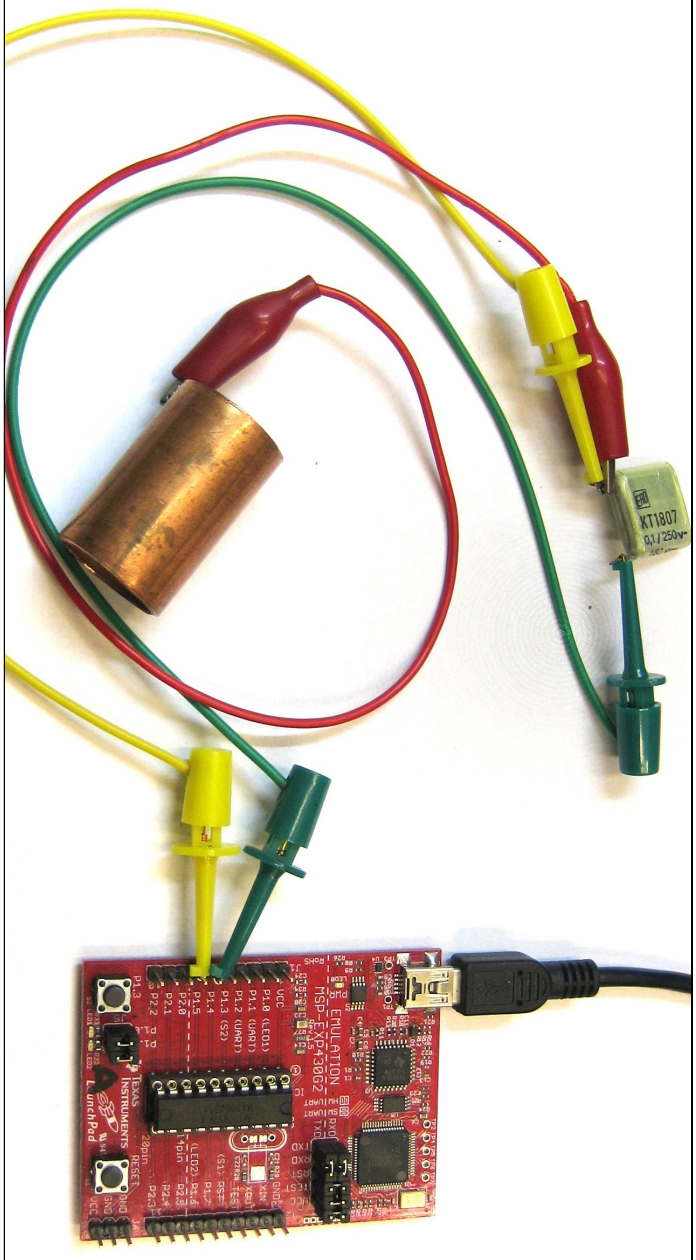
Vierte Dimension

4/2013

Das Forth–Magazin

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

In dieser Ausgabe:



Dienstleistungen und Produkte fördernder Mitglieder des Vereins

tematik GmbH
Technische Informatik

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

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

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

LEGO RCX–Verleih

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

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

Inhalt:

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

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

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

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

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

KIMA Echtzeitsysteme GmbH

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

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

FORTecH Software GmbH
Entwicklungsbüro Dr.–Ing. Egmont Woitzel

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

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

Hier könnte Ihre Anzeige stehen!

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

`Secretary@forth-ev.de`

Ingenieurbüro
Klaus Kohl–Schöpe

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

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

Impressum

Name der Zeitschrift

Vierte Dimension

Herausgeberin

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

Redaktion & Layout

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

Anzeigenverwaltung

Büro der Herausgeberin

Redaktionsschluss

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

Erscheinungsweise

1 Ausgabe / Quartal

Einzelpreis

4,00€ + Porto u. Verpackung

Liebe Leser,

Es wintert schon, und der letzte Artikel aus dem Sommerheft kommt so etwas spät zum Zug:

Die Zeiten, in denen man sein Mofa getuned hat, sind heute fast schon vorbei — Fahrräder mit Motorunterstützung sind heute elektrisch und werden besonders gerne von Rentnern gekauft. Tunen ist verboten (also, sagen wir mal: Rentner halten sich da eher daran als Jugendliche), und außerdem braucht man dazu heute Microcontroller–Knowhow. Wobei: Mit diesem Know–How kann man auch ein altes Mofa tunen, wie Karsten Roederer eindrucksvoll demonstriert. Wobei er doch nicht ganz ohne E–Bike–Komponenten auskommt...

Auch sonst kommen die Controller in diesem Heft nicht zu kurz: Ein Terminal für 4e4th, ein Matrix–Tastaturscanner, ein atmendes LED (mit MSP 430 und viel Debugging) und eine Kapazitätsmessung (ebenfalls mit dem MSP 430) füllen das Heft.

Johannes Teich debuggt, ganz ohne Controller, seine Wave Engine am PC. Jetzt klingt sie richtig sauber, nachdem sich der Versatz um ein Byte verflüchtigt hat.

Ja, und damit sich unsere Ideen nicht völlig verflüchtigen, sondern zumindest in der Cloud bleiben, fängt Gerald Wodni mit der Rubrik „Neues aus the Forth Net“ an. Da die Domain `forth.net` leider (noch?) der Foundation Of Research and Technology Hellas gehört (die übrigens auch 30 Jahre alt ist), heißt Geralds soziales Forth–Netzwerk `theforth.net`, und dient dem Sharen von Code.

Ach ja, und die Lösung des Drachenrätsels vom letzten Mal: Der Drache ist in Stein am Rhein. Da bin ich jetzt nicht extra hingefahren, um ein Beweisfoto zu machen, dafür gibt's eine URL mit hübschen Fotos: `http://dantesdame.com/temp/stein-am-rhein/`

Und weil's Drachen auf der ganzen Welt gibt...: Wo swappen diese beiden Drachen ihre Drachenpille? — *Bernd Paysan*

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
`http://fossil.forth-ev.de/vd-2013-04`

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

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

Äpfel, Birnen, Pflaumen, Zwetschgen, Kirschen, Nüsse — und Forth.

