4 board includes a number of peripherals connected to the Xilinx Artix 7 FPGA, including a LAN8720 Ethernet transceiver IC and an RJ–45 wired–Ethernet connector (fig. 1).

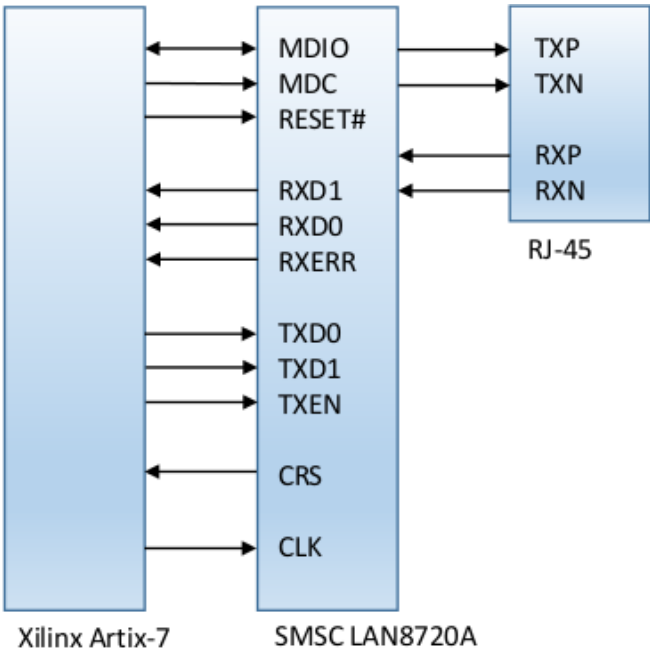


Figure 1: Ethernet resources on the Digilent Nexys 4

Referencing the model Internet protocol stack (fig. 2), the LAN8720 provides the functionality of the "PHY" layer which handles the physical transmission of signals over an Ethernet cable, including such matters as appropriate voltages, bit encodings and transmission rates. Connecting to the FPGA, the LAN8720 provides a standard "RMII[3]" interface for the reception and transmission of datapackets. The RMII incorporates 2–bit wide receive and transmit signal buses, various control lines, and a management interface. For 100 Mbits/s Ethernet communication ("100BASE–TX"), the RMII is clocked at 50MHz.

The next layer in the Internet protocol stack is the "Link" layer, which is responsible for point–to–point communications across a single link, including "media access" control protocols to manage how multiple end points share a common media. The Link layer also includes the definition of link layer address formats ("MAC addresses"). Hence the component providing link layer functionality is usually known as a "MAC controller". From the MAC controller, datapackets are passed up the protocol stack to the Network layer which provides IP address functionality, then to the Transport layer that typically provides TCP[4] or UDP[5]. Finally the application gets communication access via one or more abstract TCP or UDP ports.

Typically the Link layer MAC controller is implemented in hardware while the Network and Transport layers are implemented in software. My intention with this project is the same: firstly to develop a MAC controller in VHDL that will be instantiated as built–in peripheral of the N.I.G.E. Machine and after that develop the remaining layers in FORTH, including them as part of the N.I.G.E. Machine's system software. The present article focuses only on the MAC controller. Although this is a hardware component, its development and prototyping was actually done partly in hardware and partly in FORTH, as I will describe.

It may be worth mentioning that bespoke development of a MAC controller would not be the default option for a commercial project. A ready–made MAC core is available within the Xilinx tool set as IP. Why didn't I use it? Actually, at least for the version of the tools that I am using, it requires a paid–for license. In addition, as with many advanced FPGA IP modules, it would have been vastly over–specified for the simple requirement I have with the N.I.G.E. Machine, with the accompanying burdens of footprint and configuration complexity. And

[1] *The N.I.G.E. Machine is a user–expandable micro–computer system that runs on an FPGA development board. The key components of the system include a stack–based softcore CPU and a FORTH software environment. The system is currently hosted on a Digilent Nexys 4 development board.*

[2] *Field–programmable gate array*

[3] Reduced media–independent interface is a standard which was developed to reduce the number of signals required to connect a PHY to a MAC

[4] Transmission Control Protocol

[5] User Datagram Protocol

perhaps most importantly, had I used an off–the–shelf module I would have been denied the pleasure of developing my own!

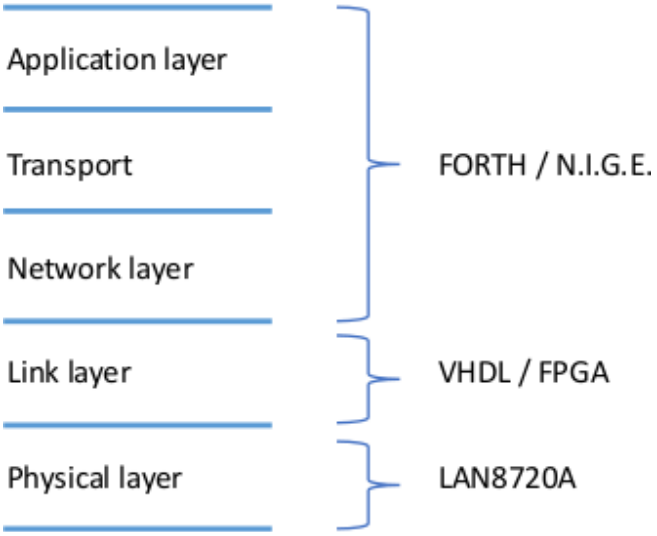


Figure 2: Ethernet technology stack model, an informal representation

First steps

At the start of this project I had no prior knowledge of Ethernet communications. Wikipedia and assorted internet pages were useful for getting oriented, but I soon needed better context (to understand how each signal/module/layer etc. fitted into the bigger picture) and much more detailed technical information. The LAN8720 datasheet thoroughly describes the operation of this IC but it assumes prerequisite knowledge of the RMII interface. I obtained a couple of excellent textbooks [1, 2] (after a returning a number of others disappointed) that provide a top–down view of Internet, but which necessarily stop short of implementation details. The Xilinx MAC core IP user–guide was also a useful resource that helped me visualize the "end–image" of what I was seeking to develop.

With different information resources covering different aspects but no single, comprehensive guide, I felt that my first step before designing anything had to be to establish a foundation of concrete knowledge about the actual system I would be interfacing with. So I decided to monitor the signals of the RMII port directly as the LAN8720 received datapackets from an Ethernet connection. My tool for this task was a Bitscope BS–10 mixed–signal PC oscilloscope [4] (fig. 3).

[6] First in, first out

Figure 3: The Bitscope BS–10 PC oscilloscope

The Bitscope has 100MHz bandwidth for analog waveforms and 25ns resolution for digital logic capture. These are impressive capabilities in a small and modestly–priced device, but I didn't anticipate that I would be able to inspect the 50MHz RMII data transmissions directly with the Bitscope. For this reason I prepared a small project for the FPGA that implemented two 1–bit FIFO[6] buffers (one for each RMII data bit), with a 50MHz input clock and a 1MHz output clock. The output side of the buffers was routed directly to FPGA pins connected to the Digilent "PMOD" expansion connectors. The FIFO buffers were instantiated as Xilinx IP components view the IP core wizard. Additional logic on the output side monitored the FIFO for the presence of data via the "Empty" signal and cycled the port via the "Read Enable" signal. The "Write Enable" port on the input side was connected directly to the "Carrier Sense / Data Valid" (CRS) signal of the RMII (fig. 4). Just to note for clarity, at this stage the N.I.G.E. Machine itself was not instantiated on the Nexys 4.

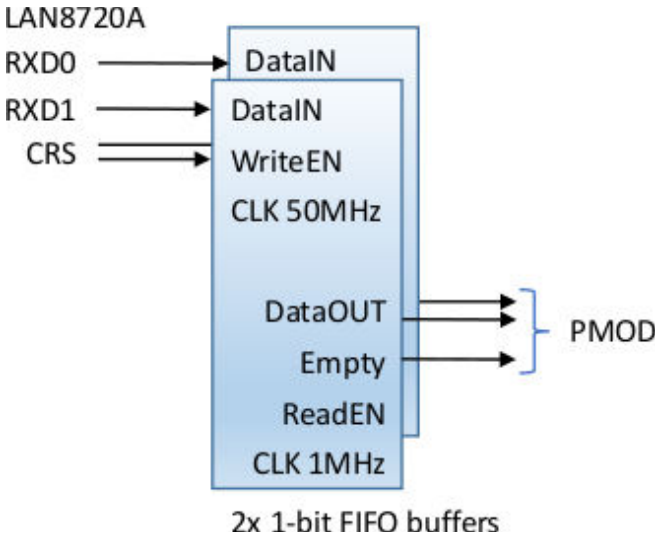


Figure 4: FIFO buffers capturing RMII data at 50MHz and outputting at 1MHz

The Bitscope's digital logic inputs were connected to the PMOD port via a small breakout connector and hookup

wires (fig. 5). To provide a source of Ethernet packets I connected the RJ45 jack on the Nexys 4 to a second Ethernet port on my PC using an Ethernet cross–over cable.

Standard "blue" Ethernet cables are appropriate for connecting end devices to a switch or router since the switch itself will correctly match the receive and transmit wires. When connecting end devices directly, a crossover cable, typically red in color, is needed to match the receive lines of one device to the transmit lines of the other device and vice–versa.

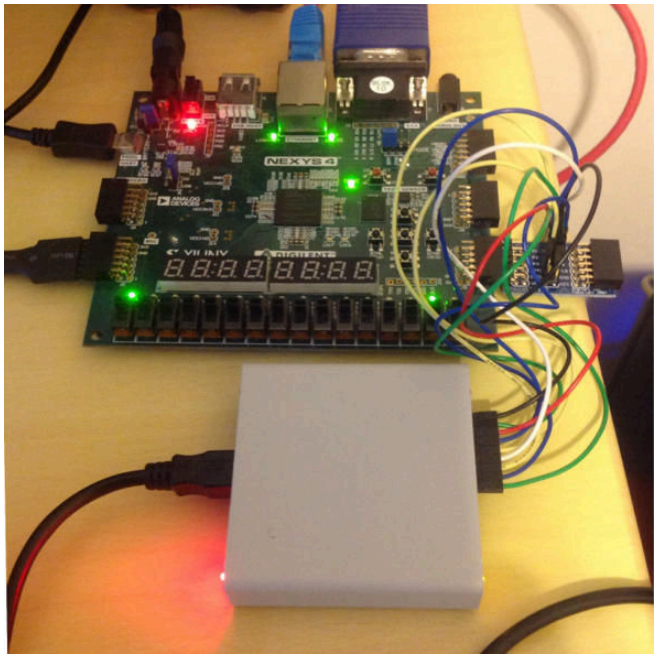


Figure 5: Bitscope connected to the Nexys 4

I had no way of controlling what IP packets the PC might be sending over the second Ethernet port but I assumed that there would be a baseline of traffic even if my own device was not reciprocating with datapackets at this stage.

This first test was a success! The Nexys 4 board configures the LAN8720 at start–up through various pull–ups and pull–downs in auto–negotiation mode. In figure 5 two green LED's are visible on either side of the RJ45 jack. The LED on the left is lit to indicate a link is established at the PHY protocol later, and the LED on the right is lit to indicate 100BASE–TX mode has been auto–negotiated. A Bitscope trace (fig. 7) shows CRS going high while an Ethernet packet is being received, followed by the output of data from the FIFO buffers. However at this stage nothing was intelligible as I had no knowledge of the Ethernet frame contents.

[7] Application–specific integrated circuit

[8] Universal asynchronous receiver/transmitter

The Wiznet serial–to–Ethernet module

Before designing the MAC controller I needed to take the monitoring one step further, to be able to consistently decode known Ethernet datapackets based on the observed RMII signals. For this I took advantage of a Wiznet WIZ550S2E–232 serial–to–Ethernet module and the accompanying development board [5] (fig. 6). The Wiznet modules include a complete TCP/IP stack inside a custom ASIC[7], the Wiznet W5500. The W5500 is commonly used on wide variety platforms such as the Arduino Ethernet shield and various ARM microcontroller boards.

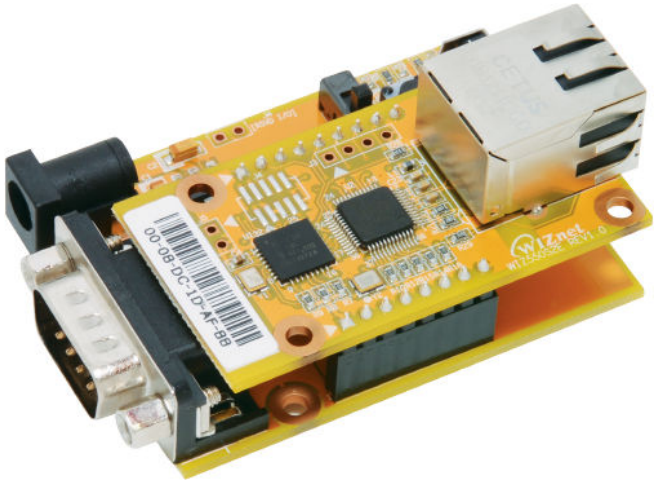


Figure 6: Wiznet serial–to–Ethernet module and development board

After connection I used the UART[8] facillity on the Wiznet module to command datapacket transmission to the Nexys 4. Listing 1 — Wiznet.txt — is an example communications with the Wiznet. For brevity the Wiznet responses are not shown. The comments indicated by \ are annotations that were not actually transmitted.

Figure 7 shows an Ethernet frame sent by the Wiznet and captured by the Bitscope. There are a few features to notice. Logic 0 is the carrier sense / data valid signal from the LAN8720. It remains high during Ethernet data reception. Logic 2 is driven with the clock on the output side of the FIFO buffer and logic 3 and logic 4 are signals RXD0 and RXD1. Looking at the first part of the trace it is apparent that the signal being received is a series of alternating 1's and 0's, since RXD0 remains high while RXD1 remains low. This is consistent with the Ethernet frame's preamble phase (fig. 9). A series of 1's follow where both RXD0 and RXD1 remain high. These correspond to the last two bits of the start of frame delimiter and a destination MAC address of FF FF FF FF FF FF. Continuing on, the transitions in RXD0 and RXD1 reflect the data of the Ethernet frame itself. The entire Ethernet frame is longer than the trace that has been captured by the Bitscope. The duration of CRS / DV high appears very brief by comparison due to the higher clock rate on the input side of the FIFO buffer.

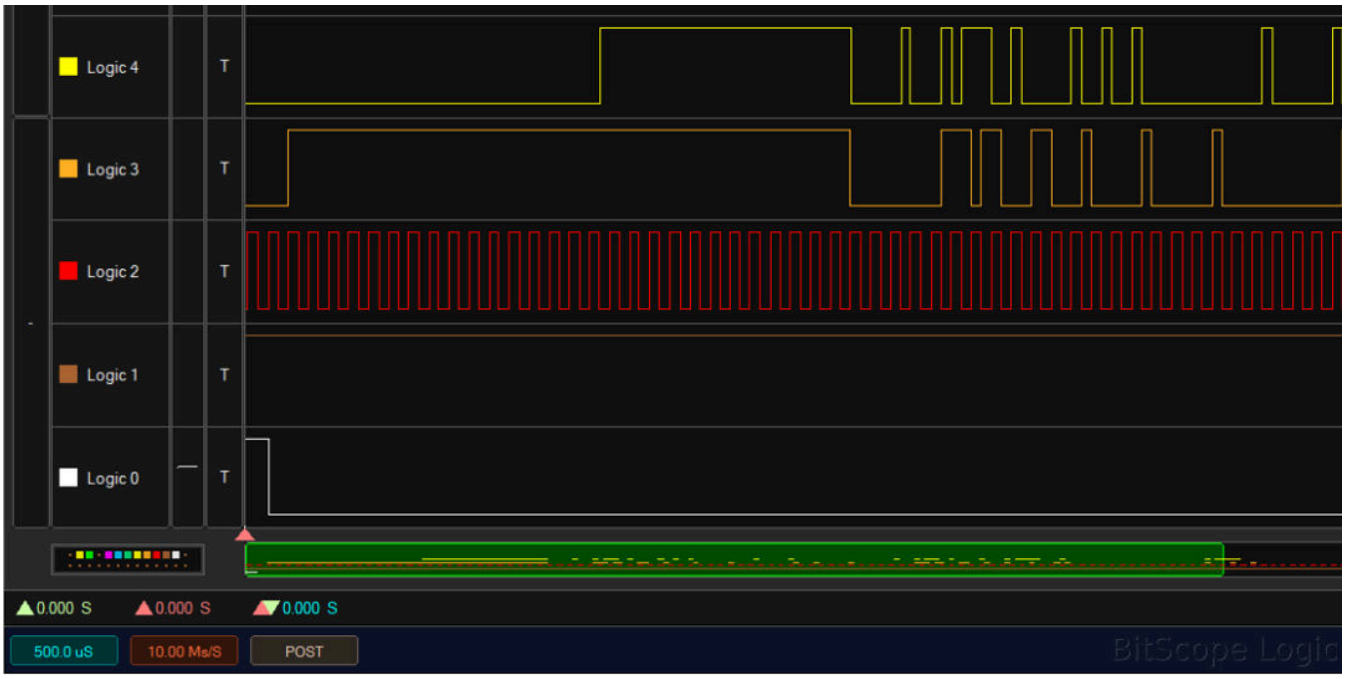


Figure 7: Ethernet frame at the RMII inspected on the Bitscope. Signals are: 0: CRS, 1: N/A, 2: FIFO clock, 3: RXD0, 4: RXD1

At this stage I was becoming confident that it would be possible to understand the signals of the LAN8720, but decoding an Ethernet frame from an oscilloscope trace is rather slow and error prone! This is where I started to involve FORTH.

Inspecting an Ethernet frame with FORTH

Until now the only circuitry instantiated in the FPGA had been the FIFO buffers connected directly to PMOD connectors and a small amount of control logic. I now imported these FIFO buffers into the N.I.G.E. Machine (fig. 8) and connected them to memory–mapped hardware registers. The control logic was also changed so that instead of outputting all data automatically, the buffers would deliver two bit of data each time the relevant hardware register was read by the CPU. Another hardware register could be read to ascertain whether the FIFO buffers were empty or not.

Listing 2 — Forth_FIFO_decode.txt — shows the simple FORTH code run on the N.I.G.E. Machine to read the RMII data held in the FIFO buffers. There is a routine to compose an octet (the name that the Ethernet standard gives to 8 bits of data) out of four 2–bit nibbles, and a simple finite state machine (FSM) to print an Ethernet frame in its constituent fields.

Figure 9 shows the structure of an Ethernet frame and the data capture by N.I.G.E. at this point. The data is the same as that in figure 7. This Ethernet frame is actually an Address Resolution Protocol (ARP) frame that the WIZNET is sending out as a preliminary to sending the datapacket to the IP address as requested in listing 1.

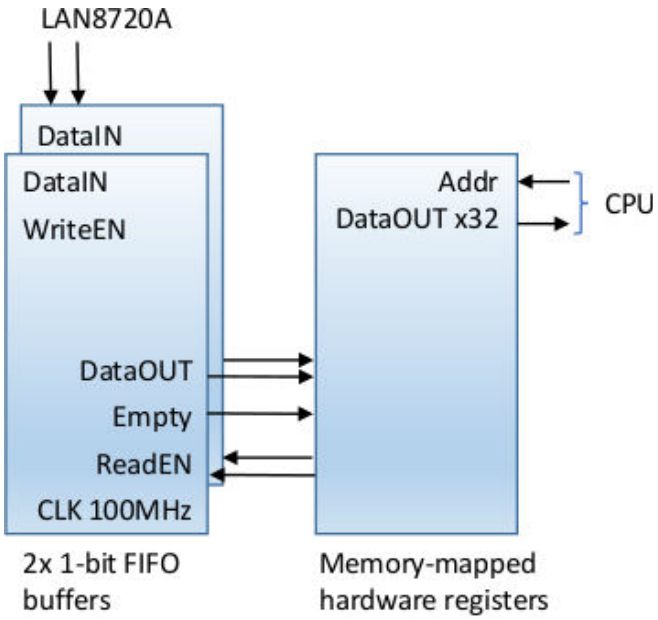


Figure 8: The RMII FIFO buffers connected into the N.I.G.E. Machine

Component	Size (bytes)	Example captured by N.I.G.E.
Preamble	7	55 55 55 55 55 55 55
Start of frame delimiter	1	D5
Destination MAC address	6	FF FF FF FF FF FF
Source MAC address	6	00 08 DC 1E 52 60
Payload type	2	08 06
Data	46 – 1500	00 01 08 00 06 04 00 01 …
CRC checksum	4	B4 BC 30 91

Figure 9: Ethernet frame generated by WIZNET and Captured by N.I.G.E.

At this stage I was successfully receiving Ethernet frames, but for prototyping purposes they were being decoded in software rather than in hardware. In addition the FSM employed was very simplistic and the Cyclic Redundancy Check (CRC) checksum was not being validated. Having reached this point it was now time to start developing a MAC receiver in VHDL. The first component that would be needed was a CRC calculator.

Preparing an CRC calculator in VHDL

A CRC calculator is most easily implemented in hardware as a shift–register with feedback. The relevant VHDL code is shown in Listing 3 — CRC_vhdl.txt. "SR" is a 32 bit shift register that contains the (inverted) checksum. "dataIN" is the new data bit provided each clock cycle. Because the RMII interface receives 2–bits per clock cycle at 50MHz, this CRC calculator processing 1–bit per clock cycle operates at 100MHz. Each clock cycle "dataIN" is combined with the shift register at various bit–positions using an XOR operation, thus performing the CRC calculation. I recommend Hank Warren's "Hacker's Delight" [3] for a full explanation of how this bit level procedure is equivalent to the CRC algorithm expressed in terms of polynomial division.

I tested the CRC calculation module using the Xilinx simulator using both captured frames (e.g. fig. 9) and example frames taken from various internet sources.

A working MAC receiver

The next stage was to create a VDHL implementation of the finite state machine for a MAC receiver that I had thus far modeled in FORTH (listing 2). I added new state transition logic to deal with the termination of a frame capture. The checksum is calculated in real time by the CRC calculator. If and when the checksum becomes equal to the "magic value" of hexadecimal 38FB2284, this indicates that a valid and complete Ethernet frame has been received. The FSM transitions to the idle state. If the CRS / DV signal from the RMII interface goes low before the magic value has been sighted, then the reception of that frame was invalid. The FSM transitions to idle but a checksum error flag is raised.

A small but critical point to note is that multiple valid Ethernet frames may be appended to each other and contained within the duration of a CRS / DV high signal. This is why it is necessary to determine the end–of–frame based on the sighting of valid checksum magic value rather than wait for CRS / DV to go low.

The MAC receiver outputs data as 8–bit octets and so the two 1–bit FIFO buffers were replaced with a single 8–bit FIFO buffer. Figure 10 illustrates the system diagram of the completed MAC receiver.

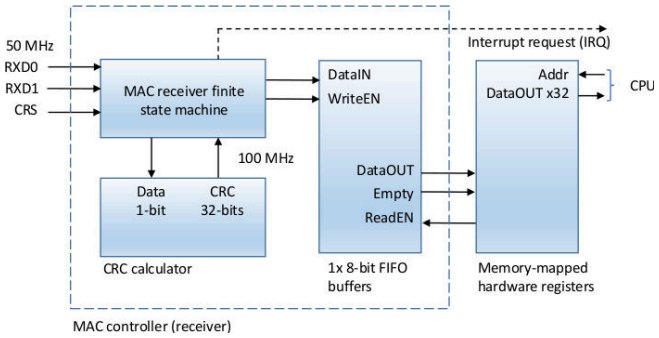


Figure 10: System diagram of the MAC receiver

At this point I enhanced the testing strategy. The FORTH code to read and print a captured Ethernet frame became much simpler. It was now just a case of reading bytes of data from the FIFO buffer through a hardware register until the buffer was empty. I wanted to inspect many more frames and also test for frame dropping. I removed the WIZNET and reconnected the N.I.G.E. Machine's Ethernet port to the spare Ethernet port of my PC. The test code on the N.I.G.E. Machine captured Ethernet frames continuously and redirected their printout to the RS232 port (Listing 4 — MAC_texting.txt). At the PC, a terminal program displayed the frames being captured by the N.I.G.E. Machine and Wireshark (PC software for Ethernet adapter monitoring) displayed the frames being transmitted by the PC. Although I did not have control over what frames were being sent, I could compare the N.I.G.E. Machine terminal and Wireshark to check if there was a good correspondence between send and received frames. The results were positive: during the testing period I did not detect any errors or dropped frames.

Figure 10 also shows a CPU interrupt request from the MAC receiver as a dashed line. The N.I.G.E. Machine has an expandable interrupt controller to receive interrupts from peripherals and vector execution according to the number of the interrupt request. During development I used a busy loop to detect the arrival of Ethernet frames, and this may still be a good solution if Ethernet frame detection is placed into a separate task with a PAUSE each iteration, but there is also the option to create an Ethernet frame arrival interrupt handler.

A working MAC transmitter in simulation

Having a working MAC receiver, I expected that the next stage to develop a MAC transmitter would be straightforward. Figure 11 shows the system diagram for the MAC controller incorporating the receiver and transmitter. Separate FSM's are responsible for transmission and reception. The transmitting FSM has its own instance of a CRC calculator to calculate the checksum of an outgoing Ethernet frame. Software on the N.I.G.E. Machine writes outgoing frame data to a transmit FIFO buffer then writes to an additional register, the transmit_request register, to initiate frame transmission by the MAC controller.

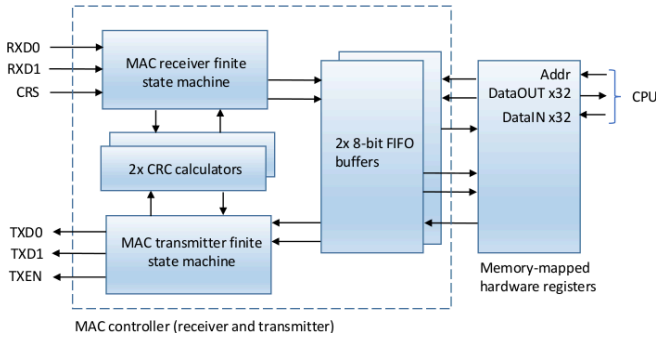


Figure 11: System diagram of the MAC transmitter and receiver

The MAC transmitter refuses to work in hardware!

I initially tested the MAC transmitter in the simulator by sending the Ethernet frame that I had captured from the Wiznet (fig. 9) and confirming that the signals generated on TXD0 and TXD1 were the same as those observed on RXD0 and RXD1 by the Bitscope (fig. 7). This test was successful. The next text, also conducted in the simulator, was to connect TXD0 to RDX0, TXD1 to RXD1, TXEN to CRS and validate that the transceiver could "loop–back" an Ethernet frame between the transmit and receive FIFO buffers without errors. This test was also successful.

Extending this test into hardware, I routed the MAC controller inputs and outputs to a PMOD and made a loop–back connection with jumpers. A subtle factor that needs to be taken account of at the hardware stage is the change in phase of the signals over the loop back with respect to the 50 MHz clock that registers the inputs. Essentially there is a finite time delay for the TX signals to route outside of the FPGA, across the Nexys board, through the jumpers, and back to the FPGA logic for the RX signals. This time delay will change the phase of the signals with respect to the 50MHz clock and needs to be considered to ensure that there are no set–up or hold time violations. Since this test was a one–off I did not make any adjustments for this. However the more correct solution would be to adjust the phase of the 50MHz clock output to the RMII interface so that set–up and hold times are within specification for both TX and RX signals. An engineer with access to a picosecond accurate oscilloscope might inspect the signals directly at the FPGA pins to confirm set–up and hold times. Without that capability, trials of different clock phases at increments can be used to find the optimum settings.

This first hardware test was sucessful. I then reconnected the MAC controller to the LAN8720 with the intention of continuing loop–back testing. With the LAN8720 in place there are two options for loop–back testing. The simplest approach is to prepare a loop–back connector from a lenght of Ethernet cable. This involves cutting an ordinary Ethernet cable and reconnecting the wires in a loop–back circuit. The loop–back cable thus fabricated can be tested on a PC running Wireshark. The alternative approach is to configure the LAN8720 in internal loop–back mode. The LAN8720 has a management interface, briefly mentioned at the start of this article. It comprises a single bi–directional serial data line and an aperiodic clock that can be used to read and write to a set of internal status and control registers. Internal loop–back mode can be instructed via a control registers.

So far loop–back tests with the LAN8720 have been unsuccessful using both of these approaches. According to the RMII specifications, the data broadcast on TXD0 and TXD1 ought to continue for as long as TXEN is held high. Currently transmissions are being curtailed after only a few bytes for reasons that I have not yet understood.

I have not yet resolved the MAC transmission issue. I have been working on other projects in the meantime (primarily a DDR memory interface destined for the N.I.G.E. Machine) and intend to return to the MAC transmitter with the benefit of fresh ideas.

Conclusions

In conclusion I would like to highlight aspects of the development approach that were particularly helpful. Firstly, the flexibility of an FPGA allowed me to instantiate temporary FIFO buffers to re–route and slow down the RMII signals so that I could make an initial inspection with an oscilloscope. Secondly, the N.I.G.E. Machine, as a FORTH microcomputer, allowed me to investigare these RMII signals interactively before developing a full MAC controller. Lastly, the stage–by–stage approach, including capturing oscilloscope traces of Ethernet packets, built up a body of data that made initial testing of the MAC transmitter much more straightforward. I believe that developing a MAC controller in isolation would have been much more difficult. This project's approach illustrates the intended application of the N.I.G.E. Machine as a platform for rapid hardware prototyping.

I hope this article has been of interest and I welcome any correspondence or suggestions. All of the N.I.G.E. Machine source code is open source [6].

Referenzen

[1] "TCP/IP Illustrated", Kevin R. Fall and W. Richard Stevens, 2nd edition revised, 2011

[2] "Computer Networks", Andrew S. Tananbaum and David J. Weatherall, 5th edition, 2010

[3] "Hacker's Delight", Henry S. Warren, 2nd revised edition, 2012

[4] Bitscope Designs, http://www.bitscope.com

[5] Wiznet, http://www.wiznet.co.kr

[6] Github, https://github.com/Anding/N.I.G.E.–Machine

Listings

```
Listing 1   -   Wiznet.txt

+++                             \ enter command mode (send no CR or LF)
AT+NOPEN=U,1000,1.1.1.1,1000    \ open a UDP connection to IP address 1.1.1.1
AT+NSEND=0,4                    \ send 4 bytes of data (to follow) to connection 0
01234                           \ 4 bytes of data
```

```
Listing 2   -   Forth_FIFO_decode.txt

hex
03F860 constant MACready
03F864 constant MACdata
decimal

: octet ( -- x)
\ fetch an octet from the MAC FIFO buffer
    0
    4 0 DO
        2 rshift     \ lo nibble received first
        MACdata @
        6 lshift
        or
    LOOP
;

0 constant #preamble
1 constant #destination
2 constant #source
3 constant #type
4 constant #other

: .frame
\ decode and display an Ethernet frame
    #preamble
    BEGIN
        CASE
            #preamble       OF MACdata @ 3 = IF #destination ELSE #preamble THEN ENDOF
            #destination    OF CR ." Dest: " 6 0 DO octet . LOOP #source ENDOF
            #source         OF CR ." Src: " 6 0 DO octet . LOOP #type ENDOF
            #type           OF CR ." Type: " octet 256 * octet + . CR #other ENDOF
            #other          OF octet . #other ENDOF
        ENDCASE
    MACready @ 0=
    UNTIL
    drop CR
;
```

N.I.G.E. News

A beta-test version of the N.I.G.E. Machine for the Nexys4DDR development board is now available in the GitHub repository. The relevant Vivado project is `Xilinx_Vivado_DDR.zip`. There is an application note explaining how to get started in the Resources directory.

```
Listing 3  -  CRC_vhdl.txt
-- CRC32 checksum calculator

inbit <= dataIN XOR SR(31);
checksum <= not SR;

process
begin
    wait until rising_edge(clk);
    if reset = '1' then
        SR <= (others=>'1');
        -- Ethernet CRC-32 sets all bits high before conversion
    else
        SR(31 downto 27) <= SR(30 downto 26);
        SR(26) <= inbit XOR SR(25);
        SR(25 downto 24) <= SR(24 downto 23);
        SR(23) <= inbit XOR SR(22);
        SR(22) <= inbit XOR SR(21);
        SR(21 downto 17) <= SR(20 downto 16);
        SR(16) <= inbit XOR SR(15);
        SR(15 downto 13) <= SR(14 downto 12);
        SR(12) <= inbit XOR SR(11);
        SR(11) <= inbit XOR SR(10);
        SR(10) <= inbit XOR SR(9);
        SR(9) <= SR(8);
        SR(8) <= inbit XOR SR(7);
        SR(7) <= inbit XOR SR(6);
        SR(6) <= SR(5);
        SR(5) <= inbit XOR SR(4);
        SR(4) <= inbit XOR SR(3);
        SR(3) <= SR(2);
        SR(2) <= inbit XOR SR(1);
        SR(1) <= inbit XOR SR(0);
        SR(0) <= inbit;
    end if;
end process;
```

```
Listing 4    -   MAC_testing.txt

: octet ( -- x)
\ read an octet from the FIFO buffer
    MACdataRX @
;

: .frame
\ display a captured Ethernet frame
    CR
    BEGIN
        MACreadyRX @ 0<>
    WHILE
            octet 2 .r
    REPEAT
        MACchecksum_err @ IF ." CRC error" CR THEN
;

: capture
\ capture Ethernet frames continuously until a key is pressed
    \ redirect output to RS232
    >remote
    BEGIN
        MACreadyRX @ 0<> IF .frame THEN
        key?
    UNTIL
    \ redirect output to the local terminal (the screen)
    >local
;
```

Modules

Ulrich Hoffmann

Das Paket **`modules`** *auf* **`theForth.net`** *definiert VFX–Forth–artige Module in Standard–Forth. Damit lassen sich nun portabel modulare Forth–Programme mit internen und externen Definitionen als Basis für wiederverwendbare Pakete formulieren.*

Motivation

Die Strukturierung von Programmen in Module gehört in vielen Programmiersprachen zur Standard–Ausstattung. Nicht so in Forth. Hier definieren wir ein Wort nach dem anderen und am Ende haben wir dabei das System so erweitert, dass es die Anwendung umfasst. Hilfsdefinition, die wir dabei (etwa wegen guter Faktorisierung) machen, bleiben für alle darauf folgenden Definitionen sicht- und verwendbar. Sinnvoller wäre es, wenn es *externe* Definitionen gäbe, die für die höheren Ebenen bestimmt wären, und *interne* Worte, die nur für die Realisierung der externen Worte gebraucht werden.