Nun, die Brücke über den Forth in Schottland unweit Edinburgh kennt jeder Forthler, also die Menschen, die sich mit der Programmiersprache Forth befassen, ja schon. Dass sich das ganze Gebiet um den Flusslauf herum inzwischen als *Forth Area* versteht, war mir hingegen noch nicht so bekannt. So wie der Begriff *Ruhrgebiet* mit dem industriellen Zusammenschluss der Region entstand, und heute ein zusammenhängendes ökonomisches und ökologisches Gebiet ist, so wird die Forth Area inzwischen ebenfalls als zusammenhängende Region wahrgenommen. Die Initiative *Forth Environment Link (FEL)* macht das deutlich. Sie machen interessante Projekte: So die Zero Waste Communities Initiative (Null–Müll–Initiative) und das Stirling Cycle Hub (Stirling–Rad–Zentrum). Anregende Lektüre wünscht mk. `http://www.forthenvironmentlink.org/`

Zweikanaloszilloskop in Forth *Mecrisp–Stellaris*

Es ist viel geschehen seit der Ankündigung von Mecrisp für MSP430! Nachdem mir Michael Kalus ein TI Stellaris–Launchpad geschenkt hat, ist *Mecrisp–Stellaris* für die ARM Cortex–M–Familie entstanden, welches mittlerweile auf Chips von TI, STM und Freescale läuft. Wer bereits Mecrisp auf dem „kleinen“ Launchpad ausprobiert hat, wird sich so auch in der ARM–Welt schnell heimisch fühlen können. Für einen ersten Einstieg ist das Stellaris–Launchpad von TI sehr empfehlenswert, welches eine ausgewogene Mischung von Fähigkeiten und Komplexität besitzt — eins der Beispiele implementiert die Basis für ein Zweikanaloszilloskop mit je 1 MHz Abtastrate direkt in Forth. Wer gerne Arduino–Erweiterungen anschließen möchte und mit weniger Speicher und Rechenleistung auskommt, für den wird die Freescale Freedom FRDM–KL25Z Platine unterstützt, die dafür im Gegensatz zum Stellaris–Launchpad einen analogen Ausgang bietet und für dessen kleinen M0–Prozessorkern ein Disassembler zur Verfügung steht. Ganz viel Rechenleistung, sehr viel Speicher und analoge Fähigkeiten vom Feinsten sind auf dem STM32F4 Discovery versammelt — diese Platine ist allerdings auf Grund ihrer erheblichen Komplexität und verwirrender Funktionenfülle nur für Fortgeschrittene zu empfehlen. Weitere werden bald folgen, Mecrisp–Stellaris kann bereits jetzt für M0, M3 und M4–Kerne assembliert werden und eine Portieranleitung liegt bei. Viele Ideen, die auf dem MSP430 sehr erfolgreich gewesen sind, wurden übernommen – und so steht der gewohnt komfortable Umgang mit Flashspeicher sowie die kleinen Feinheiten im Hintergrund wie Konstantenfaltung und direkte Übersetzung in Maschinensprache nun auch für ARM Cortex–Chips zur Verfügung. Helferinnen und Helfer sind gern gesehen — insbesondere die Fülle an Möglichkeiten auf den schon jetzt unterstützten Chips bedarf noch vieler weiterer Forth–Beispiele.

— Matthias Koch

`http://mecrisp.sourceforge.net/`

> Mecrisp is an implementation of a standalone native code Forth for MSP430 microcontroller chips... Mecrisp–Stellaris is the younger sister of Mecrisp and mastered the jump to the ARM Cortex M architecture...

SVFIG Meeting Notes vom Forth Day

Unscheinbar auf der Homepage, aber gehaltvoll, sind die Notizen zu den Treffen der *Silicon Valley Forth Interest Group*. Die Gruppe trifft sich nach wie vor fast monatlich, und die Meeting Notes werden seit 1999 geführt. Einmal jährlich ist Forth Day. Am 16.11.2013 kamen 36 Teilnehmer dahin. Die Themen waren:

Implementing APL concepts in Forth — Bob Armstrong
Cortex–M4 SwiftX — Leon Wagner
Break & The LED Cube — CH Ting
64-bit Standalone Forth System — Stefan Mauerhofer (from Switzerland)
Simple Scalable Fonts in Forth — Brad Nelson
The Flight of the Kestrel II — Samuel A. Falvo II
Green Arrays — Multiple Presenters
- Intro: The State of Green Arrays — Greg Bailey
- Low Frequency Clock and UART — Stefan Mauerhofer
- Roots and Other Memories — John Rible
- XXth Century Arithmetic (1900)
- Synthesis–Aided Compiler — Mangpo Phitchaya Phothilimthana
- Building Boot Streams — Charlie Shattuck

Fireside Chat — Chuck Moore

`http://www.forth.org/svfig/kk/11-2013.html`
`http://www.forth.org/svfig/index.html`

Gruppenfoto siehe Seite 30.

mk

Neues auf the Forth Net

Gerald Wodni

Viele bekannte Programmiersprachen haben ein großes Portal zum Sourcecodetausch, JavaScript hat [npm], PHP hat [PEAR], TEX hat [CPAN].

the Forth Net[tFN] ist der Versuch, so etwas für Forth zu erschaffen.

Der Code muss weder schön sein, noch einem Standard gerecht werden, und er braucht nicht einmal zu compilieren! Es ist schon toll, wenn ein Suchender eine Idee bekommt, wie man ein Problem löst, oder einfach nur Spaß daran hat, durch alten Code zu stöbern, und sich anzusehen, wie man ein Problem in „den guten alten Zeiten“ angegangen ist.

Manche Projekte lassen sich aber ganz leicht auf aktuellen Forth–Systemen nutzen, und um auch hier an den Erfolg anderer Programmiersprachenwebsites anzuknüpfen, gibt es unser Pendant zu npm, und zwar [fget].

`fget <name>` lädt zum Beispiel das Projekt herunter, und installiert es auf dem Rechner lokal in einen eigenen Ordner.

`frun <name>` lädt das Paket nur in den Zwischenspeicher, und führt es gleich aus.

`finfo <name>` zeigt Informationen zu einem Projekt an.

`fsearch <name>` rundet das Angebot ab, und erlaubt dem Benutzer, direkt aus Forth heraus Projekte zu suchen, man braucht also nicht einmal einen Webbrowser, sondern kann auch alles von Forth aus machen.

fget funktioniert im Moment nur unter Gforth, aber das kann man leicht ändern, ich freue mich über jeden Vorschlag.

Bernd Paysan hat Gforth auf Android portiert, aber nicht jeder Nutzer möchte auf seinem Handy programmieren, sondern vielleicht einfach ein kleines Forth–Programm oder –Spiel darauf ausführen. Um dies zu erleichtern, haben wir auf der EuroForth 2013 damit begonnen, so etwas wie einen App Store zu implementieren. Die Idee ist, dass man ein Programm auf the Forth Net online stellt, als App markiert, und diese dann ganz einfach in Gforth auf dem Handy suchen und herunterladen kann.

Wer also neugierig geworden ist, der kann gleich einmal darin herumstöbern, oder besser: seinen eigenen Code hochladen (ganz egal was).

Dieser Artikel soll die Kolumne „Neues auf the Forth Net“ starten, in der ich neue Projekte vorstellen möchte. Den Anfang machen die folgenden drei:

volksFORTH [vF] Manager: Carsten Strotmann
Ein 16-Bit–Forth–System entwickelt von der Forth–Gesellschaft e.V. von 1985–1989 und 2005 wiederbelebt. Carsten hat gleich 3 Versionen davon online gestellt:

- Atari ST
- CP/M
- MS–DOS

Ich bin leider zu jung, um diese Versionen auf ihren Originalsystemen in Aktion gesehen zu haben, dennoch war es interessant zu sehen, wie das System aufgeteilt, und wie der Editor programmiert ist. Außerdem habe ich gemerkt, dass es schön wäre, die Screens direkt im Browser korrekt darstellen zu können, um das Stöbern einfacher zu machen.

miniterm[mt] Manager: Carsten Strotmann
Kleines Beispiel zur Implementation eines Fileinterfaces über die serielle Schnittstelle von Klaus Kohl.

Die Schnittstellenbefehle stammen aus dem PC–volksFORTH 3.81 von Klaus Schleisiek. Sie wurden weitgehend unverändert übernommen, sind aber auf 4KByte–Puffer erweitert.

Euler 303 [e303] Manager: Anton Ertl
Project Euler `http://projecteuler.net/` ist eine Website die interessante mathematisch–informatische Probleme beherbergt. Jedes Problem ist mit einer Nummer versehen, und man sieht, wie viele Leute es schon gelöst haben. Hier ist das Problem 303 „Multiples with small digits“ in wenigen Zeilen mit Forth gelöst.

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

Referenzen

[tFN] `http://theforth.net/`

[fget] `http://theforth.net/projects/fget`

[vF] `http://theforth.net/projects/VolksForthAtariST`

[mt] `http://theforth.net/projects/miniterm`

[e303] `http://theforth.net/projects/euler303`

[npm] `https://npmjs.org/`

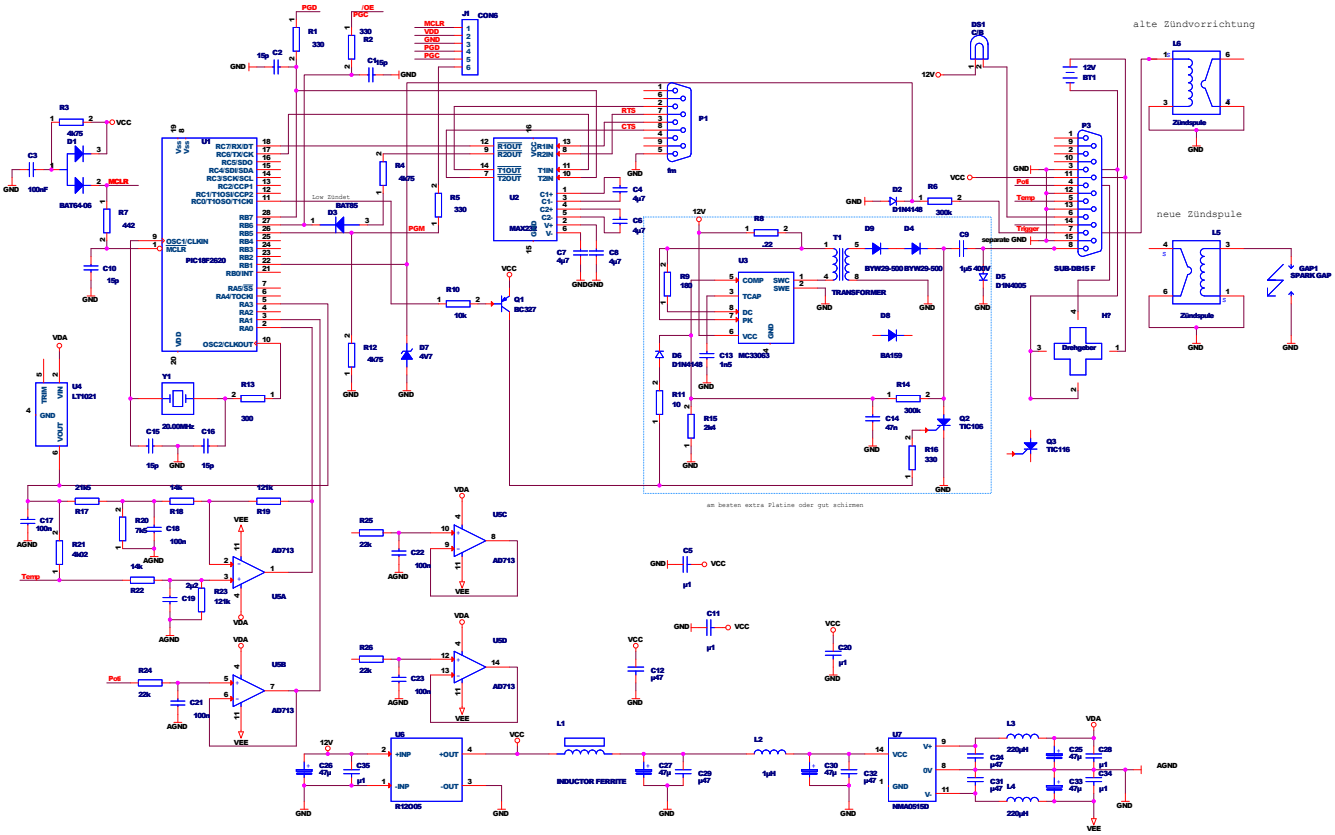
[PEAR] `http://pear.php.net/`

[CPAN] `http://www.cpan.org/`

Zündung mit Handverstellung

Karsten Roederer

Nach einigen Umbauten an der Zündanlage meines Bromfietses — ein Leichtmofa (Saxonette von Hercules mit 30 ccm Motor) — und dreifachem Ersatz der Zündspule (stücka 120,—€), reift eine alternative Zündanlage unter Einsatz eines PIC µC in meinem Keller und bringt meine Betriebserlaubnis zum Erlöschen. Theoretisch sollten statt der 0,7 PS bis zu 9 PS möglich sein, ohne dass ein Kompressor zum Einsatz käme.