Traditionelles Forth kennt Vokabulare, die die Sichtbarkeit von Worten regeln können. Standard–Forth hat Wortlisten.

Die hier vorgestellten Definitionen realisieren das Modul–Konzept von VFX–Forth [1] auf Basis von Standard–Forth (Forth–94 oder Forth–2012) mit Hilfe von Wortlisten. Externe Definitionen werden *export*iert, alle anderen Definitionen sind intern.

Installation

Das `modules`–Paket [2] befindet sich auf `theForth.net` [3].

Nachdem das Paket heruntergeladen wurde, können Module in jedem Standard–System verwendet werden, nachdem man `INCLUDE modules.fs` ausgeführt hat. Das Paket ändert keine Standard–Worte, so dass man auch nach Laden des Pakets immer noch ein Standard–System vor sich hat.

Dokumentation

Es gibt zum `modules`–Paket ein Glossary, das die Benutzung der definierten Worte erläutert. Es befindet sich im File `glossary.md` [4] (Markdown–Format).

Beispielbenutzung

Module werden wie folgt verwendet:

```
MODULE greet
  : hello ( -- )  ." Hello" ;
  : mods  ( -- )  ." Modules" ;

  : hi ( -- )  hello ." , " mods ." !" cr  ;

EXPORT hi

END-MODULE
```

Ist diese Module–Definition geladen, stehen die `EXPORT`ierten Worte (hier `hi`) zur Verfügung. Alle anderen Worte (hier `hello` und `mods`) sind intern und nicht mehr sichtbar.

```
hi ( Hello, Modules! )
hello ( nicht zugreifbar: hello ? )
```

Die internen Defintionen sind in einer eigenen Wortliste versteckt. Diese kann man mit `EXPOSE-MODULE greet` in die Suchordnung mit aufnehmen, sollte man doch einmal interne Worte verwenden wollen.

Wenn man `immediate`–Worte exportiert, wird die `EXPORT name IMMEDIATE`–Syntax verwendet.

Fehler melden

Fehler, die im `modules`–Pakte gefunden werden, sowie Anregungen und Vorschläge, bitte an Ulrich Hoffmann <uho@xlerb.de> schicken.

Standardkonformität

Das `modules`–Paket genügt dem Forth–94– und dem Forth–2012–Standard.

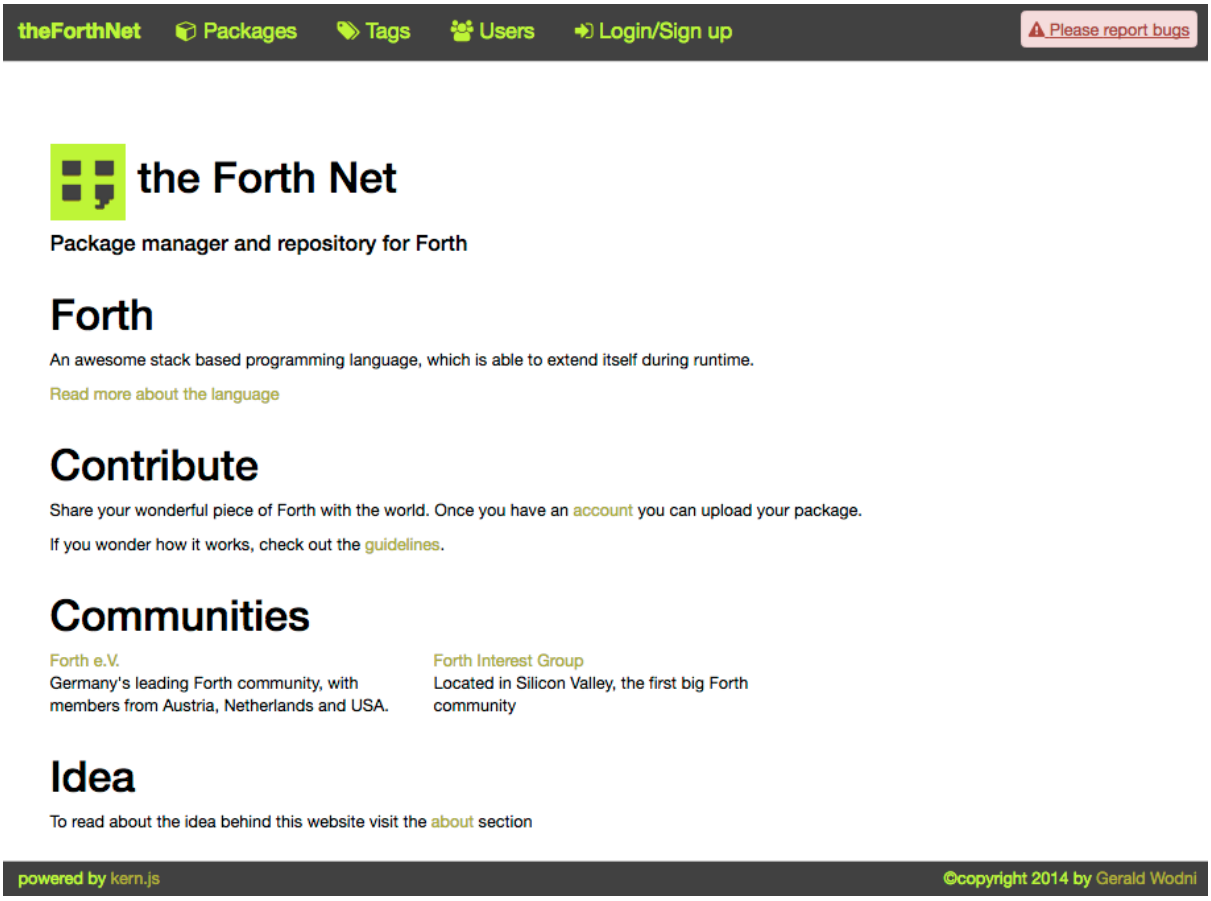
May the Forth be with you!

Referenzen

1. `http://www.mpeforth.com/vfxcom.htm`
2. `http://theforth.net/package/modules`
3. `http://theforth.net/`
4. `http://theforth.net/package/modules/current-view/glossary.md`

Listings

```
1  \ VFX like modules based on Forth-94 wordlists      uho 2016-04-16
2  \ ----------------------------------------------------------------
3
4  : MODULE ( <name> -- old-current )
5      GET-CURRENT   WORDLIST CREATE DUP >R ,
6      GET-ORDER R@ SWAP 1+ SET-ORDER
7      R> SET-CURRENT ;
8
9  : EXPORT ( <name> old-current -- old-currrent ) >R
10     >IN @  ’  SWAP >IN !  GET-CURRENT    R@ SET-CURRENT
11     CREATE SWAP , SET-CURRENT R>
12     DOES> @ EXECUTE ;
13
14 : EXPOSE-MODULE ( <name> -- )
15     GET-ORDER  ’ >BODY @  SWAP 1+ SET-ORDER ;
16
17 : END-MODULE ( old-current -- )
18     SET-CURRENT   GET-ORDER NIP 1- SET-ORDER ;
```



DES Data Encryption Standard

Rafael Deliano

Der 1977 in den USA eingeführte Algorithmus war wegweisend und ist für embedded Anwendungen auch heute noch geeignet.

Es finden sich zwar in allen einschlägigen Handbüchern Beschreibungen. Diese orientieren sich aber an der Darstellung im Standard [1], die unzureichend ist. Die Dokumentation ist hier deshalb modifiziert. Aber nur soweit, dass der Vergleich mit [1] noch möglich ist.

Der 64–Bit–Schlüssel enthält im obersten Bit des Bytes jeweils ein odd–parity–Prüfsummenbit. Es wird in den meisten Implementierungen und auch hier ignoriert. Die 64–Bit–Datenworte werden damit anhand eines 56–Bit–Schlüssels chiffriert und dechiffriert (Abb.1).

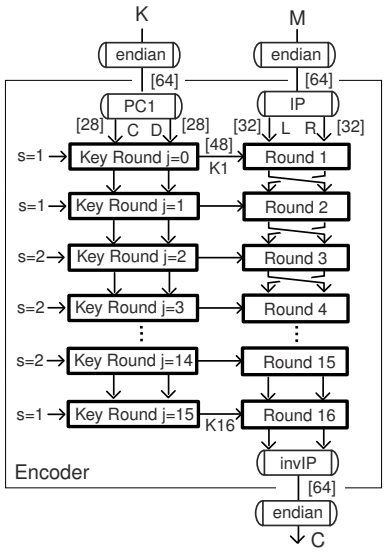


Abbildung 1: Rounds Encoder

Die Verarbeitung erfolgt sequentiell in 16 Schritten, für die Teilschlüssel berechnet werden müssen (Abb.2).

Eine Grundfunktion von DES ist die Permutation, ein Umsortieren der Bits im Datenwort. Hier dargestellt an IP (Abb.3), ein simpler Fall, in dem das Ausgangsdatenwort gleich lang ist.

In [1] ist das unterste Bit links angeordnet (Abb.3). Für Implementierung in Hardware hat das damals weniger gestört als heute. Als Nebeneffekt muss man in Software die Bitordnung innerhalb aller Bytes an Ein- und Ausgang umdrehen. In Abb.1 als endian–Permutation aufgeführt.

In der Originalversion werden die Bits auch 1,2,3 ... gezählt. Das ist hier bereits auf 0,1,2 ... umgestellt.

Permutation ist in Hardware simpel, in Software zäh. Simpelste Vorgehensweise ist, das Ausgangsdatenwort auf null zu initialisieren, das Eingangsdatenwort Bit für Bit abzuprüfen, und wenn das Bit gesetzt ist, über eine Indextabelle das Bit im Ausgangsdatenwort zu setzen. Jede der Permutationen ist damit über eine Tabelle, die die Adresse des Zielbits angibt, definierbar und implementierbar (Tab.1).

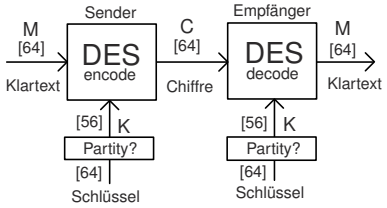


Abbildung 2: Simple Anwendung Blockchiffre

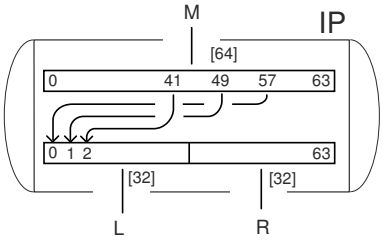


Abbildung 3: Permutation

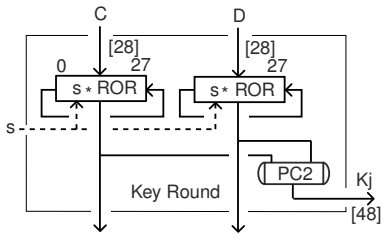


Abbildung 4: Berechnung Teilschlüssel

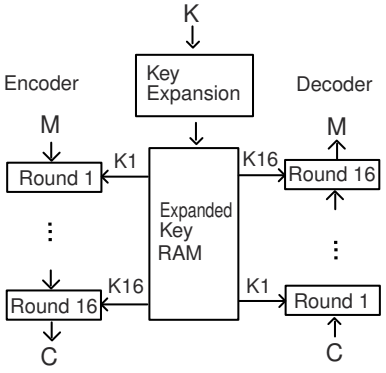


Abbildung 5: Key–Expansion

Key Expansion

In den Teilschritten (Abb.4) wird jedes Datenwort 1 oder 2 mal rotiert. Den „wie oft"–Wert s entnimmt man anhand des Indexzählers j aus der s–Tabelle (Tab.2).

In Hardware war es einfach, die Teilschritte immer neu zu berechnen. In Software erzeugt man üblicherweise erstmal den ganzen Satz Teilschlüssel, belegt 96 Byte RAM. Steht aber nun für alle folgenden Datensätze unmittelbar zur Verfügung (Abb.5).

Rounds

Die bitweisen XORs sind einfach implementierbar (Abb.6). „Expansion"–Block ist eine Permutation, die die Zahl der Bits von 32 auf 48 erhöht.

Die S–Boxen sind hier auf 8 Tabellen zu 64 Byte vereinfacht (Tab.3). Sie enthalten bereits die endian–Drehung der Datenworte an Ein- und Ausgang.

Normalerweise werden L und R anschließend vertauscht, nicht aber im letzten Durchlauf (Abb.1).

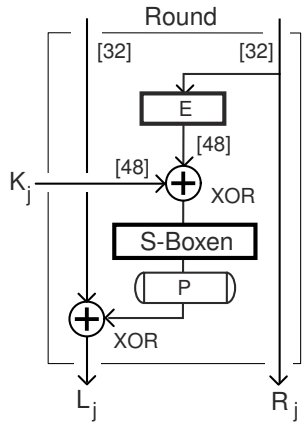


Abbildung 6: Rounds

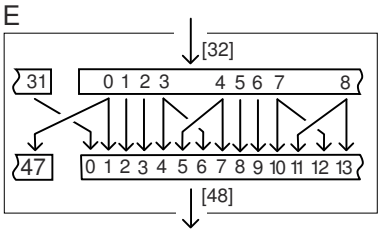


Abbildung 7: E–Permutation für Rounds

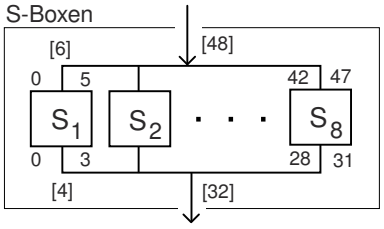


Abbildung 8: S–Boxen für Rounds

Decoder

Der ist angenehm simpel: Man muss nur die Reihenfolge der Teilschlüssel umdrehen (Abb.9).

Implementierung

Nützlich ist vor allem [4], weil es detaillierte bitgenaue Printouts eines Testlaufs enthält. Die Source in APL dürfte weniger von Interesse sein, sie ist aber erstaunlich kompakt. Umfängliche offizielle Testvektoren finden sich in [2]. Gängig und normalerweise ausreichend sind die 3 Testworte aus [3].

Gegenüber der hier erfolgten Darstellungen sind weiter Optimierungen nötig, um auf erträglich schnelle Software zu kommen. Erstens werden die Endian–Permutationen an der Schnittstelle (Abb.1) in die ohnehin vorhandenen Permutationen PC1, IP, invIP integriert.

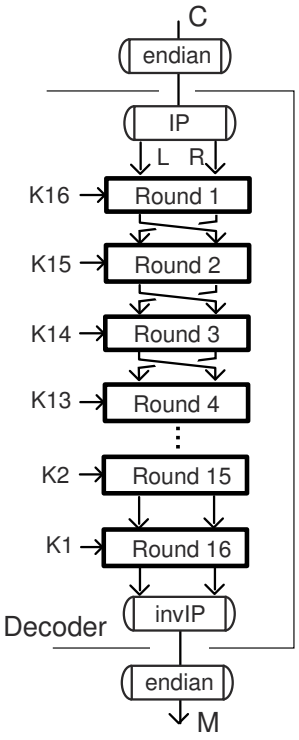


Abbildung 9: Decoder

Wenn man zwei redundante Bits pro Byte einfügt, wird das Eingangswort der S–Boxen für die 8 Tabellenzugriffe angenehmer. Das zieht allerdings einen Rattenschwanz Änderungen nach sich (Abb.10,11) und auch das KEY–RAM wird nun 128 Byte lang.

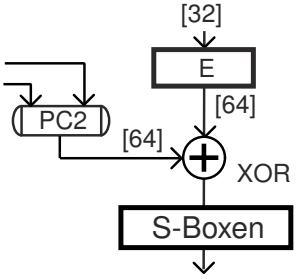


Abbildung 10: Modifikation 6 Bit auf 8 Bit

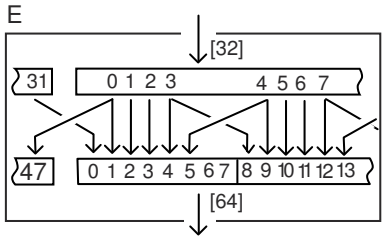


Abbildung 11: Modifikation 6 Bit auf 8 Bit — Box E

Diese Variante in FORTH ist portabel aber langsam (Abb.4). Auf weitere Optimierung der Key Expansion wurde verzichtet.

Wenn man alle Unterprogramme für Encoder und Decoder in Assembler programmiert, wird man deutlich schneller. Den Unterschied machen die Permutationen, belässt man sie in HLL[1], dauert es weiterhin 450msec.

Auch in Assembler wurden die Permutationen hier in einer konventionellen Schleifenstruktur abgearbeitet. Wenn man E– und P–Blöcke in linearen Code umsetzt, steigt der Speicherverbrauch wohl um 1k Byte, aber auch die Geschwindigkeit wird sich nochmal verbessern.

DES wurde im Lauf der Zeit ausgiebig auf Optimierungen untersucht und es scheint wohl möglich, Permutationen komplett zu eliminieren [5]. Allerdings steht der Entwicklungsaufwand vermutlich nicht mit der auf 8–Bit–CPU erreichbaren Geschwindigkeit in sinnvollem Verhältnis.