Feature

Diese Zündanlage kann in allen Ottomotoren eingesetzt werden. 4 OPVs sind für analoge Signalverarbeitung vorgesehen (Herr Aliasing lässt grüßen), an denen solche Scherze, wie Unterdruckversteller, Temperatur oder Ansaugunterdruck (Lastsensor) angeschlossen werden können. Das kommt für mich erst mal nicht in Frage, da das Bromfiets derlei Schnickschnack nicht unterstützt. Aus Unkenntnis habe ich mich für die manuelle Zündverstellung entschlossen und dafür einen E–Bike Gasgriff vorgesehen, der mit einem Hall–Element standardmäßig ausgerüstet ist. Der hat also einen Spannungsversorgungseingang und einen Spannungsausgang der so ca. 1 — 3 V (#100 - #900 sagt der A/D–Wandler im PIC), je nach Drehwinkel des Griffs liefert. Den gemessenen Spannungs–Bereich habe ich mir in der Software in 4 Teile gesplittet (so wie sie aus den A/D–Registern kommen):

1. < #200 alten Zünd–Winkel beibehalten
2. #200 bis < #400 erhöhe den Frühwinkel um 1°
3. #400 bis < #700 erniedrige den Frühwinkel um 1°
4. > #700 reset auf default–Winkel

Das Bromfiets arbeitet mit einer Kondensator–Zündanlage (CDI), wie sie sogar in den Sternmotoren für Flugzeuge vorgeschrieben sind, und die gelten eigentlich als unverwüstlich. Das Bromfiets bildet da eine unrühmliche Ausnahme, da es im Sommer (hoffentlich kommt er endlich mal) für die Spulen (3–4 Wicklungen, um genauer zu sein) wohl zu heiß wird in dem kaum belüfteten Einbauraum.

Für den „Nordpuls“ (in diesem Falle 17° vor OT mit der abfallenden Flanke) muss die alte Zündspule weiterverwendet werden. Für einen anderen Nordpuls–Geber muss ev. der Eingangswiderstand erniedrigt und seine Grad–Zahl vor OT in der Software (ist da eine Konstante) eingetragen werden.

Von der Spannungsgenerierung (ca. 160V) befreit, muss nun die Hochspannung aus einer anderen Energiequelle, sprich Batterie (hier 12V) transformiert werden. Dazu griff ich erst einmal zu einer Nähmaschine, die einen Unterfaden wickeln kann. Verwendet wurde ein Ferritspulenkörper von Konrad #: 51 66 43-88

Pri: 25 Umdrehungen mit 0,5mm Kupferlackdraht

Sec: 160 Umdrehungen mit 0,15mm Kupferlackdraht

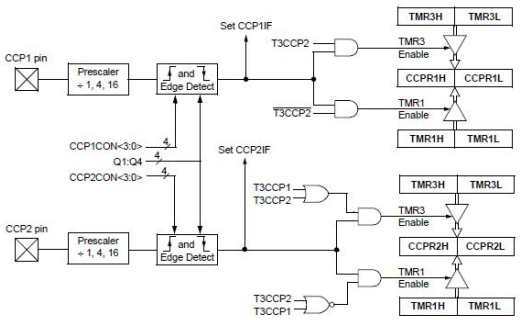
Die beiden Ferritelemente sollten mit einem Stück Papier getrennt sein. Dazu ein Spannungswandler–Kontroller in Form eines MC33063 und ein bisschen Diskretes. Das hatte ich als erstes aufgebaut und getestet. Ergebnis: lief.

PIC $\Longrightarrow$ FlashForth

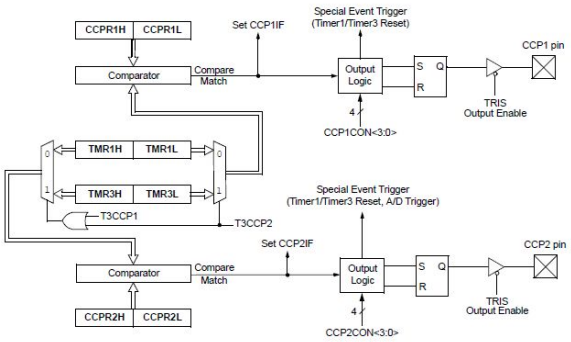
Ein PIC μC mit der Bezeichnung PIC18F2620 wurde mit dem FlashForth von Mikael Nordmann versehen — erst die Version 3.6, dann 3.8, jetzt wieder 3.6 — und die einzelnen Themen programmiert. In der Schaltung ist eine ICSP–Schaltung (In Circuit Seriell Programming) vorgesehen, falls eine Platine mit aufgelötetem Kontroller zum Einsatz kommt.

Arbeiten mit dem PIC

Der PIC ist mit Capture/Compare

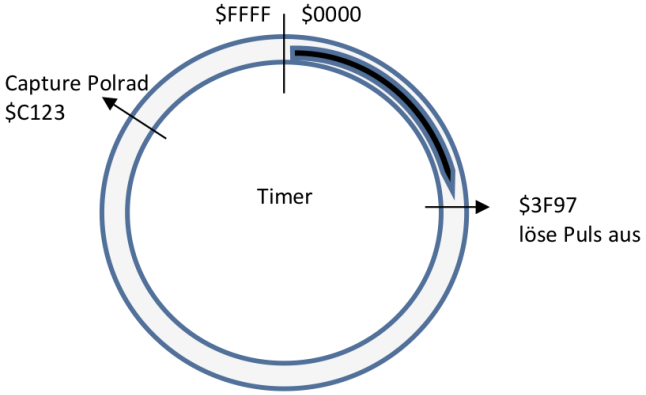


und PWM:



ausgerüstet. Dabei ist es für diese Anwendung wichtig, dass der Timer1 immer durchläuft. So treten quasi keine Echtzeitprobleme in der Software auf. Das Capture–Register captured, wenn der Nordpuls kommt, und wann PWM auslöst, kann ich mir fast eine Runde (Kubelwellenumdrehung) lang überlegen. Ich hoffe niemand hat ein Problem damit, wenn ich jetzt erst die Berechnung für den nächsten Puls vornehme. Es soll ja Leute geben, die bauen das Polrad um, um genügend Vorlauf für die Software zu bekommen. Um sozusagen, dicht am Geschehen, in derselben Kurbelumdrehung auf ev. Geschwindigkeitsänderungen der Kurbelwelle noch zu reagieren.