Tabellen

```
Tabelle 1: Permutationen
IP
39 31 29 21 19 11 09 01
3B 33 2B 23 1B 13 0B 03
3D 35 2D 25 1D 15 0D 05
3F 37 2F 27 1F 17 0F 07
38 30 28 20 18 10 08 00
3A 32 2A 22 1A 12 0A 02
3C 34 2C 24 1C 14 0C 04
3E 36 2E 26 1E 16 0E 06
invIP
27 07 2F 0F 37 17 3F 1F
26 06 2E 0E 36 16 3E 1E
25 05 2D 0D 35 15 3D 1D
24 04 2C 0C 34 14 3C 1C
```

[1] high level language, i.e. Forth

```
23 03 2B 0B 33 13 3B 1B
22 02 2A 0A 32 12 3A 1A
21 01 29 09 31 11 39 19
20 00 28 08 30 10 38 18
E
1F 00 01 02 03 04
03 04 05 06 07 08
07 08 09 0A 0B 0C
0B 0C 0D 0E 0F 10
0F 10 11 12 13 14
13 14 15 16 17 18
17 18 19 1A 1B 1C
1B 1C 1D 1E 1F 00
P
0F 06 13 14 1C 0B 1B 10
00 0E 16 19 04 11 1E 09
01 07 17 0D 1F 1A 02 08
12 0C 1D 05 15 0A 03 18
PC1
38 30 28 20 18 10 08 00
39 31 29 21 19 11 09 01
3A 32 2A 22 1A 12 0A 02
3B 33 2B 23 3E 36 2E 26
1E 16 0E 06 3D 35 2D 25
1D 15 0D 05 3C 34 2C 24
1C 14 0C 04 1B 13 0B 03
PC2
0D 10 0A 17 00 04
02 1B 0E 05 14 09
16 12 0B 03 19 07
0F 06 1A 13 0C 01
28 33 1E 24 2E 36
1D 27 32 2C 20 2F
2B 30 26 37 21 34
2D 29 31 23 1C 1F
Tabelle 2: Shifts
S-TABLE
1 1 2 2 2 2 2 2
1 2 2 2 2 2 2 1
Tabelle 3: S-Boxen
S1
7 2 C F 4 B A C
B 7 6 9 D 4 0 A
2 8 5 3 F 6 9 5
8 1 3 E 1 D E 0
0 F 5 A 7 2 9 5
E 1 3 C B 8 C 6
F 3 6 D 4 9 A 0
2 4 D 7 8 E 1 B
S2
F 0 9 A 6 5 3 9
1 E 4 3 C B A 4
8 7 E 1 D 2 0 C
7 D B 6 2 8 5 F
C B 3 D F C 6 0
2 5 8 E 1 2 D 7
B 1 0 6 4 F 9 A
E 8 5 3 7 4 A 9
S3
5 B 8 D 6 1 D A
9 2 3 4 F C 4 7
0 6 B 8 C F 2 5
7 9 E 3 A 0 1 E
B 8 4 2 C 6 3 D
0 B A 7 6 1 F 4
E 5 1 F 2 9 D A
9 0 7 C 5 E 8 3
S4
E 5 8 F 0 3 D A
7 9 1 C 9 E 2 1
B 6 4 8 6 D 3 4
C 0 A 7 5 B F 2
B C 2 9 6 5 8 3
D 0 4 A 0 B 7 4
1 F E 2 F 8 5 E
A 6 3 D C 1 9 7
S5
4 2 1 F E 5 B 6
2 8 C 3 D E 7 0
3 4 A 9 5 B 0 C
8 D F A 6 1 9 7
7 D A 6 2 8 C 5
4 3 F 0 B 4 1 A
D 1 0 F E 7 9 2
3 E 5 9 8 B 6 C
S6
3 9 0 E 9 4 7 8
5 F C 2 6 3 A D
8 7 B 0 4 1 E B
F A 2 5 1 C D 6
5 2 6 D E 9 0 6
2 4 B 8 9 F C 1
F C 8 7 3 A D 0
4 3 7 E A 5 1 B
S7
2 8 C 5 F 3 A 0
4 D 9 6 1 E 6 9
D 2 3 F 0 C 5 A
7 B E 1 B 7 8 4
B 6 7 9 2 8 4 7
D B A 0 8 5 1 C
0 D C A 9 2 F 4
E 1 3 F 5 E 6 3
S8
B E 5 0 6 9 A F
1 2 C 5 D 7 3 A
4 D 9 6 F 3 0 C
2 8 7 B 8 4 E 1
8 4 3 F 5 2 0 C
B 7 6 9 E 1 9 6
F 8 A 3 C 5 7 A
1 E D 0 2 B 4 D
```

Referenzen

[1] *„Data Encryption Standard“* FIPS Publication 46 US Government of Commerce, National Bureau of Standards 1977 (im www)

[2] *„Validation the Correctness of Hardware Implementation of the NBS Data Encryption Standard“* nbs special publication 500–20 1977 (im www)

[3] *„DES Modes of Operations“* FIPS 81 (im www)

[4] Katzan, *„The Standard Data Encryption Algorithm“,* Petrocelli Books 1977

[5] Fumy, Riess, *„Kryptographie: Entwurf und Analyse symmetrischer Kryptosysteme“* Oldenbourg 1993

Timing & Speicher MC68HC908 @ 2,45 MHz

	HLL	Assembler	
FLASH	2,8	2,6	kBytes
RAM	32+128	33+128	Byte
Key–Expansion	320	320	msec
Encoder	500	50	msec
Decoder	500	50	msec

Listing

```
HEX
TABLE IP-TABLE
3E C, 36 C, 2E C, 26 C, 1E C, 16 C, 0E C, 06 C,
3C C, 34 C, 2C C, 24 C, 1C C, 14 C, 0C C, 04 C,
3A C, 32 C, 2A C, 22 C, 1A C, 12 C, 0A C, 02 C,
38 C, 30 C, 28 C, 20 C, 18 C, 10 C, 08 C, 00 C,
3F C, 37 C, 2F C, 27 C, 1F C, 17 C, 0F C, 07 C,
3D C, 35 C, 2D C, 25 C, 1D C, 15 C, 0D C, 05 C,
3B C, 33 C, 2B C, 23 C, 1B C, 13 C, 0B C, 03 C,
39 C, 31 C, 29 C, 21 C, 19 C, 11 C, 09 C, 01 C,
TABLE invIP-TABLE
1F C, 3F C, 17 C, 37 C, 0F C, 2F C, 07 C, 27 C,
1E C, 3E C, 16 C, 36 C, 0E C, 2E C, 06 C, 26 C,
1D C, 3D C, 15 C, 35 C, 0D C, 2D C, 05 C, 25 C,
1C C, 3C C, 14 C, 34 C, 0C C, 2C C, 04 C, 24 C,
1B C, 3B C, 13 C, 33 C, 0B C, 2B C, 03 C, 23 C,
1A C, 3A C, 12 C, 32 C, 0A C, 2A C, 02 C, 22 C,
19 C, 39 C, 11 C, 31 C, 09 C, 29 C, 01 C, 21 C,
18 C, 38 C, 10 C, 30 C, 08 C, 28 C, 00 C, 20 C,
TABLE E-TABLE
1F C, 00 C, 01 C, 02 C, 03 C, 04 C, 20 C, 20 C,
03 C, 04 C, 05 C, 06 C, 07 C, 08 C, 20 C, 20 C,
07 C, 08 C, 09 C, 0A C, 0B C, 0C C, 20 C, 20 C,
0B C, 0C C, 0D C, 0E C, 0F C, 10 C, 20 C, 20 C,
0F C, 10 C, 11 C, 12 C, 13 C, 14 C, 20 C, 20 C,
13 C, 14 C, 15 C, 16 C, 17 C, 18 C, 20 C, 20 C,
17 C, 18 C, 19 C, 1A C, 1B C, 1C C, 20 C, 20 C,
1B C, 1C C, 1D C, 1E C, 1F C, 00 C, 20 C, 20 C,
TABLE P-TABLE
0F C, 06 C, 13 C, 14 C, 1C C, 0B C, 1B C, 10 C,
00 C, 0E C, 16 C, 19 C, 04 C, 11 C, 1E C, 09 C,
01 C, 07 C, 17 C, 0D C, 1F C, 1A C, 02 C, 08 C,
12 C, 0C C, 1D C, 05 C, 15 C, 0A C, 03 C, 18 C,
TABLE PC1-TABLE
3F C, 37 C, 2F C, 27 C, 1F C, 17 C, 0F C, 07 C,
3E C, 36 C, 2E C, 26 C, 1E C, 16 C, 0E C, 06 C,
3D C, 35 C, 2D C, 25 C, 1D C, 15 C, 0D C, 05 C,
3C C, 34 C, 2C C, 24 C, 39 C, 31 C, 29 C, 21 C,
19 C, 11 C, 09 C, 01 C, 3A C, 32 C, 2A C, 22 C,
1A C, 12 C, 0A C, 02 C, 3B C, 33 C, 2B C, 23 C,
1B C, 13 C, 0B C, 03 C, 1C C, 14 C, 0C C, 04 C,
TABLE PC2-TABLE
0D C, 10 C, 0A C, 17 C, 00 C, 04 C, 0 C, 0 C,
02 C, 1B C, 0E C, 05 C, 14 C, 09 C, 0 C, 0 C,
16 C, 12 C, 0B C, 03 C, 19 C, 07 C, 0 C, 0 C,
0F C, 06 C, 1A C, 13 C, 0C C, 01 C, 0 C, 0 C,
28 C, 33 C, 1E C, 24 C, 2E C, 36 C, 0 C, 0 C,
1D C, 27 C, 32 C, 2C C, 20 C, 2F C, 0 C, 0 C,
2B C, 30 C, 26 C, 37 C, 21 C, 34 C, 0 C, 0 C,
2D C, 29 C, 31 C, 23 C, 1C C, 1F C, 0 C, 0 C,
TABLE S-TABLE
00 C, 00 C, 01 C, 01 C, 01 C, 01 C, 01 C, 01 C,
00 C, 01 C, 01 C, 01 C, 01 C, 01 C, 01 C, 00 C,
TABLE S1-TABLE
07 C, 02 C, 0C C, 0F C, 04 C, 0B C, 0A C, 0C C,
0B C, 07 C, 06 C, 09 C, 0D C, 04 C, 00 C, 0A C,
02 C, 08 C, 05 C, 03 C, 0F C, 06 C, 09 C, 05 C,
08 C, 01 C, 03 C, 0E C, 01 C, 0D C, 0E C, 00 C,
00 C, 0F C, 05 C, 0A C, 07 C, 02 C, 09 C, 05 C,
0E C, 01 C, 03 C, 0C C, 0B C, 08 C, 0C C, 06 C,
0F C, 03 C, 06 C, 0D C, 04 C, 09 C, 0A C, 00 C,
02 C, 04 C, 0D C, 07 C, 08 C, 0E C, 01 C, 0B C,
TABLE S2-TABLE
F0 C, 00 C, 90 C, A0 C, 60 C, 50 C, 30 C, 90 C,
10 C, E0 C, 40 C, 30 C, C0 C, B0 C, A0 C, 40 C,
80 C, 70 C, E0 C, 10 C, D0 C, 20 C, 00 C, C0 C,
70 C, D0 C, B0 C, 60 C, 20 C, 80 C, 50 C, F0 C,
C0 C, B0 C, 30 C, D0 C, F0 C, C0 C, 60 C, 00 C,
20 C, 50 C, 80 C, E0 C, 10 C, 20 C, D0 C, 70 C,
B0 C, 10 C, 00 C, 60 C, 40 C, F0 C, 90 C, A0 C,
E0 C, 80 C, 50 C, 30 C, 70 C, 40 C, A0 C, 90 C,
TABLE S3-TABLE
05 C, 0B C, 08 C, 0D C, 06 C, 01 C, 0D C, 0A C,
09 C, 02 C, 03 C, 04 C, 0F C, 0C C, 04 C, 07 C,
00 C, 06 C, 0B C, 08 C, 0C C, 0F C, 02 C, 05 C,
07 C, 09 C, 0E C, 03 C, 0A C, 00 C, 01 C, 0E C,
0B C, 08 C, 04 C, 02 C, 0C C, 06 C, 03 C, 0D C,
00 C, 0B C, 0A C, 07 C, 06 C, 01 C, 0F C, 04 C,
0E C, 05 C, 01 C, 0F C, 02 C, 09 C, 0D C, 0A C,
09 C, 00 C, 07 C, 0C C, 05 C, 0E C, 08 C, 03 C,
TABLE S4-TABLE
E0 C, 50 C, 80 C, F0 C, 00 C, 30 C, D0 C, A0 C,
70 C, 90 C, 10 C, C0 C, 90 C, E0 C, 20 C, 10 C,
B0 C, 60 C, 40 C, 80 C, 60 C, D0 C, 30 C, 40 C,
C0 C, 00 C, A0 C, 70 C, 50 C, B0 C, F0 C, 20 C,
B0 C, C0 C, 20 C, 90 C, 60 C, 50 C, 80 C, 30 C,
D0 C, 00 C, 40 C, A0 C, 00 C, B0 C, 70 C, 40 C,
10 C, F0 C, E0 C, 20 C, F0 C, 80 C, 50 C, E0 C,
A0 C, 60 C, 30 C, D0 C, C0 C, 10 C, 90 C, 70 C,
TABLE S5-TABLE
04 C, 02 C, 01 C, 0F C, 0E C, 05 C, 0B C, 06 C,
02 C, 08 C, 0C C, 03 C, 0D C, 0E C, 07 C, 00 C,
03 C, 04 C, 0A C, 09 C, 05 C, 0B C, 00 C, 0C C,
08 C, 0D C, 0F C, 0A C, 06 C, 01 C, 09 C, 07 C,
07 C, 0D C, 0A C, 06 C, 02 C, 08 C, 0C C, 05 C,
04 C, 03 C, 0F C, 00 C, 0B C, 04 C, 01 C, 0A C,
0D C, 01 C, 00 C, 0F C, 0E C, 07 C, 09 C, 02 C,
03 C, 0E C, 05 C, 09 C, 08 C, 0B C, 06 C, 0C C,
TABLE S6-TABLE
30 C, 90 C, 00 C, 0E0 C, 90 C, 40 C, 70 C, 80 C,
50 C, F0 C, C0 C, 020 C, 60 C, 30 C, A0 C, D0 C,
80 C, 70 C, B0 C, 000 C, 40 C, 10 C, E0 C, B0 C,
F0 C, A0 C, 20 C, 050 C, 10 C, C0 C, D0 C, 60 C,
50 C, 20 C, 60 C, 0D0 C, E0 C, 90 C, 00 C, 60 C,
20 C, 40 C, B0 C, 080 C, 90 C, F0 C, C0 C, 10 C,
F0 C, C0 C, 80 C, 070 C, 30 C, A0 C, D0 C, 00 C,
40 C, 30 C, 70 C, 0E0 C, A0 C, 50 C, 10 C, B0 C,
TABLE S7-TABLE
02 C, 08 C, 0C C, 05 C, 0F C, 03 C, 0A C, 00 C,
04 C, 0D C, 09 C, 06 C, 01 C, 0E C, 06 C, 09 C,
0D C, 02 C, 03 C, 0F C, 00 C, 0C C, 05 C, 0A C,
07 C, 0B C, 0E C, 01 C, 0B C, 07 C, 08 C, 04 C,
0B C, 06 C, 07 C, 09 C, 02 C, 08 C, 04 C, 07 C,
0D C, 0B C, 0A C, 00 C, 08 C, 05 C, 01 C, 0C C,
00 C, 0D C, 0C C, 0A C, 09 C, 02 C, 0F C, 04 C,
0E C, 01 C, 03 C, 0F C, 05 C, 0E C, 06 C, 03 C,
TABLE S8-TABLE
B0 C, E0 C, 50 C, 00 C, 60 C, 90 C, A0 C, F0 C,
10 C, 20 C, C0 C, 50 C, D0 C, 70 C, 30 C, A0 C,
40 C, D0 C, 90 C, 60 C, F0 C, 30 C, 00 C, C0 C,
20 C, 80 C, 70 C, B0 C, 80 C, 40 C, E0 C, 10 C,
80 C, 40 C, 30 C, F0 C, 50 C, 20 C, 00 C, C0 C,
B0 C, 70 C, 60 C, 90 C, E0 C, 10 C, 90 C, 60 C,
F0 C, 80 C, A0 C, 30 C, C0 C, 50 C, 70 C, A0 C,
10 C, E0 C, D0 C, 00 C, 20 C, B0 C, 40 C, D0 C,

8 ZVARIABLE M    \ Enc: Message in , Chiffre out
                 \ Dec: Chiffre in , Message out
8 ZVARIABLE K    \ Key
8 ZVARIABLE L/R L/R CONSTANT SPK \ scratch keys
8 ZVARIABLE C/D C/D CONSTANT SPR \ scrtch rounds

$HEAP CONSTANT KEY-RAM  \ 128 bytes

\ bitmanipulation in 64 bit words
: (B)     \ ( Addr Bit  --- Addr' Bit' )
  SWAP OVER 1SHIFT> 1SHIFT> 1SHIFT> + SWAP 07 AND ;

: wB@  (B) B@  ; \ ( Addr Bit  --- Flag ) @ bit
: wB1! (B) B1! ; \ ( Addr Bit  ---  )  set bit
: wB0! (B) B0! ; \ ( Addr Bit  ---  ) clear bit

\ ------------------------------------------------
\ 32 and 64 bit XOR   (addr1) <- (addr1) XOR (addr2)

: (XOR)   \ ( Addr1 Addr2  ---  ) 16 Bit
  @ OVER @ XOR SWAP ! ;
: qXOR    \ ( Addr1 Addr2  ---  ) 32 Bit
  2DUP (XOR) 2+ SWAP 2+ SWAP (XOR) ;
: wXOR    \ ( Addr1 Addr2  ---  ) 64 Bit
```

```
  2DUP qXOR 4 + SWAP 4 + SWAP qXOR ;

\ ------------------------------------------------
: IP-PERM          \ (  ---  ) L/R <- M
  L/R 8 00 FILL
  63 0 DO M   IP-TABLE    I + C@ wB@
          IF L/R I wB1! THEN
       LOOP ;

: invIP-PERM       \ (  ---  ) M <- L/R
  M 8 00 FILL
  63 0 DO L/R invIP-TABLE I + C@ wB@
          IF M   I wB1! THEN
       LOOP ;

: P-PERM           \ (  ---  ) SPR <- L/R
  SPR 4 00 FILL
  31 0 DO L/R P-TABLE     I + C@ wB@
          IF SPR I wB1!
       THEN LOOP ;

: E-PERM             \ (  ---  ) SPR <- R
  SPR 8 00 FILL
  63 0 DO L/R 4 + E-TABLE I + C@ wB@
          IF SPR I wB1!
       THEN LOOP ;

: PC1-PERM         \ (  ---  ) C/D <- K
  C/D 8 00 FILL
  55 0 DO K   PC1-TABLE   I + C@ wB@
          IF C/D I wB1! THEN
          LOOP ;

: PC2-PERM         \ (  ---  ) SPK <- C/D
  SPK 8 00 FILL
  63 0 DO C/D PC2-TABLE   I + C@ wB@
          IF SPK I wB1!
       THEN LOOP ;

: S-BOX             \ (  ---  ) L/R <- SPR
SPR     C@ 3F AND S1-TABLE + C@
SPR 1+  C@ 3F AND S2-TABLE + C@ OR L/R     C!
SPR 2+  C@ 3F AND S3-TABLE + C@
SPR 3 + C@ 3F AND S4-TABLE + C@ OR L/R 1+  C!
SPR 4 + C@ 3F AND S5-TABLE + C@
SPR 5 + C@ 3F AND S6-TABLE + C@ OR L/R 2+  C!
SPR 6 + C@ 3F AND S7-TABLE + C@
SPR 7 + C@ 3F AND S8-TABLE + C@ OR L/R 3 + C! ;

DECIMAL

: ROR         \ (  ---  )  Rotate C and D right
  0 7 0 DO
  C/D 7 I - + C@ OR DUP 1SHIFT>
  C/D 7 I - + C! 8<SHIFT
  LOOP
  C/D 27 wB@ IF C/D 55 wB1! THEN C/D 27 wB0!
  H% 100 AND IF C/D 27 wB1! THEN ;

: KEY-EXP      \ ( --- ) in: Key in K ; out: KEY-RAM
  PC1-PERM          \ C/D <- K
  KEY-RAM 15 0 DO
    ROR
    s-TABLE I + C@  \ (  --- s ) s = 1,2
    IF ROR THEN
  PC2-PERM          \ SPK <- C/D
  SPK OVER 8 CMOVE 8 +
  LOOP DROP ;

: ENC    \ ( --- ) Message in M, Key Exp. in KEY-RAM
  IP-PERM                \ L/R <- M
  KEY-RAM 15 0 DO
    L/R M 8 CMOVE        \  M  <- L/R
    E-PERM               \ SPR <- L/R ; only R
    SPR OVER wXOR 8 +    \ SPR <- SPR XOR Key
    S-BOX                \ L/R <- SPR
    P-PERM               \ SPR <- L/R
    M   SPR qXOR         \  M  <-  M  XOR SPR ; only L
  I 15 =
    IF   M     L/R     8 CMOVE  \ L/R <- M
    ELSE M 4 + L/R     4 CMOVE  \ L/R <- M ; swap M
         M     L/R 4 + 4 CMOVE
    THEN
  LOOP DROP invIP-PERM ;  \ M <- L/R

: invENC    \ ( --- ) Cipher in M, Key Exp. in KEY-RAM
  IP-PERM                \ L/R <- M
  KEY-RAM 120 + 15 0 DO
    L/R M 8 CMOVE        \  M  <- L/R
    E-PERM               \ SPR <- L/R ; only R
    SPR OVER wXOR 8 -    \ SPR <- SPR XOR Key
    S-BOX                \ L/R <- SPR
    P-PERM               \ SPR <- L/R
    M   SPR qXOR         \  M  <- M XOR SPR ; only L
  I 15 =
    IF   M     L/R     8 CMOVE  \ L/R <- M
    ELSE M 4 + L/R     4 CMOVE  \ L/R <- M ; swap M
         M     L/R 4 + 4 CMOVE
    THEN
  LOOP DROP invIP-PERM ;        \ M <- L/R

HEX

: M.   7 0 DO M I + C@ CH. LOOP ;

: MK1!      \ (  ---  ) testdata
  4E  M     C! 01  K     C!
  6F  M 1 + C! 23  K 1 + C!
  77  M 2 + C! 45  K 2 + C!
  20  M 3 + C! 67  K 3 + C!
  69  M 4 + C! 89  K 4 + C!
  73  M 5 + C! AB  K 5 + C!
  20  M 6 + C! CD  K 6 + C!
  74  M 7 + C! EF  K 7 + C! ;

: TEST
  MK1! CR M. KEY-EXP  ENC  M.  ." = Chiffre"
  CR D% 24 SPACES ." 3F A4 0E 8A 98 4D 48 15 = soll"
  invENC CR M. ;
```