Erst beim Finish der Programmierung des PIC fiel mir auf, wie immens wichtig es für diese Applikation ist, dass das Capture– wie auch das PWM–Register unabhängig voneinander von ein und demselben Timer bedient werden, der dabei nicht unterbrochen wird in seinem Tun und durchläuft:



So könnte die Situation z. B. aussehen bei einem Überlauf des Timers, der dann zu berücksichtigen ist.

Software

Kommen wir zur Programmierung des Themas. Das FF bietet Interrupt–Programmierung und Bit–Manipulationen auf Forth–, wie auch auf Assemblerebene.

Konstant oder variabel

Hier ein paar Konstanten und Variablen, die von der CPU–Clock (Damit zählt ja der Timer) unabhängig sind.

```
1  tmr0ie intcon mclr \ disable the tmr0 irq to prevent crash
2
3  \ false di to irq ei \ zero the interrupt vector
4  -zuendung
5  marker -zuendung
6  decimal ram
7    #2280 constant idling \ engine idling
8    #3750 constant maxrpm20 \ max rpm at 20 km/h
9    #0017 constant tdccorrector   \ 17° trigger before top dead cente
10   #0030 constant maxdegree      \ 30° max pre-ignition
11     #32 constant prescale \ fit with t1con !
12     #04 constant cnts  \ cnts for hand-throttel stay time
13 variable channel
14 variable allad  \ all sampels in the array discarded
15 variable igcountdown \ ignition count down is set
16 variable period \ actual rotation speed (pr1)
17 variable onedegree \ counts for 1° by actual rotation speed
18 1 onedegree !
19 variable degrees \ degrees through pre-ignitioner
20 1 degrees !
21 variable res1grad \ cnts for reset to 1°
22 variable tunichts \ cnts for doing nothing more
23 variable frueher \ cnts for      pre-ignition
24 variable spaeter \ cnts for retarded-ignition
25 maxdegree tdccorrector -
26 constant maxdegreer
27
28 Dann fehlt dem FF noch das case? zum Decoding:
29 : ?exit         ( F1 -- )
30                 postpone  if
31                 postpone  exit
32                 postpone  then ; immediate
33
34 : case?         over = dup invert ?exit nip ;
```

Initialisierung

Die ev. überflüssige ADC–Initialisierung für eine Interrupt gesteuerte unabhängige Sampel–Sammelei. Im Forth–Vordergrund wird nur noch geschaut, ob wieder neue Werte vorliegen. Man beachte, wie das FF auf inline–Assembler umschaltet. Mit [geht's in den Assembler, der mit] wieder abgeschlossen wird. Wegen der Bedeutung der Bits und Befehle will ich nur auf das PIC–Datenbuch (Microchip, 2008) verweisen. Die Sequenz adgo weiter unten z.B. soll das bit1 im adcon0–Register setzen (bsf), was die A/D–Sampelei in Gang setzt.

```
35  : array create cells allot does> swap 2* + ;
36   ram #10 array sampel
37
38  : adgo   [ adcon0 1 a, bsf, ] ; \ A/D go
39  : cmpgo  [ pie1   2 a, bsf, ] ; \ enable CCP1IE interrupt
40
41  \ ADC-Initialisierung
42
43  : adinit ( -- )
44    [ porta a, clrf, ]
45    [ lata  a, clrf, ]
46    %11010010 adcon2 c!
47     \ right justified 32 Tosc 4TAD for 20 MHz; for 8 also ok
48    %11000101 adcon1 c! \ set AN0-AN9 as A/D pins
49    %00111111  trisa c!   \ AN0 - AN5 input (future use)
50    [ trisb 2 a, bsf, ] \ AN8 input
51    [ trisb 3 a, bsf, ] \ AN9 input
52    [ trisb 4 a, bcf, ] \ RB4 output for test use
53    [ trisc 1 a, bsf, ] \ CCP2 inp, 17° Trigger falling edge
54    [ trisc 2 a, bcf, ] \ zuendpuls
55    [ adcon0 0 a, bsf, ] \ A/D on
56    [ pir1   6 a, bcf, ] \ clear A/D interrupt
57    ccp1ifm intcon mtst drop
58    0 allad !
59    0 channel !
60     ;
```

Die Timer–Initialisierung sieht so aus:

```
61  \ Timer-Initialiserung
62
63  : timerinit
64    [ t3con 7 a, bsf, ] \ 16-bit operation
65    [ t3con 6 a, bcf, ] \ timer 1 is clock source for ccp1+2
66    [ t3con 3 a, bcf, ]
67    %11111001 ccp1con c! \ compare and falling edge of ccp1 pin
68    %11110100 ccp2con c!
69     \ capture every falling edge of ccp2 pin
70    [ ccpr2l a, clrf, ] \ clear capture2 data low  reg
71    [ ccpr2h a, clrf, ] \ clear capture2 data high reg
72    %10110101 t1con c! \ Timer1Konfig :8 prescaler
73    [ tmr1l a, clrf, ] \ clear Timerreg
74    [ tmr1h a, clrf, ] \ clear Timerreg
75  ;
```

Interrupt

Die Interrupt–Routine ist nun das Filet–Stück der Anwendung, da hier schon alle Echtzeit–Probleme erschlagen werden. Sie beginnt mit einem „[i“. Das heißt nichts anderes, als dass der Code, der jetzt folgt, an Adresse $0008 (der sogenannte Interrupt–Vektor) abgelegt wird. Abgeschlossen wird mit der Sequence „i] ‚i“. Der 18er–PIC kommt zwar auch mit einem „low priority interrupt vector“ an Adresse $0018 daher, den ich aber noch nicht verwendet habe. So eine Art priorisiertes Multitasking auf Interrupt–Ebene (zzzzhh..). Dabei priorisiere ich doch schon durch die Reihenfolge der Interrupt–Flag–Abfrage!? Der Code wird dadurch doch nur undurchschaubarer. War wohl ein Bug im Gehirn des Designers, der jetzt als Feature verkauft wird.

```
76  : irqserv ( -- ) \ interrupt service routine
77    [i
78     ccp1ifm pir1 mtst \ is it ccp1if?
79     if [ pir1    2 a, bcf, ] \ clear ccp1if
80        [ ccp1con 0 a, bsf, ] \ make him high
81       hopefully
82     then
83     ccp2ifm 1 pir2 mtst  \ 17° Trigger
84     if [ pir2  0 a, bcf, ] \ clear ccp2if
85        [ portb 4 a, bsf, ] \ set for test RB4
86        ccpr2h c@ 8 lshift ccpr2l c@ +
87        dup dup \ messe Periode
88        capt3 @ dup rot >
89        if swap invert 1+ + period !
90          else - period ! then \ war Überlauf
91        capt3 ! \ store new capture
92        period @ dup 360 / dup onedegree ! degrees @ * -
93          \ new igntcomptime
94         capt3 @ + dup $ff00 and 8 rshift ccpr1h c!
95          \ new compare high
96         $ff and ccpr1l c! \ new compare low
97        [ portb 4 a, bcf, ] -1 igcountdown ! \ clear
98                  for foreground routine
99     then
100    adifm pir1 mtst \ is it the ADC-Interrupt?
101    if
102       adresh c@ 8 lshift adresl c@ +
103       channel @ sampel ! \ store sample
104       adifm pir1 mclr
105       channel @ 1+ dup channel ! \ increment
106 A/D-Channel
107       dup
108       1 > if drop 0 dup channel ! -1 allad ! then
109         \ only channel 8 & 9
110       $8 + 2 lshift \ here only channel 8 & 9
111       adcon0 c@ $c1 and or adcon0 c!
112       [ adcon0 1 a, bsf, ]      \ go
113    then
114    i]
115    ;i
```

Da die Interrupts noch disabled sind, passiert jetzt noch nichts, außer dass der Code da jetzt steht, der dann zur Ausführung kommt. Die Interrupts können erst zuschlagen, wenn sie enabled sind.

Foreground

Der Vordergrund beschäftigt sich mit den Dingen, die immer noch über die serielle Schnittstelle kommen können, und wertet das im Interrupt Geleistete aus. So steht an erster Stelle natürlich die Auswertung der vom ADC kommenden Signale an. Die Werte des handelsüblichen E–Bike–Gasgriffs stehen zur Auswertung an. In einer Schleife die zur Zeit mit einem $4000 for next beginnt, da der ms–Timer nicht mehr tut, werden die mit dem jetzt am Lenker links befestigten Griff erzeugten Werte angeschaut und dekodiert. Wird nicht dran gedreht, bleibt der status quo. Ein bisschen drehen erhöht in jeder Schleife um ein Grad. Ein bisschen mehr erniedrigt um ein Grad. Vollgas stellt zurück auf default. Dabei werden die Werte erst verstellt, wenn der betreffende Bereich cnts mal (hier jetzt 4) anstand, da es ja unvermeidlich beim Einstellen zu Überstreichungen nicht gewollter Bereiche kommt.

```
116 : start0   ( -- )
117   timerinit
118   cr ." tmroif" cr
119   adinit
120   [ pir2 0 a, bcf, ] \  clear CCP2IF interrupt
121   [ pir1 2 a, bcf, ] \  clear CCP1IF interrupt
122 \  [ pie2 0 a, bsf, ] \ enable CCP2IE interrupt
123                      \ führt zu Absturz
124   ['] irqserv di is irq ei ." EI" cr / Interrupt läuft
125   tmr0ifm intcon mclr
126   tmr3ifm pir2 mclr
127   adgo
```

```
\  cmpgo Absturz
  cr ." GO" \ jetzt läuft die A/D-Wandelei
 ;

: under   ( n1 n2 -- n2 n1 n2 )   swap over ;

: m/mod   ( d n -- rem quot )
dup >r  abs over 0<
if under + swap then um/mod r@ 0<
  if negate over if swap r@ + swap 1- then then rdrop ;

: cp   ( -- n )                   \ compare @ for test
     ccpr1h c@ 8 lshift ccpr1l c@ + ;

: cntreset   ( -- ) \ resets the hand-throttel
     0 tunichts ! 0 frueher ! 0 spaeter ! 0 res1grad ! ;

: decode   ( -- )
    1 sampel @ dup .
    dup #200 < true case? if tunichts @ 1+ dup
                             cnts > if
                               drop cntreset
                             else tunichts ! then
                             drop exit then drop
    dup #400 < true case? if frueher @ 1+ dup
                             cnts > if
                               drop cntreset  1 degrees +!
                             else frueher !  then
                             drop exit then drop
    dup #700 < true case? if spaeter @ 1+ dup
                             cnts > if
                               drop cntreset -1 degrees +!
                             else spaeter ! then
                             drop exit then drop
    drop res1grad @ 1+ dup
    cnts > if drop cntreset 1 degrees !
           else res1grad ! then
 ;

: start ( -- )
  cwd start0
   begin
    $4000 for next \ ms funktioniert nicht mehr
    allad @
    if decode 0 allad ! then \ so wait for new ADC-inputs
 cr
    period  @ u.
    capt3   @ u.
  key?
  until
   cp u. ;

: tit  ( -- n )                \ timer @ for test use
  tmr1h c@ 8 lshift tmr1l c@ + ;
```

Ein paar auskommentierte Zeilen sind im Code noch enthalten, da sie an der Stelle zwar logisch hingehören, aber bei der Ausführung zum Absturz führten. Das gibt einen Aufschluss über die Ungereimtheiten, die einen beim Test so erwarten. Die Textausgaben bei start0 dienen zum Debuggen und geben Aufschluss darüber, bis wohin der Hase gelaufen ist. `cwd` cleart den Watchdog im PIC. `allad` dient dazu sicherzustellen, dass neue ADC–Werte vorliegen, wenn gesetzt. Das Ganze habe ich dann mit einem Frequenzgenerator getestet. Ergebnis: lief.

Ein Zusammenschalten beider Schaltungsteile offenbarte jedoch erheblich Abschirmmängel der Hochspannungserzeugung, der insbesondere stark in den Analogteil einstreute, sodass dieser Teil jetzt eines Neuaufbaus harrt. Das und einen Erfahrungsbericht werde ich dann nachreichen, sobald ich damit durch bin.

FF 3.8 ?