DES: Die Kontroverse

Rafael Deliano

Kahns voluminöses Standardwerk [1] über die Geschichte der Kryptographie erschien 1967. Ursprünglich eine Domäne des Militärs, rückte das Thema damals verstärkt in die Öffentlichkeit. Durch die rasche Ausbreitung von Computern in allen Regierungseinrichtungen und in kritischen Wirtschaftsbereichen wie Banken ergab sich eine gefühlte Bedrohung. Teil der Gegenmaßnahmen sollte ein Standard für Datenverschlüsselung der US–Regierungsbehörde NBS (National Bureau of Standards) sein. Die Reaktion auf die öffentliche Ausschreibung 1973 für einen Algorithmus war null. Im zweiten Anlauf 1974 kam ein einziger Vorschlag, der von IBM. Außer Ruhm & Ehre war ja nichts zu holen und man musste auch Patentansprüche aufgeben.

LUCIFER

Dr. Horst Feistel war 1934 in die USA ausgewandert und hatte für IBM in den 60er Jahren die LUCIFER–Chiffre entwickelt [2]. Pikantes Detail: IBM setzte damals auf 128–Bit–Schlüssellänge. Die Entwicklungsabteilung Kryptographie bei IBM war Anfang der 70er auf Dr. Walt Tuchman übergegangen der damit für die Entwicklung von DES verantwortlich wurde. Die NBS konnte die Brauchbarkeit des Vorschlags von IBM nicht beurteilen und wandte sich an die zuständige, öffentlichkeitsscheue Behörde NSA [7] (Abb.1). Sie stellte fest, dass die Chiffre technisch unangenehm gut war, deshalb wurden die Unterlagen zur Entwicklung nun als *classified* eingestuft. Konheim, IBM: „We sent the S–boxes off to Washington. They came back and were all different.“ Die offizielle Linie von IBM ist jedoch, dass die NSA nichts verändert hat: „The NSA did not dictate a single wire!“ Die war guter Kunde der IBM Federal Systems Division, das Verhältnis entsprechend herzlich. Das Senate Select Committee [8] würde die schöne Formulierung beisteuern: „... indirectly assisted in the development of the S–box structures. Die Verschwörungstheorien, dass man eine *trapdoor*[1] eingebaut hatte, würden in den nächsten Jahrzehnten kein Ende nehmen. Zweck der trapdoor wäre, dem Kundigen die bequeme Dechiffrierung zu ermöglichen. Die NSA äussert sich prinzipiell nicht dazu. Heute besteht die Vermutung, dass man DES gegen differentielle Kryptoanalyse verbessert hat. Die war den Akademikern damals unbekannt, aber die NSA war absehbar schon weiter. Sie war in der Zwickmühle: Sie wollte wohl keinen Standard, hatte aber die Pflicht ihn stark genug zu machen: „Federal Agencies are required to use DES for the protection of unclassified data [4]“. Aber eben nicht zu stark. Das Senate Committee [8] deutete die Verkürzung der Schlüssellänge so an: „NSA convinced IBM that a reduced key size was sufficient.“ Selbst 64 Bit waren wohl noch zu lang, unbekannt, wie lange sie um die Parity–Bits gefeilscht haben.

Verabschiedung

1975 kündigte die NBS den Standard an und pro forma waren Einsprüche möglich. Von den Akademikern, die DES kritisierten, war der lautstärkste und deshalb querbeet meistzitierte [3] Dr. Martin Hellman, Stanford. Zusammen mit seinem Noch–Studenten Whitfield Diffie verkündete er rastlos: „DES will be totally insecure within 10 years.“ Weil die Schlüssellänge zu kurz ist. Eine *brute force* Schlüsselsuchmaschine, „a parallel computer using 1 million chips“, könne man für 20 Mio Dollar heute[2] schon bauen. Und in 10 Jahren würde die Maschine dann mit den Fortschritten in der Hardware nur noch $200k kosten „and be within the reach of any large organization.“ Ernst Stavro Blofelds SPECTRE hat er nicht beim Namen genannt.

Abbildung 1: The Puzzle Palace

Die Befürworter waren um grosse Zahlen auch nicht verlegen. Dr. Ruth Davies, NBS 1977: „While no code is theoretically unbreakable, 2500 years of computer time on a general–purpose computer significantly faster then a CDC7600 would be required to derive a key ... well over $100 million could be spent five years from now to find a key [6]“. Weitere wüste Abschätzungen beider Seiten sind in [8] angeführt. Zur Verbesserung ihres Images hat die NBS noch Workshops zum Thema trapdoors und Erhöhung der Schlüssellänge abgehalten. Auch die Kritiker waren eingeladen, es wurde lebhaft diskutiert. Dass deren Erhöhung technisch keine nennenswerten Mehrkosten verursachen würde, war klar. Aber eine Lebensdauer von 10 ... 15 Jahren wurde als ausreichend angesehen.

[1] engl. für „Falltür“. Siehe auch engl. *backdoor*, „Hintertür“. Bezeichnet einen (oft vom Autor eingebauten) Teil einer Software, der es Benutzern ermöglicht, unter Umgehung der normalen Zugriffssicherung Zugang zum Computer oder einer sonst geschützten Funktion eines Computerprogramms zu erlangen.

[2] In den 1970er Jahren

Die Einschätzung auf dem Workshop: „A machine which finds, on the average, one key per day could probably not be built until 1990, and the probability factor of it being available even then is estimated to be between 0.1 and 0.2. In addition the cost of such a machine would be several tens of millions of dollars[5c]“. Angeblich fürchteten die Hersteller drakonische Einschränkungen der Exporterlaubnis, wenn der Schlüssel länger würde. Da ihr lukrativster Kunde die US–Regierung war, kamen sie den unausgesprochenen Vorstellungen der NSA ohnehin gern entgegen. Der Standard wurde 1977 unmittelbar nach den Workshops verabschiedet. Ein Senate Select Committee beschäftigte sich dann nochmal mit der Rolle der NSA [8], aber DES war nicht mehr aufzuhalten, wurde dann alle 5 Jahre bestätigt, das letzte Mal 1999, und ist seit 2005 obsolet. Mehrere US–Organisationen, wie ANSI, haben den Standard übernommen und er hat sich auch international de facto durchgesetzt. Es gab 1986 einen Anlauf für einen entsprechenden weltweiten ISO–Standard [4]. Dort hat man dann aber bald darauf kalte Füße bekommen.

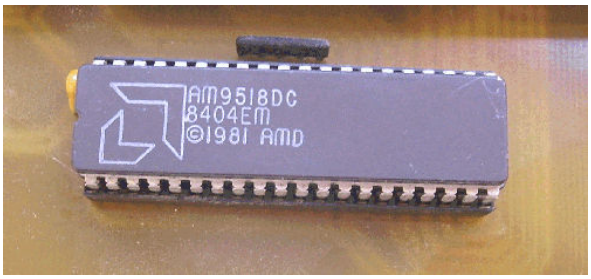


Abbildung 2: DES–IC

Hardware versus Software

Implementierung in Software war nach Definition von NBS nicht „compliant[3]“, nur Hardware und eventuell Firmware konnte von NBS zertifiziert werden. Boards kosteten ehedem $1000 ... $3000, es wurden bis 1988 nur ca. 31 Produkte freigegeben [4]. Goldgrube für eine Handvoll, die Hoflieferanten der Regierung. Für Banken ist ANSI relevant und die erlaubte explizit Implementierung in Software. Die Halbleiter–Hersteller lieferten bald ICs. Intels 8294, ein maskenprogrammierter 8042 Controller, wurde am schnellsten verfügbar. Motorola MC6859, eine echte Logikschaltung, sollte am MC6800–Bus betrieben werden, wurde aber wohl nie handelsüblich. Der AMD AM9518 (Abb.2) von 1982 mit seiner pro forma second–source Zilog AmZ8068 kostete damals stolze $55 und hatte einige Verbreitung. Nach diesen ersten NMOS–ICs zeigte sich, dass kein Markt für derartige Halbleiter existierte. Das Pentagon als klassischer Großkunde hatte andere Verfahren und würde DES nicht mit spitzen Fingern anfassen.

Export

War aus den USA ehedem nur beschränkt möglich, Einzelfallprüfung nötig. Wenn amerikanische DES–ICs nicht exportierbar waren, wäre die Annahme gewesen, dass japanische und europäische Halbleiterhersteller die Lücke füllen würden. Letztlich verhinderte das mehr die unzureichende Nachfrage als Regierungsinterventionen.

[3] engl.: verträglich, erwünscht

DES–Cracker

Eine Schwachstelle zu finden, hatte sich als unergiebig erwiesen und so konzentrierten sich die Kritiker auf *brute force* Abklappern der Schlüssel. Die Entwicklung immer schnellerer Hardware kam dem entgegen. Ab Ende der 90er Jahre wurden publicityträchtige Suchmaschinen vermeldet. Eine wirkliche Gefahr für Anwender sind sie nicht.

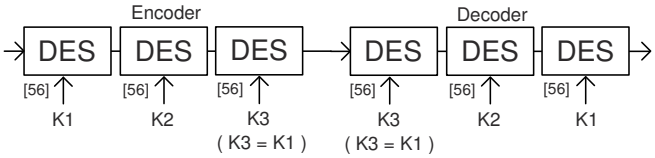


Abbildung 3: Triple–DES

Triple DES

Es war von Anfang an bekannt, dass man die effektive Wortlänge durch Kaskadierung erhöhen kann. Tuchman hatte bereits 1978 darauf hingewiesen [5c], aber Hellman hatte das immer als unzureichend abgelehnt. Ende der 90er Jahre wurde das Problem aber akut und 1998/1999 wurden schließlich formal Standards verabschiedet. Man muss DES dreimal ausführen, um effektiv auf einen ca. 112 Bit langen Schlüssel zu kommen (Abb.3). Praktisch wurde deshalb kein 168–Bit–Schlüssel verwendet, sondern ein Teilschlüssel doppelt eingesetzt. Man hatte die DES–Cracker damit abgeschüttelt, die Umständlichkeit der Lösung war aber offensichtlich. Mittlerweile waren gute Algorithmen, die viel besser in Software implementierbar waren, entstanden. DES erschien immer mehr als Anachronismus. Das letzte Wort zum Oldtimer soll James Massey haben: „The general consensus of cryptologic researchers today is that DES is an extremly good cipher with an unfortunately small key“ [9].

Referenzen

[1] Kahn, *„The Codebreakers“,* Macmillan 1967

[2] Feistel, *„Cryptograpy and computer privacy“,* Scientific American, May 1973 (im www)

[3] DrDobbs, *„How secure is computer data? Not very, say Stanford experts“,* September 1976

[4] Smid, Branstad, *„The Data Encryption Standard: Past and Future“,* Proceedings IEEE May 1988, NBS

[5] IEEE Spectrum, Schwerpunktthema July 1979

[5a] *„DES will be totally insecure within 10 years“,* Hellman, Stanford

[5b] *„Hellman's scheme breaks DES in its basic form“,* Davida, National Science Foundation

[5c] *„Hellman presents no shortcut solutions to the DES“,* Tuchman, IBM

[5d] *„Hellman's data does not support his conclusion“,* Branstead, NBS

[6] Katzan, *„The Standard Data Encryption Algorithm“,* Petrocelli Books 1977

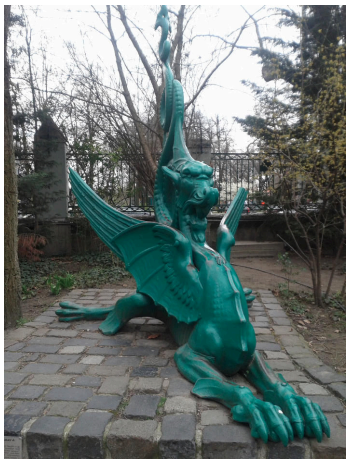
[7] Bamford, *„The Puzzle Palace“,* Houghton Mifflin 1982

[8] *„Unclassified summary: Involvment of NSA in the development of the Data Encryption Standard“,* Senate Select Committee on Intelligence, April 1978, (im www)

[9] Massey, *„Contemporary Cryptography: An Introduction“,* in: Simmons *„Contemporary Cryptology“,* IEEE Press 1992

SWAP und seine Freunde

SWAP hat Freunde in aller Welt — hier wiederum ein Geschwisterpaar — dieses bewacht wilde Tiere:



Frage: **Wo befinden sich diese beiden Swap–Freunde?**

Antworten bitte an die Redaktion unter `vd@forth-ev.de`.