Mit dem FF 3.8 hatte ich in einer anderen Anwendung auch schon Erfahrungen sammeln dürfen. Es stürzte mir aber leider ab einer gewissen, ich will mal sagen: Returnstack–Komplexität schon beim Kompilieren ab, sodass ich lieber auf die ältere Version 3.6 zurückgegangen bin. Die stürzt beim Kompilieren zwar auch manchmal ab, aber spätestens beim 2. Versuch klappt's dann doch. Und das obwohl ich den UART mit Hardware–Handshake fahre.

An A/D–Inputs habe ich noch eine Temperaturerfassung mit eingeplant. Außerdem ist es denkbar weiterhin auch einen Unterdrucksensor zu verarbeiten, wenn denn vorhanden. Das Design ist für weitergehende Automatisierungen offen: Zündkennfelder verarbeiten, Mehrzylinder, komplexe Diesel-, und Benzineinspritzung, Mehrfachzündung. All das sollte den PIC noch nicht überfordern.

Source

Zum einen fehlen noch die Zuweisungen für die Register:

```
\ Register associated with Zuendung

\ tmr0ie intcon mclr     \ disable the tmr0 irq
                         \ to prevent crash
\ false di to irq ei       \ zero the interrupt vector
-regs
marker -regs
decimal ram

: as0 ( opcode "name" -- ) ( -- )
  constant
  does> i, ;

: as2 ( opcode "name" -- ) ( f a -- )
  constant
  does> rot ic, or ic, ;

$6a as2 clrf,           ( f a -- )

\ Registeradressierung

$ff92 con trisa         \ Richtungsreg
$ff93 con trisb         \ Richtungsreg
$ff94 con trisc
$ff80 con porta         \ Output-Reg
$ff81 con portb         \ Output-Reg
$ff82 con portc
$ff89 con lata          \ Latch-Reg
$ff8a con latb
$ff8b con latc
$fff2 con intcon        \ Interrupt-Control-Reg
$ff9e con pir1          \ Pheripheral-Inter-Request-Reg
$ffa1 con pir2          \ Pheripheral-Inter-Request-Reg
$ff9f con ipr1
$ff9d con pie1          \ Pheripheral-Inter-Enable-Reg
$ffa0 con pie2          \ Pheripheral-Inter-Enable-Reg
$ffa2 con ipr2
$ffd0 con rcon
$ffb6 con eccp1as       \ auto-shutdown-control reg
$ffb7 con pwm1con       \ pwm1 control reg
$ffba con ccp2con       \ Capture 2 Control Reg
$ffbb con ccpr2l
$ffbc con ccpr2h
$ffbd con ccp1con
$ffbe con ccpr1l
$ffbf con ccpr1h
$ffb1 con t3con         \ Timer3 Control-Reg
$ffb2 con tmr3l
$ffb3 con tmr3h
$ffca con t2con         \ Timer2 Control-Reg
$ffcb con pr2
$ffcc con tmr2
$ffcd con t1con         \ Timer1 Control-Reg
$ffce con tmr1l         \ Timer1 Low-Reg
$ffcf con tmr1h         \ Timer1 High-Reg
$ffd5 con t0con         \ Timer0 Control-Reg
$ffd6 con tmr0l
$ffd7 con tmr0h
$ffc2 con adcon0        \ AD-Control-Reg
$ffc1 con adcon1        \ AD-Control-Reg
$ffc0 con adcon2        \ AD-Control-Reg
$ffc3 con adresl        \ AD-Low-Reg
$ffc4 con adresh        \ AD-High-Reg
$ffb4 con cmcon         \ Comparator-Control-Reg

```

```
\ Interruptflags maskbits

$0020 con tmr0iem        \ Timer0 Interrupt Enable
$0004 con tmr0ifm        \ Timer0 Interrupt Flag
$0001 con tmr1ifm        \ Timer1 Interrupt Flag
$0002 con tmr3ifm        \ Timer3 Interrupt Flag
$0001 con rbifm          \ PortB Interrupt Flag
$0040 con adifm          \ ADC Interrupt Flag
$0040 con peiem          \ Pheripheral Interrupt Flag
$0010 con eeifm          \ Eeprom Interrupt Flag
$0004 con ccp1ifm        \ Capture Compare Interrupt Flag
$0001 con ccp2ifm        \ Capture Compare Interrupt Flag
$0040 con cmifm          \ Comparator Interrupt Flag
```

Und gut zum Debuggen:

```
\ tmr0ie intcon mclr
\ false di is irq ei
-nmea
marker -nmea

: .r   ( n n+ -- )   8 u.r ;

: porttest  ( -- ) base @ bin
   cr ."  trisa: " trisa c@ .r
   cr ."   lata: " lata c@ .r
   cr ."  porta: " porta c@ .r
   cr ."  trisb: " trisb c@ .r
   cr ."   latb: " latb c@ .r
   cr ."  portb: " portb c@ .r
   cr ."  trisc: " trisc c@ .r
   cr ."   latc: " latc c@ .r
   cr ."  portc: " portc c@ .r
   cr
   cr ." intcon: " intcon c@ .r
   cr ."   rcon: " rcon c@ .r
   cr ."   pir1: " pir1 c@ .r
   cr ."   pie1: " pie1 c@ .r
   cr ."   ipr1: " ipr1 c@ .r
   cr ."   pir2: " pir2 c@ .r
   cr ."   pie2: " pie2 c@ .r
   cr ." adcon0: " adcon0 c@ .r
   cr ." adcon1: " adcon1 c@ .r
   cr ." adcon2: " adcon2 c@ .r
   cr ."   tmr2: " tmr2 c@ .r
   cr ."    pr2: " pr2 c@ .r
   cr ."  t1con: " t1con c@ .r
   cr ."  t2con: " t2con c@ .r
   cr ."  t3con: " t3con c@ .r
   cr ." ccpr1l: " ccpr1l c@ .r
   cr ." ccpr1h: " ccpr1h c@ .r
   cr ." ccp1co: " ccp1con c@ .r
   cr ." ccpr2l: " ccpr2l c@ .r
   cr ." ccpr2h: " ccpr2h c@ .r
   cr ." ccp2co: " ccp2con c@ .r
   cr ." eccp1a: " eccp1as c@ .r
   cr ." pwm1co: " pwm1con c@ .r
   base !
;
```

Abbildung 1: Ur–Saxonette mit Hilfsmotor und Version des Autors (Bj. 1992)

Referenzen

[1] http://www.sourceforge.net/projects/flashforth. (kein Datum). FlashForth3.6.
[2] Microchip. (2008). 39626e.pdf.
[3] Roederer, K. (2010). Das FlashForth von Mikael Nordman. Vierte Dimension 1/2010, S. 12–18.