Die Antwortgeber der richtigen Antworten werden im kommenden Forth–Magazin veröffentlicht. uho

mcForth – Ein Forth für viele Microcontroller

Klaus Kohl–Schöpe

Herzlich willkommen bei meinem ersten Artikel über mcForth. Dieses für viele Microcontroller geeignete Forth wurde erstmals auf der Jahrestagung der Forth–Gesellschaft e.V. im April 2016 in Augsburg vorgestellt. Weitere Informationen und erste Implementierungen sind auf www.mcforth.net verfügbar.

Vorgeschichte

Meinen ersten Forth–Kontakt hatte ich vor 30 Jahren mit einer Version für den Sharp–PC–1500–BASIC–Taschenrechner, die für ein Messsystem realisiert wurde. Wenige Jahre später wurde der mit Microchip PIC16 in Assembler realisierte Abtastsequenzer auf größeren Systemen durch ein Novix NC4000 bzw. dann Harris RTX–2000 ersetzt. Während dieser Zeit wurden auch die ersten Kontakte mit der Forth–Gesellschaft geknüpft und einige Projekte realisiert. Logische Folge war natürlich ein eigenes Forth, angelehnt an das damals beliebte Forth–83 bzw. volksForth. Da ich schon immer diverse Prozessoren nutzte, wurde das KKForth neben 8086 (PC, NEC–V20) auch an Z80, RTX2000 und 68332 angepasst. Aber immer nur als 16–Bit–Version, meist für EPROM oder RAM.

Als Specialized Field Application Engineer (SFAE) für Microcontroller bei einem der weltweit größten Bauteile–Distributoren habe ich mit vielen Architekturen — aktuell meist mit 32–Bit und großem Flash — zu tun. Schon seit Jahren dachte ich deshalb über ein Upgrade des KKForth nach. Weil auf dem beruflich genutzten Windows 7 kein DOS mehr direkt verfügbar ist, wurde die Verwendung des KKForth schwieriger und der Wunsch nach einem neuen Forth immer größer.

Das Ziel dieses neuen *mcForth* war, dass es auf möglichst vielen Systemen läuft und wegen der besseren Erlernbarkeit und einfacheren Portierbarkeit von Programmen (auch aus KKForth) möglichst dem aktuellen Standard entspricht. Es muss auf PC (unter Windows 7 oder Linux) und diversen ARM–Cortex–M's als 32–Bit–Version im Flash laufen. Aber auch andere Architekturen bzw. 16–Bit–Versionen sollten möglich sein.

Im Einzelnen waren folgende Features gewünscht:

- Nicht nur PC (DOS) als Entwicklungsumgebung
- Unterstützt 16– bis mindestens 32–Bit–Systeme
- An viele Architekturen anpassbar (x86, M0, x51 . . .)
- Sollte interaktiv ins Flash compilieren können
- Auch auf Harvard–Architektur realisierbar
- Wenig Abweichungen von aktuellen Standards
- Sollte leicht portierbar sein

Bis zum aktuellen Stand dauerte es noch viele Jahre mit Realisierung von diversen Targetcompilern und Simulatoren. Falls sich jemand für die Frage nach Henne und Ei bei mcForth interessiert: Es war tatsächlich ein Targetcompiler und ein 16–Bit–Simulator unter KKForth die Ausgangsbasis. Danach folgten die 32–Bit–Version des Targetcompilers (später auch unter mcForth) und die in C und zuletzt mit MASM32 realisierten 32–Bit Simulatoren.

Herausforderungen

Aufbau eines Forth–Systems

Ein Forth–System besteht aus einem Prozessor (Execution Unit, diverse Register, ALU) und Speicher (RAM und evtl. Flash). Beide Stacks sind entweder im RAM des Hauptspeichers oder in getrennten Speichern realisiert. Die entsprechenden Zeiger (RP, SP und manchmal die obersten Stackeinträge TOS=Top of Stack, NOS=Next of Stack und TOR=Top of Returnstack) sind, wie auch der Programmzeiger PC, im Prozessor. Programme können im Flash oder im RAM sein. Variablen und diverse Arbeitsbereiche sind immer im RAM, wobei ein Forth–System immer unterschiedliche Adressen für Flash und RAM erwartet. Optimal ist es dann noch, wenn viele Forth–Befehle wie Unterprogramm–Aufrufe, EXIT, DUP, + . . . in einem Takt ausgeführt werden können.

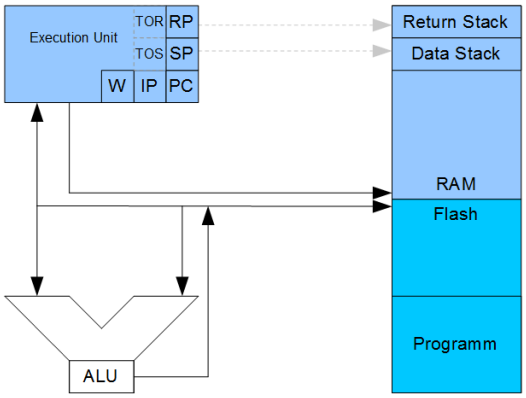


Abbildung 1: Aufbau eines Forth–Systems

Leider weicht ein realer Prozessor deutlich von dieser Struktur ab, was zu folgenden Problemen führt:

- Zu wenig Register für alle Pointer
- Bitbreite abweichend (z.B. nur 8–Bit bei AVR)
- Überlappende Flash/RAM–Adressen (z.B. 8051)
- Getrennte Befehle für Zugriffe Flash/RAM
- Alignment für Speicherzugriffe notwendig
- Sprünge erreichen nicht den ganzen Adressbereich
- Viele Assembler–Befehle für ein Forth–Wort nötig

Realisierung des Forth

Es gibt viele Möglichkeiten, wie ein Forth implementiert werden kann. Am häufigsten genutzt wird ITC– (=Indirect Threaded Code) und die STC–Version (=Subroutine Threaded Code), gezeigt am folgenden Beispiel:

```
Variable counter
: counter++ counter @ 1+ counter ! ;
```

Bei ITC (siehe Abb.2) zeigt das Codefeld (CFA) auf die eigentlich auszuführende Assemblerroutine. In einer :–Definition sind nur Zeiger auf die CFA des Wortes (neben Inline–Daten wie Literals) abgelegt. Während der Ausführung eines solchen Forth ist neben dem Assembler–Programmzeiger (PC) auch noch ein Forth–Programmzeiger notwendig. Dafür wurde das IP–Register und ein Arbeitsregister W (enthält meist die Adresse des Parameterfelds PFA des auszuführenden Wortes) definiert. Vorteil von ITC vor allem bei 16–Bit–Systemen ist, dass jeder Befehl nur zwei Adressen belegt und das Alignment für Speicherzugriffe einfacher einzuhalten ist.

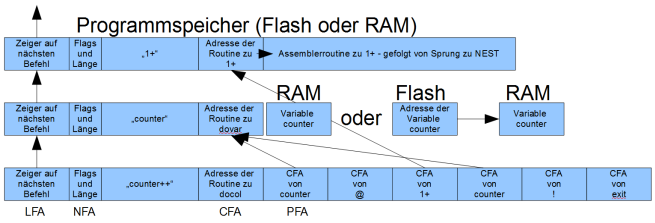


Abbildung 2: ITC — Indirect Threaded Code

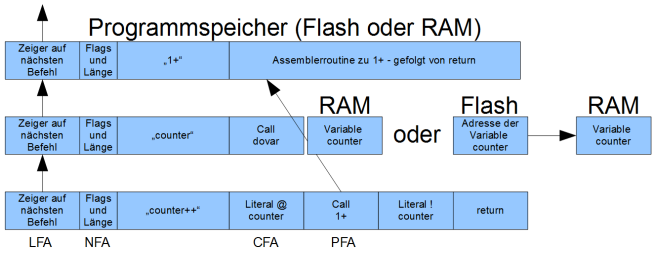


Abbildung 3: STC — Subroutine Threaded Code

Bei einem meist schnelleren STC (siehe Abb.3) gibt es keinen Zeiger im CFA, sondern dort beginnt der Assembler–Code des Wortes oder ein Unterprogrammaufruf der entsprechenden Routine. Nichts hindert einen Entwickler daran, ganze Programmsequenzen direkt als Assembler–Code in das neue Wort zu übernehmen. Ein gutes Beispiel dazu ist das MECRISP für Cortex–M von Matthias Koch.

Da bei einem portablen Forth beide Versionen realisierbar sein sollten und die Strukturen sich deutlich unterscheiden, sind unterschiedliche Targetcompiler, aber auch angepasste Sourcen für das entsprechende Forth nötig.

Speicher

Wie schon erwähnt, haben fast alle Microcontroller einen Flash–Programmspeicher und ein meist deutlich kleineres RAM für Daten. Die Einschränkung des Flash ist, dass es, im Gegensatz zu RAM und FRAM (deshalb ist der MSP430FR so beliebt), nur einmal — meist in größeren Blöcken — geschrieben werden kann. Vorher muss man einen noch deutlich größeren Block löschen. So sind z.B. bei einem Infineon XMC1100 die 256 Byte–Blöcke bis zu 50000 mal löschbar und müssen wegen ECC in Gruppen von 16 Byte geschrieben werden. Viele Forth–Versionen schreiben deshalb neue Wörter ins RAM und speichern am Ende das gesamte RAM–Image ins Flash. Beim Neustart wird dann dieser Teil wieder ins RAM kopiert und ausgeführt. Bei einem typischen Verhältnis von 4:1 (Flash:RAM) kann man sich vorstellen, dass Flash dabei kaum genutzt und das für Variablen und Daten notwendige RAM sehr begrenzt ist.

Aufpassen muss man bei Controllern mit Harvard–Architektur, wie dem 8051, dass dort Programme im RAM überhaupt nicht ausführbar sind und der Trick von oben nicht funktioniert. Es existieren auch unterschiedliche Befehle für Zugriff auf Programm- oder Variablenspeicher, was man z.B. berücksichtigen muss, wenn bei der Verkettung der Befehle das Link–Feld LFA im Flash oder im RAM liegen kann.

Schnittstellen

Bei DOS, Windows oder Linux hat man nicht das Problem eines Filesystems. Entsprechende Aufrufe sind meist aus den Forth–Systemen möglich. Bei kleinen Microcontroller hat man aber oft nur die serielle (Debug–)Schnittstelle während der Entwicklung. Man muss sich eine andere Lösung zum Handling von Files überlegen. Das Einfachste ist für Download ein Copy/Paste im Terminalprogramm, wobei man die Wartezeit nach einer Zeile nicht vergessen darf. Für das Speichern reicht auch ein Hex–Dump, welches für das entsprechende Programmiertool mitgelogged wird. Angenehmer ist es aber, das Handling von Files dem Forth auf dem Microcontroller zugängig zu machen.

mcForth Lösungsansätze

Der virtuelle Prozessor VP32

Um ein Forth leichter anpassen zu können, ist ein Prozessor mit einfacher Struktur und wenigen Befehlen optimal. Forth–Prozessoren haben meist kein Debug–Interface und die aktuellen Prozessoren sind zu komplex. Deshalb die Entscheidung, mit VP32 einen Core mit wenigen Befehlen zu definieren und einen anpassbaren Simulator zu schreiben. Damit konnten die Unterschiede von Neumann– oder Harvard–Architektur und evtl. nötiges Alignment getestet werden. Außerdem erledigt sich damit die Suche nach einem Entwicklungssystem, weil man nur die C– oder Assembler–Sourcen anpassen muss um das mcForth–Image unverändert fast überall laufen zu lassen. Wie man in Abb.4 sieht, hat der 64 Einträge umfassende Befehlssatz des VP32 viele bekannte Forth–Wörter.

Code	FORTH-Befehl	Code	FORTH-Befehl	Code	FORTH-Befehl	Code	FORTH-Befehl
$00	CALL rel	$10	R>	$20	c@	$30	@IO
$01	BRANCH rel	$11	R@	$21	c!	$31	!IO
$02	0BRANCH rel	$12	RDROP	$22	W@	$32	C@X
$03	NBRANCH rel	$13	AND	$23	W!	$33	C!X
$04	LIT value	$14	OR	$24	@	$34	W@X
$05	EXIT	$15	XOR	$25	!	$35	W!X
$06	EXECUTE	$16	+C	$26	C@P	$36	@X
$07	SP@	$17	-C	$27	C!P	$37	!X
$08	SP!	$18	LSHIFT	$28	W@P	$38	FIP@
$09	DROP	$19	RSHIFT	$29	W!P	$39	FIP!
$0A	DUP	$1A	ASHIFT	$2A	@P	$3A	FW@
$0B	OVER	$1B	UM*	$2B	!P	$3B	FW!
$0C	SWAP	$1C	UM/MOD	$2C	C@IO	$3C	NEST
$0D	RP@	$1D	0<	$2D	C!IO	$3D	>CODE
$0E	RP!	$1E	U<	$2E	W@IO	$3E	SYSCALL
$0F	>R	$1F	<	$2F	W!IO	$3F	NOP

Gruppe
Programmlauf
Literal
Datenstack
Returnstack
Arithmetik/Shift/Compare
RAM (Variablen)
Flash (Programm)
IO (Port)
Extended Memory
Indirect Threaded Register
Realer Prozessor

Abbildung 4: Befehlssatz des virtuellen Forth–Prozessors VP32

Da meist üblich, hat jede Adresse und damit jedes Byte tatsächlich 8 Bit. Ein Befehl belegt ein Byte, wobei in einer 32–Bit–Version an Sprünge und Calls ein 32–Bit–Offset und bei Literals der Wert — ebenfalls 4 Bytes — angehängt wird. Bei einem 16–Bit–System hat der Anhang natürlich nur 2 Byte.

Es gibt Arithmetik– und Shift–Befehle, die für eine mehrfachgenaue Berechnung ein Carry–Flag benötigen. Da ich kein Flagregister einführen wollte, missbrauchte ich das nur für ITC benötigte W–Register für den Übertrag aus +c, –c und den drei Shift–Befehlen (nur 0 oder 1). Mit FW@ kann ich dieses Register bei Bedarf auslesen.

Um auch für Harvard gerüstet zu sein, gibt es getrennte Befehle für den Zugriff auf Variablen (RAM – kein Anhang), Programm (...p), Ports (...io) und externer Speicher (...x — doppeltgenaue Adressen). `W@`... und `W!`... sind für den Zugriff auf 16–Bit–Werte definiert, weil oft .w für gleiche Zwecke in Assemblern verwendet wird.

Neben den Befehlen zum Abfragen und Setzen des Forth–IP– und Forth–W–Registers wurde aus Geschwindigkeitsgründen auch der NEST–Befehl realisiert, was auch für Debugging hilfreich ist.

Der Rest ist dann noch die Möglichkeit, mit `>Code` den nachfolgenden Code durch den realen Prozessor auszuführen, mit `SYSCALL` eine Betriebssystemroutine aufzurufen oder einfach bei `NOP` nichts zu tun. Die restlichen Codes $40 ... $FF führen zum Abbruch des Simulators.

Angelehnt an ARM Cortex–M habe ich beim VP32 festgelegt, das der Flash–Bereich bei $00000000 beginnt und 2MByte groß ist. RAM wurde mit 1MByte ab $20000000 auch großzügig dimensioniert. Alle diese Vorgaben und auch der Test von Harvard und Alignment können im Simulator geändert werden.

Entwicklungsumgebung

Ziel war natürlich, das mcForth mit mcForth zu entwickeln und zu testen. Nach der Henne–Ei–Phase (siehe oben) ist jetzt die einzige Anbindung an den realen Prozessor der Simulator, aktuell realisiert in dem frei verfügbaren MASM32 unter Windows. Wegen Portierbarkeit habe ich mich entschlossen, dessen Sourcen auf *www.mcforth.net* zu veröffentlichen. Die C–Version wird noch folgen und erlaubt noch einfachere Portierung selbst auf Microcontroller wie z.B. dem STM32F429 (interessant wegen TFT auf einem 30€–Entwicklungskit).

Alle anderen Tools wie Targetcompiler, (Screen–)Editor oder Assembler und Beispielprogramme wie Life oder das Terminalprogramm sind in mcForth geschrieben und damit fast unverändert auf allen Versionen lauffähig. Zum Test der ANS–Forth–Befehle gab es ANS_Tester_V1.1.f, das ich wegen der unten beschriebenen Abweichungen an einigen Stellen anpassen musste und deshalb auf mcFCoreTest.f umbenannt habe. Generell sind diese Programme eine hervorragender Testumgebung für das mcForth und halfen mir, viele Fehler zu finden.

Speicherbelegung des mcForth

Abb.5 zeigt die Speicherbelegung des mcForth. Ab Adresse 0 ganz unten beginnt das Programm–Image im Flash. Am Anfang gibt es einen kleinen Header, der neben dem Sprung zum Programm weitere Informationen zu dieser Version beinhaltet. In der nächsten Version werden dort auch Konstanten abgelegt, um auch ohne Targetcompiler im mcForth–Image die Flash– und RAM–Größe von 2M/1M z.B. auf 64K/16K zu reduzieren. Danach folgen das Programm und eine Kopie des Variablen– und Heap–Bereiches, die beim Start des mcForth wieder ins RAM kopiert werden. Der Rest ist gelöschtes Flash, in das neue Wörter compiliert werden. Da es mehrere Variablen/Heap–Images geben kann, sucht mcForth beim Start vom Flash–Ende bis zum Anfang nach entsprechenden Signaturen und verwendet dann das letzte mit `SAVE` gespeicherte Image.

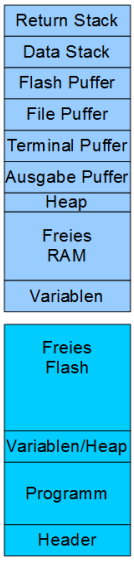


Abbildung 5: Speicheraufbau

Am Anfang des RAMs sind die Variablen oder sogar Programme zu finden. Am Ende des freien Bereiches folgt der Heap. Aus Platz– und Verschlüsselungs–Gründen nutzte ich immer schon den Heap zum Ablegen von Befehlsnamen, die später nicht mehr sichtbar sein sollen. Dies ist in allen mcForth–Versionen möglich und wird mit | für das nächste Wort bzw. +| dauerhaft eingeschaltet. Mit –| landet der Header wieder beim Programm. Mit `CLEARH` wird der Heap gelöscht und |– Befehle sind nicht mehr sichtbar.

Der Rest wird durch Puffer für Ein– und Ausgabe, Files und Stacks belegt. Eine Besonderheit ist hier der Flash–Puffer, der als Zwischenspeicher bei Compilierung von Programmen ins Flash genutzt wird. Mit üblicherweise 1KByte ist dieser groß genug für die meisten Forth–Wörter. Er kann aber auch bei Anwendungen als Zwischenpuffer genutzt werden.

Vokabulare und Befehlsheader in mcForth

Beim Design von mcForth wurde festgelegt, dass man Flash und RAM mit unterschiedlichen Befehlen ansprechen muss. Bei diesem interaktiven System sollte man auch Befehle wieder löschen können, was natürlich im Flash deutlich kritischer ist. Ergebnis war die Notwendigkeit, beim Zugriff auf LFA bei RAM das @ und bei Flash @P zu nutzen. Abb.6 zeigt die zugegeben nicht einfach zu durchschauende Struktur der Verkettung der mcForth–Vokabulare. Wichtig ist nur zu wissen, dass die Verkettung im RAM bei `VOC-LINK` beginnt und dann für jedes Vokabular je ein Zeiger auf das letzte LFA im Flash und im RAM existiert. Da die LFAs im Flash nicht änderbar sind, können dort nur die letzten Befehle gelöscht werden. Im RAM kann LFA verändert werden, was ein beliebiges Ändern der Verkettung im RAM und Heap erlaubt. Bei der Suche wird normalerweise zuerst das RAM und dann das Flash durchsucht. Man kann aber auch mit `||` (auch innerhalb einer Definition) für das nächste Wort, und mit `+||` dauerhaft, einstellen, dass zuerst das Flash durchsucht wird. Mit `-||` wird wieder das RAM zuerst durchsucht.

Solange man eine Neumann–Architektur verwendet und Sprünge von Flash ins RAM möglich sind, darf man die Compilierung auch mit `>VAR` auf das RAM bzw. mit `>PRG` wieder zurück auf Flash stellen. Mit `>VAR?` erfährt man, ob gerade RAM genutzt wird. Weitere Befehle in diesem Zusammenhang sind `UNIFIED?` (`TRUE`, wenn `>VAR` möglich ist) und `PRG?` (`TRUE`, wenn Flash vorhanden ist).

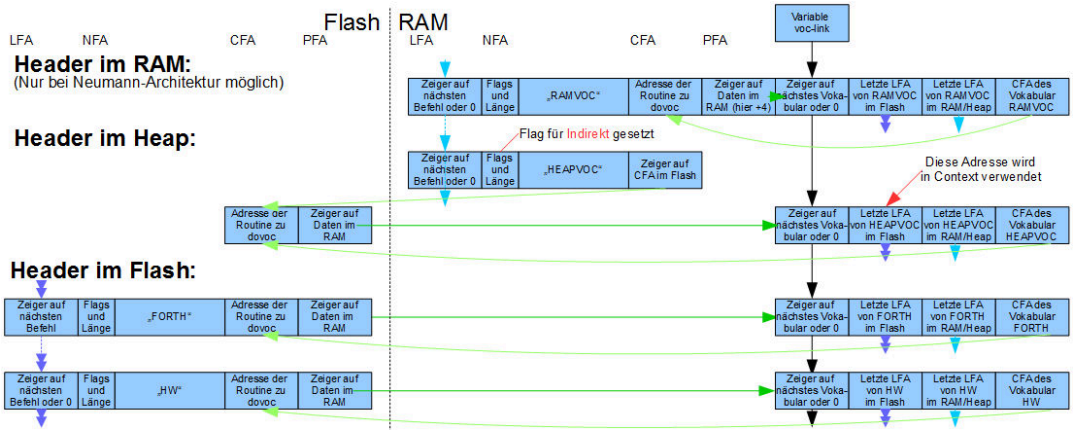


Abbildung 6: Vokabulare und Befehlsheader

Flash und mcForth

Die Herausforderung bei Forth war immer schon die Nutzung von Flash. Bei mcForth habe ich dafür zusätzliche Befehle wie `@p` , `c@p` oder `,p` definiert und setzte eine bewusste Handhabung dieser Tatsache bei der Programmierung voraus. Dies ist aber sehr einfach, weil die Variablen, Konstanten, :–Definitionen und auch `CREATE ... DOES>` wie gewohnt genutzt werden können. Nur wenn man Datenfelder im Flash anlegen will, kommen die neuen Befehle zum Tragen:

```
: 3D: ( Name; C: x y z -- ; -- x y z )
Createp rot ,p swap ,p ,p
Doesp> 3 FOR dup @p swap cell+ NEXT drop ;
0 0 0 3D: 0Vektor
1 2 3 3D: 123Vektor
```

0Vektor bringt 3 mal die 0 und 123Vektor, wie zu erwarten, 1, 2 und 3 auf den Datenstack. Ohne den Anhang `p` würden die Daten ins RAM geschrieben werden und bei `@` statt `@p` der Simulator wegen falschem Adressbereichs abbrechen. Übrigens muss ich das Programm nicht ändern, wenn mit `>VAR` das Programm ins RAM compiliert wird. Denn wenn es erlaubt ist (siehe `UNIFIED?`), sind alle `...p` Befehle mit den entsprechenden RAM–Zugriffen identisch.

Falls sich hier jemand über `3 FOR` wundert: Ich nutze `FOR ... n ... NEXT` für viele Zählschleifen, verwende aber im Gegensatz zu cmForth[1] die Schleifenanzahl. Mit `?FOR` mache ich nichts bei `0`.

cmForth verwendete, soweit ich das weiß, zum ersten Mal den 1–Takt–Rücksprung (NBRANCH genannt). Ist 0 auf dem Returnstack, wird diese entfernt und im Programm weitergemacht. Ansonsten der Wert um 1 erniedrigt und zurückgesprungen. Wie üblich hat Charles Moore extrem optimiert und für FOR nur >R verwendet, was dazu führt das bei 1 FOR ... NEXT die Schleife zweimal durchlaufen wird. Bei meinem mcForth habe ich dies korrigiert und verwende 1– >R für FOR und bei ?FOR wird nur >R und dann direkt der Sprung zu NEXT compiliert. Damit kann ich die richtige Schleifenanzahl für FOR angeben und bei 0 ?FOR ... NEXT wird die Schleife übersprungen.

Damit ich auch die Option habe, ein Register für den Schleifenzähler zu verwenden, habe ich N (ist aktueller Schleifenanzahl−1 ... 0) und NN (zweiter Schleifenwert) eingeführt. Ältere Schleifenzähler landen auf dem Returnstack, weshalb man wie bei DO ... LOOP den Returnstack möglichst unverändert lassen soll.

Wenn mcForth ein Wort ins Flash compiliert, nutzt es den oben schon erwähnten Zwischenpuffer im RAM. Damit es weiß, wann der Puffer initialisiert und wieder zurückgeschrieben werden soll, gibt es die kaskadierbaren Befehle `BEGIN-CREATE` (wird von `HEADER`, `CREATE` oder `:` genutzt) und `END-CREATE` (von `DOES>` und `;` genutzt). Da `CONSTANT` und `VARIABLE` auch beide Wörter nutzen, sieht man bei den meisten Programmen nichts davon. Trotzdem muss man daran denken, dass wenn `CREATE` ohne `DOES>`–Teil verwendet wird, man selbst das `END-CREATE` anhängen muss.

Jeder Befehl in Forth belegt eigentlich mindestens einen schreibbaren Flash–Block, der, wie schon erwähnt, auch sehr groß sein kann. Deshalb die Möglichkeit, am Anfang und Ende eines Programmteils (nicht größer als der Flash–Puffer) ein `BEGIN-CREATE` und `END-CREATE` zu nutzen und damit zu erzwingen, dass die leeren Bereiche zwischen den Wörtern klein bleiben.

[1] Gemeint ist das cmForth für den RTX2000 von Charles Moore

[2] Nur innerhalb einer Colon–Definition erlaubt.

Hier ein Beispiel für den Infineon XMC1100, bei dem man immer 16 Byte schreiben muss:

```
\ Diese Variable benötigt LFA, Länge+Name, CFA
\ und die PFA mit dem Zeiger ins RAM = 14 Bytes
Variable X

\ Diese Variable verbraucht ohne Align 17 Bytes
Variable XXXX
```

Da ich immer $n * 16$ Byte schreiben muss, würde für die `Variable X` ein ganzer Block, also 16 Bytes, verbraucht und für die `Variable XXXX` gleich zwei Blöcke, zusammen somit schon 3 Blöcke. Und es gibt MCU's mit noch größeren Schreibblöcken (64 Bytes bei Atmel SAMD), dann wird das schnell ärgerlich.

Deshalb habe ich BEGIN–CREATE und END–CREATE kaskadierbar gemacht.

```
BEGIN-CREATE
Variable X
Variable XXXX
END-CREATE
```

Dadurch können Definitionen kompakter erzeugt werden. Das Beispiel verbraucht dann nur $14+17 = 31$ Bytes, somit nur noch 2 Blöcke. Der Nachteil davon ist, dass `X` und `XXXX` erst nach dem letzten `END-CREATE` im Vokabular erscheinen.

Aus Geschwindigkeitsgründen wollte ich nicht, dass die Speicherzugriffe wie `@p` oder `c@p` ständig auf den Puffer für das Schreiben auf das Flash achten müssen. Deshalb nutze ich innerhalb aller ...,p – Befehle das `(@p` und `(!p` . Diese überprüfen, ob man innerhalb einer `BEGIN-CREATE ... END-CREATE` (also auch innerhalb `: ... ;`) ist und verwenden dann das RAM des Flash–Puffers. Übrigens gibt es gar kein `c!p` oder `!p` , weil das direkte Schreiben auf Flash gar nicht geht.

Mit `MARKER <NAME>` merke ich mir die aktuellen Positionen im Flash, RAM und Heap und lösche bei Aufruf von `NAME` wieder alle neueren Programmteile. Da ich aber nur ganze Blöcke (bei XMC1100 sind es 256 Byte) löschen kann, muss ich den Marker und die dann folgenden Befehle in den nächsten 256 Byte–Block compilieren. Es gibt aber auch Controller, deren löschbare Blöcke deutlich größer sind (Kinetis–K: 4K). Insofern ist die Verwendung von `MARKER` je nach Größe des Löschblocks in einer MCU kritisch.

Größter Unterschied zu Standard–Forth ist, dass man die Flags für Immediate und Restrict[2] **vor** dem Befehl setzen muss. Dies kann entweder mit den Befehlen `(i` , `(r` und `(ir` oder eleganter bei :–Definitionen mit i: , `r:` und `ir:` erfolgen.

Schnittstellen in mcForth

Generell werden beim VP32 die SYSCALLs verwendet, um Funktionen des Betriebssystems über Simulator aufzurufen. Dazu werden die Parameter zusammen mit der Funktionsnummer auf den Stack gelegt. Eine Liste der Befehle ist in der mcForth–Präsentation und im ebenfalls auf dem Web verfügbaren Handbuch zu finden.

Auf Microcontrollern wird — wie schon im KKForth — eine ESC–Sequenz genutzt, um diese Befehle an das (natürlich in VP32 realisierte) Terminalprogramm und von dort an den PC zu übermitteln. Damit stehen alle Betriebssystem–Befehle des VP32–Simulators anderen mcForth–Versionen z.B. auf dem XMC2Go zur Verfügung. Ich habe mich – vor allem wegen Verwendung von „dummen“ Terminals – dazu entschlossen, diese ESC–Sequenzen erst dann zu nutzen, wenn man dies mit dem Befehl +MCFT (mcForth–Terminal) erlaubt.

Ausblick

Was ist verfügbar?

Noch vor dem langen Wochenende mit Jahrestagung der Forth–Gesellschaft e.V. habe ich die schon über 2 Jahre existierende Webseite www.mcforth.net aktualisiert. Damit sind Präsentation, Handbuch, die VP32–Version mit Simulator (einschließlich Sourcen), die XMC2Go–Version und einige Beispielprogramme für alle verfügbar.

Woran wird noch gearbeitet?

Es gibt noch eine größere Wunschliste, die ich realisiert will. Einige dieser Punkte waren schon in KKForth verfügbar, müssen aber noch angepasst werden:

- Speicherbereiche über Konstanten anpassbar
- C–Version des VP32–Simulators
- Weitere Dokumentationen und Einführungen
- 16–Bit–Versionen (VP16, x51, MSP430, RL78)
- Einfacher sequenzieller Editor (aktuell Screens)
- Multitasker und Interrupts
- Floatingpoint und Grafik

Schlusswort

Ich hoffe, ich habe mit diesem Artikel das Interesse an einem Forth geweckt, das einiges anders macht und trotzdem fast wie gewohnt zu nutzen ist. Mir ist klar, dass es noch einige Fehler gibt und auch Verbesserungen in Konzept und Realisierung nötig sind. Deshalb freue ich mich über alle Rückmeldungen zum mcForth über meine E–Mail–Adresse *kks@mcforth.net*.

Über die Freigabe der Sourcen für das mcForth oder sogar des Targetcompilers verhandele ich noch (mit mir). Bei KKForth war dies nicht nötig und ersparte mir auch viel Support. Aber wenn das neue mcForth auf allen mir verfügbaren 300+ Boards laufen soll, geht es nicht ohne externe Hilfe, über die ich mich natürlich freue.

Generell ist mcForth nur mein Arbeitswerkzeug zum Erforschen und Programmieren des entsprechenden Prozessors. Deshalb werde ich auch in Zukunft andere Forth–Versionen wie z.B. das mecrisp mit meiner Erfahrung bei Initialisierung und Ansteuerung der Schnittstellen, aber auch in Bezug auf Applikationen und Tools unterstützen. Auch der von mir betriebene, aber zur Zeit kaum genutzte Vertrieb der Forth–Gesellschaft ist weiterhin verfügbar.

Was der Swap bei mir erfuhr.

Karsten Roederer

Bekanntlich ist der bronzene SWAP ein Wanderpokal. Je ein Jahr lang weilt er beim jeweiligen Drachenträger, schaut ihm über die Schulter beim täglichen Tun, und erfährt auch sonst so einiges. 2015 kam er zu Karsten. Sie haben geplaudert, dabei Erinnerungen ausgetauscht und in Fotos gekramt und so einiges zum Thema Forth nachgezeichnet. (mk)

Es begab sich zu der Zeit[1], dass ein desillusionierter Reisender nach einem halbjährigen Neuseelandaufenthalt zurückkam und hierzulande[2] frische Kontakte knüpfte, geboren aus „Auswanderer Connections“. Das Hemmnis für den Verbleib bei den Antipoden war das Diplom jenes Elektroingenieurs, Schwerpunkt Übertragungstechnik, ausgestellt von der FH Hamburg. Das stellte die neuseeländischen Behörden vor das Dilemma einer Gehaltseingruppierung, nach altem Muster oder nach neuem Muster war die Frage, very british. Nach vier Monaten der Diskussion wurde die Entscheidung da unten wieder auf unbestimmte Zeit vertagt und eine angebotene Beschäftigung bei der New Zealand Railway damit unmöglich. So verschlug es ihn, wieder hier im Lande, per mündlicher Auskunft zur Firma System Partner GmbH.

Abbildung 1: Uwe Schwarz und Wolf Gevert in der Bahn

Dort traf er auf die Herren Uwe Schwarz und Klaus Schleisiek. So sah man als Herr damals aus - Abb.1+2. Es war Klaus Schleisiek, der damals aus dem frisch gebackenen Analogtechniker einen halbwegs brauchbaren Fachmann für eingebettete Systeme machte.

Abbildung 2: Klaus Schleisiek, auch in der Bahn

Zu entwickeln galt es eine Blumenbank für Rechtsanwälte. Beinhaltend einen Z80 Rechner mit einer USV[3]. Es war Zusatzbedingung für die ABM[4], dass er Forth lernen sollte, wovon derjenige noch nie gehört hatte - an der FH wurde damals Algol mit Teletype-Maschinen programmiert.

Der Z80-Rechner dafür musste von dem Probanden selbst aufgebaut werden, ein Nachbau der Compulady[5]. Der Aufbau mit 5 1/4" Laufwerk klappte denn auch schon ganz gut, nachdem ein „765“ Floppy-Disk-Controller besorgt war. Was sich als schwierig herausgestellt hatte, weil IBM alle vom Markt weggeklaubt hatte für den entstehenden IBM-PC. Allein das 8 1/2" Laufwerk wollte nicht mit Klaus und auch nicht mit dem Probanden. Erst nachdem eine Frequenz - sie war mit dem einfachen analogen Oszilloskope nicht richtig zu messen - einfach mal verdoppelt wurde, lief das dann auch. Mit den Laufwerken konnte es auch anders herum gewesen sein.

Mit *Starting Forth*[6]bewaffnet ging es ans Programmieren und gipfelte, nach einer Telefon- und Adressverwaltung, in der Anpassung eines Lagerhaltungsprogramms - alles in Forth. Dann traf man sich, um die Gründung der Forth-Gesellschaft auf dem Dachsberg[7] in Angriff zu

[1] 1983

[2] Deutschland

[3] Unterbrechungsfreie Stromversorgung

[4] Arbeitsbeschaffungsmaßnahme. Das waren in Deutschland zu Zeiten hoher Arbeitslosigkeit von der Arbeitsagentur bezuschusste Tätigkeiten.

[5] Compulady von Feltron, Z80A, gebaut ab 1982.05 (http://www.computer-archiv.de/)

[6] http://www.forth.com/starting-forth/

[7] In Kamp-Lintfort 1984

nehmen. Klaus Schleisiek war, denke ich, der Hauptinitiator.

Danach wurde gemeinsam der FBR 1200 in Angriff genommen, was in das erste postzugelassene Modem mit Selbstwähleinrichtung mündete. Benötigt wurde es für die Anbindung ans DAKOSY[8]. Das war ein neues System für Tallymänner[9], um Schiffsladungen zu managen. Der IBM-PC war noch nicht erfunden, sodass als Terminal „TI-PCs“ (Z80 mit CPM) zum Einsatz kamen. Das, was Big Blue für 500.000 DM anbot, sollte für ca. 70.000 DM pro Tallymann angeboten werden. Das missfiel Big Blue aber, sodass die Tallymänner gezwungen wurden von den - leider mündlich geschlossenen - Verträgen Abstand zu nehmen. Mit der Vermarktung des FBR wurde dennoch begonnen und ein Vertriebsmann diesbezüglich zu Rate gezogen, leider ohne Erfolg.

So nahm der Proband eine Stelle beim Institut für Geophysik (IGH) an. Ozeanbodenseismographen (OBS) wurden sein Plaisir. Die Arbeit mündete in ein digitales Gerät, das 6000m tief tauchen und tagelang seismische Daten auf DAT-Bändern aufzeichnen konnte (Abb.3).

Abbildung 3: OBS-Aussetzen bei langsamer Fahrt des Schiffes

In dem roten Plastik-Gehäuse befand sich eine Glaskugel für den Auftrieb aus 6000m Wassertiefe. Unten der Metallklotz, der das Gerät abtauchen lässt. Links davon, am Rahmen befestigt, das Hydrophon. Darüber der 433-MHz-Sender (Dauerstrich). Rechts davon die Auslöseuhr für den Auslösehaken. Im Aluzylinder die Registriereinheit. Ein Blitzer ist nicht bestückt, da es in der Gegend dort um Mitternacht nicht viel dunkler ist als auf dem Bild, das im Scoresby Sund aufgenommen worden ist.

Abbildung 4: OBS, geöffnet

So sah ein OBS von innen aus (Abb.4), bestückt mit NC4000 und DAT-Interface - noch ohne Datenkompression, also mit 20 kHz Bandbreite. Anfangs mit NC4000 (von Flesch) und später mit RTX2000-Prozessoren bestückt wurden die Geräte erfolgreich in den Einsatz geschickt.

Die Geräte funktionierten. Doch die Institute AWI und Geomar stellten auch Anträge auf Ingenieurstellen zur Entwicklung entsprechender Geräte. Unter Verweis auf funktionierende Geräte des IGH wurden die Nachfolge-Anträge des Probanden von deren Gutachtern negativ beurteilt. Warum die DFG[10] jene beiden Gutachter (von dreien) auswählte, blieb ein Rätsel. So mussten die begonnenen Projekte des IGH fallen gelassen werden. Und die Geophysik wurde doch nicht kostengünstiger gestaltet. Die geplante Entlastung der Cray durch graphische Tools zur Stapelung von Rohdaten und eine akustische Fernauslöse der OBS wurde nicht mehr gemacht. Und der Proband begab sich nochmal in die Arbeitslosigkeit.

Nach stapelweisen Bewerbungen bekundete die Deutsche-Zähler-Gesellschaft Interesse an dem Probanden. Hier stand die Entwicklung eines gemischt analog-digitalen „USIC“ an, mit der Teilaufgabe eines kleinen Rechenwerkes mit I^2C-Interface für das Einladen und Überprüfen von Abgleichdaten des Time-Division-Multiplikation-Messchips (TDM). Diese Stelle war erheblich besser dotiert, aber, und immer noch, vollständig über Steuermittel finanziert. Auch hier kam für das benötigte Abgleichprogramm Forth zum Einsatz. Das sollte, nachdem es einwandfrei funktionierte, in C umgeschrieben werden, womit der Proband sich schwer tat. Er hasst C wegen dessen schlechter Abbildung von Vorgängen in einem Prozessor. Dank der umfangreichen Hilfe von Ulrich Hoffmann - unter anderem gestaltete der für die Software-Mitarbeiter der DZG einen C-Grundkurs - gelang auch dies. Im Grunde genommen schrieb er das komplette C-Programm.

Eine Zufallsbekanntschaft - sie konnte, dank verbaler Penetranz heiße Luft verkaufen - brachte Kunden mit

[8] DAKOSY AG, DatenKomunikationSysteme, seit 1982; bietet Lösungen für die internationale Speditionsabwicklung (See, Luft) und die Zollabwicklung (ATLAS, EMCS, ICS, Europa) an.

[9] Ladungskontrolleur im Seehafen

[10] http://www.dfg.de/

Aufträgen für Bandmittigregelung und Näpfchenvermessung und führte zu einem erneuten Versuch der Selbständigkeit mit Forth. Das mündete in die G+R Elektronik GmbH. Es wurden erfolgreich RTX2000-bestückte Messsysteme entwickelt und verkauft. Für eine einigermaßen funktionierende Software sorgte auch hier wieder einmal mehr Ulrich Hoffmann, der den Spaghetti-Code des Probanden aufgriff und zum Funktionieren brachte. Auch dabei war es hilfreich, dass das Entwicklungssystem auf dem Target vorhanden war, also Forth. Und daher vor Ort schließlich eine Halle vor dem drohenden Einsturz bewahrt werden konnte, weil aberwitzige Vorgaben des Kunden nicht passten. Das konnte damals natürlich schleunigst geändert und an der Maschine, in diesem Falle eine mitlaufende Schere für Stahl-Coils, so um die 40 t pro Coil, getestet und im Target gleich dauerhaft hinterlassen werden. Doch dann führte eine nicht nachvollziehbare Eingebung des Mitgesellschafters dazu, dass der Kunde Dätwyler von seinen Bestellungen, acht Geräte Stücker 65.000,- DM, wieder Abstand nahm. Der Mitgesellschafter hielt den Einsatz von nicht mehr lieferbaren Prozessoren, was ja zu diesem Zeitpunkt gar nicht stimmte, für zu riskant. Man trennte sich.

So kam es zum Einsatz bei Mannesmann in Wetzlar, wo die Überführung des Sourcecodes des ersten vermarkteten Navigationsprogramms von Philips auf einen neuen Prozessors anstand. Alles in C mit f?get, makegen, replace etc. Diesmal alles ohne Forth. Das währte ein ganzes Jahr.

Dann der Einsatz bei Deuta (das ist das D von VDO) in Bergisch Gladbach. Drei PICs sollten den Betrieb eines Tachos für die Führerstandsanzeige einer Lokomotive ermöglichen. Hier kam immerhin ein bisschen Forth zum Einsatz. Diese Arbeiten ermöglichte der Sklavenhändler EDM.

Mit dem nächsten Sklavenhändler, Silver Atena, ging es wieder zurück nach Hamburg und zu Airbus. In der Abteilung Gas - Wasser - Scheisse verhinderte der Proband, dass Letzere einfriert. Da oben in der Luft sind gern mal -65°C. Auch hier kam Forth zum Einsatz für die Registrierung von Gerätedaten der Ice Protection Control Unit (IPCU) der 340-600 und A380 Airbusse. Highlight war neben den vielen Flight-Tests die Teilnahme an der Cold Weather Campaign mit Flug nach Iqaluit (Nunavut, 0,02 Einwohner/km^2) mit der A380. Dank zwei Dutzend vertauschter Sensoren von ca. 180, neben anderen Problemen, war eine Rückkehr mit dem Flieger für den Probanden erst mal nicht möglich. Erst nach etlichen äußeren „Lagerfeuern“, ließ sich der Flieger dann doch mit frischem Nass befüllen und der elektrische Strom war auch wieder da und so konnte am Ende der Woche die Heimreise mit diesem Flieger angetreten werden. Nach 12 Jahren Airbus hatte der Mohr als LAK[11] dort aber seine Schuldigkeit getan und durfte, weiterhin in Finkenwerder, jedoch nicht mehr auf dem Airbus-Gelände, diverse Projekte vom Büro aus tätigen.

Hier endet die Forthgeschichte erst mal - vorläufig. :-)

Und den SWAP habe ich 2016 weitergegeben an den nächsten Würdigen.

Abbildung 5: Der Drachenträger 2016 und die Tagungsteilnehmer an der Computer Skulptur der Hochschule in Augsburg

[11] Leiharbeitskraft

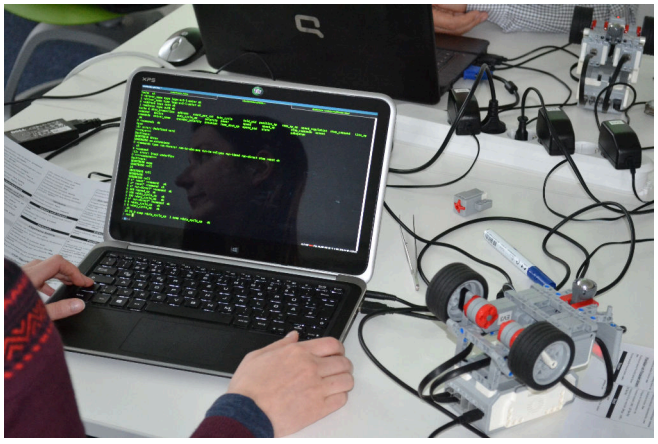
Meine Sicht auf die Forth–Tagung 2016 in Augsburg

Martin Bitter

Die diesjährige Tagung fand in den Räumen der Hochschule Augsburg statt. Ein Novum! Es bot sich die Gelegenheit, in lockerer Zusammenarbeit mit einigen unverzagten und hilfsbereiten Hochschulmitarbeitern die Programmiersprache Forth einem breiteren Publikum bekannt zu machen. Zeitgleich fanden die RetroPulsiv und der Augsburger Linux–Info–Tag statt.

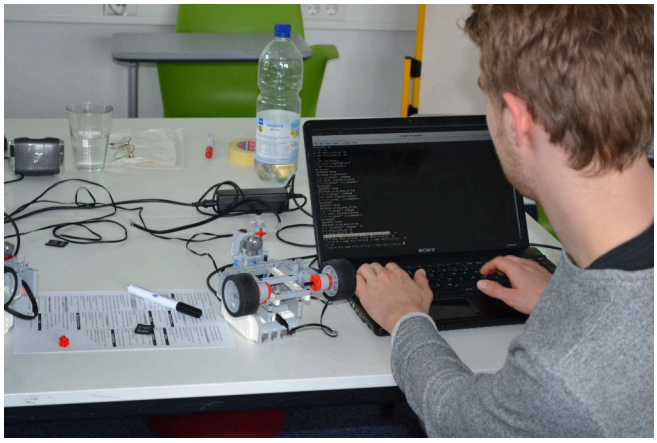
Der EV3 Workshop ...

Ich hatte mich entschlossen, einen Workshop mit der aktuellsten Version des Lego–Mindstorm–Sets – dem EV3 – anzubieten, um Forth unter 'das Volk' zu bringen. Dank des Einsatzes von Hochschulangehörigen, besonders Fr. Bäurle, standen EV3 in ausreichender Anzahl zur Verfügung. Voranmeldungen zu dem Workshop gab es genau *eine*! Trost aus dem Hochschulbereich: Dichte Termine, viele Fahrstudenten, die am Wochenende nicht anwesend sind, etc. Meine Regel: Solange die Anzahl der Teilnehmer nicht geringer ist, als die der Vortragenden, wird der Workshop durchgeführt. Spontan entschloss sich eine junge Frau (ähem: Geralds Frau) auch mitzumachen. Es gab also zwei Teilnehmer.



... und so lief es dann

Vorbereitet hatte ich eine Übersicht über die Geschichte von Forth, Besonderheiten von Forth, Einrichten des EV3–Roboters, Programmieren von Fahrbewegungen, ... Resümee: Die beiden Teilnehmer waren motiviert, willig und sehr umgänglich. Ich selbst werde beim nächsten Mal die Geschichte von Forth ganz fallen lassen und sofort mit dem Programmieren von Fahrbewegungen anfangen. Nicht zu unterschätzen sind die rein technischen Hindernisse, wie das Einrichten einer USB–Ethernet–Verbindung und eines Terminalprogrammes auf dem Laptop der Teilnehmer. Glücklicherweise hatten beide Teilnehmer ein Linux–System laufen. Beim nächsten Workshop sollten idealerweise EV3s und passende Laptops (Raspberry Pi) zur Verfügung stehen. (Man wird doch wohl träumen dürfen.)



Drumherum

Die Vorbereitung und Durchführung des Workshops band meine Aufmerksamkeit sehr. Von den vielen Vorträgen unserer Forthtagung konnte ich nur wenige besuchen. Am meisten hat mich Gerald mit seinem Vorschlag für ein Forth–Repository beeindruckt. Ich habe gleich mein `mword`[1] dort eingestellt. Die Retro–Leute und den Linux–Info–Tag habe ich kurz besucht, konnte aber dort keine Aufmerksamkeit für Forth oder unsere offene Tagung erzeugen.

Fazit

Mir hat der Workshop Spaß gemacht. Ich habe viel für ein nächstes Mal gelernt. Die Tagungsteilnehmer – nicht die Organisatoren – sollten sich bei ähnlichen Chancen überlegen, wie sie Publikum für *unsere* Veranstaltungen gewinnen können.

Link

`http://www.theforth.net`

[1] Näheres in der nächsten Ausgabe

Forth–Gruppen regional

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

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

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

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

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

Gruppengründungen, Kontakte

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

μP–Controller Verleih

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

Spezielle Fachgebiete

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

Termine

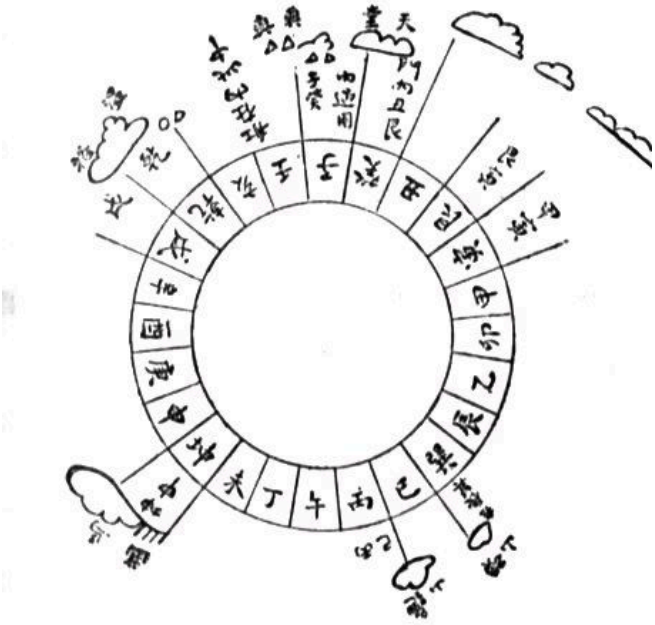
Donnerstags ab 20:00 Uhr
Forth–Chat net2o `forth@bernd` mit dem Key `keysearch kQusJ`

27.–29. Mai 2016:
Maker Faire Hannover
`http://www.makerfairehannover.com`

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

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

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



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

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

EuroForth — die internationale Forth–Konferenz

Hotel mein Inselglück, Insel Reichenau, Deutschland

9.–11. September 2016

Termine

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

Weitere Deadlines (paper deadline usw.) werden noch angekündigt

Erwartete Kosten

Expected cost:	**Delegate**	**Spouse**
Standards meeting:	360€	170€
EuroForth:	370€	190€
4th day:	100€	60€

Anreise

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

Organisator

Dieses Jahr wird die EuroForth von Klaus Schleisiek organisiert.

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