Listing

Und der komplette Sourcecode nochmal:

```
\ ***************************************************************************
\    Bromfiets controller for FlashForth                                     *
\    Filename:      Zuendung5.fth                                            *
\    Date:          10.05.2013                                               *
\    FF Version:    3.6                                                      *
\    Copyright:     Mikael Nordman                                           *
\    Author:        Karsten Roederer                                         *
\ ***************************************************************************
\   FlashForth is licensed according to the GNU General Public License*
\ ***************************************************************************
\ Bromfiets to go
```

```
12  \
13
14  tmr0ie intcon mclr       \ disable the tmr0 irq to prevent crash
15
16  \ false di to irq ei       \ zero the interrupt vector
17  -zuendung
18  marker -zuendung
19  decimal ram
20    #2280 constant idling         \ engine idling
21    #3750 constant maxrpm20       \ max rpm at 20 km/h
22    #0017 constant tdccorrector   \ 17° trigger before top dead center
23    #0030 constant maxdegree      \ 30° max pre-ignition
24      #32 constant prescale       \ fit with t1con !
25      #04 constant cnts           \ cnts for hand-throttel stay time
26          variable channel
27          variable allad          \ all sampels in the array discarded
28          variable igcountdown    \ ignition count down is set
29          variable period         \ actual rotation speed (pr1)
30          variable onedegree      \ counts for 1° by actual rotation speed
31                 1 onedegree !
32          variable degrees        \ degrees through pre-ignitioner
33                 1 degrees !
34          variable res1grad       \ cnts for reset to 1°
35          variable tunichts       \ cnts for doing nothing more
36          variable frueher        \ cnts for      pre-ignition
37          variable spaeter        \ cnts for retarded-ignition
38
39   maxdegree tdccorrector - constant maxdegreer
40
41   cpu_clk #1000 um* idling um/mod #60 um* prescale um/mod dup
42          constant y2 period ! 2drop \ x2 = y2 later for the math, idle
43                                \ speed
44
45   cpu_clk #1000 um* maxrpm20 um/mod #60 um* prescale um/mod
46          constant x1  2drop      \ x1 later for the math, idle
47
48   x1 maxdegree tdccorrector - um* #360 um/mod x1 swap -
49          constant y1 drop        \ y1 later for the math, idle
50
51          variable capt3          \ old capture value
52          variable earlier        \ new plugtime for compare
53          period @ earlier !      \ default at the beginning
54
55   y2 y1 - #1000 um* y2 x1 - um/mod x1 um* #1000 um/mod y1 -
56          constant z 2drop        \ often used constant during math
57
58  : ?exit         ( F1 -- )
59                  postpone  if
60                  postpone  exit
61                  postpone  then ; immediate
62
63  : case?         over = dup invert ?exit nip ;
64
65  : array create cells allot does> swap 2* + ;
66   ram #10 array sampel
67
68  : adgo   [ adcon0 1 a, bsf, ] ; \ A/D go
69  : cmpgo  [ pie1   2 a, bsf, ] ; \ enable CCP1IE interrupt
70
71  \ ADC-Initialisierung
```

```
72
73  : adinit ( -- )
74    [ porta a, clrf, ]
75    [ lata  a, clrf, ]
76    %11010010 adcon2 c!           \ right justified 32 Tosc 4TAD
77                                  \ for 20 MHz; for 8 also ok
78    %11000101 adcon1 c!           \ set AN0-AN9 as A/D pins
79    %00111111  trisa c!           \ AN0 - AN5 input (future use)
80    [ trisb 2 a, bsf, ]           \ AN8 input
81    [ trisb 3 a, bsf, ]           \ AN9 input
82    [ trisb 4 a, bcf, ]           \ RB4 output for test use
83    [ trisc 1 a, bsf, ]           \ CCP2 inp, 17° Trigger falling edge
84    [ trisc 2 a, bcf, ]           \ zuendpuls
85    [ adcon0 0 a, bsf, ]          \ A/D on
86    [ pir1   6 a, bcf, ]          \ clear A/D interrupt
87    ccp1ifm intcon mtst drop
88    0 allad !
89    0 channel !
90     ;
91
92  \ Timer-Initialiserung
93
94  : timerinit
95    [ t3con 7 a, bsf, ]           \ 16-bit operation
96    [ t3con 6 a, bcf, ]           \ timer 1 is clock source for ccp1+2
97    [ t3con 3 a, bcf, ]
98    %11111001 ccp1con c!      \ compare and falling edge of ccp1 pin
99    %11110100 ccp2con c!      \ capture every falling edge of ccp2 pin
100   [ ccpr2l a, clrf, ]       \ clear capture2 data low  reg
101   [ ccpr2h a, clrf, ]       \ clear capture2 data high reg
102   %10110101 t1con c!        \ Timer1Konfig :8 prescaler
103   [ tmr1l a, clrf, ]        \ clear Timerreg
104   [ tmr1h a, clrf, ]        \ clear Timerreg
105 ;
106
107 : irqserv ( -- ) \ interrupt service routine
108   [i
109    ccp1ifm pir1 mtst
110    if [ pir1    2 a, bcf, ]             \ clear ccp1if
111       [ ccp1con 0 a, bsf, ]             \ make him high hopefully
112    then
113    ccp2ifm 1 pir2 mtst                  \ 17° Trigger
114    if [ pir2  0 a, bcf, ]               \ clear ccp2if
115       [ portb 4 a, bsf, ]               \ set for test RB4
116       ccpr2h c@ 8 lshift ccpr2l c@ +
117       dup dup                           \ messe Periode
118       capt3 @ dup rot >
119       if swap invert 1+ + period !
120         else - period ! then
121       capt3 !                           \ store new capture
122       period @ dup 360 / dup onedegree ! degrees @ * - \ new igntcomptime
123        capt3 @ + dup $ff00 and 8 rshift ccpr1h c!       \ new compare high
124        $ff and ccpr1l c!                               \ new compare low
125       [ portb 4 a, bcf, ] -1 igcountdown ! \ clear for foregr.
126    then
127    adifm pir1 mtst
128    if
129       adresh c@ 8 lshift adresl c@ +
130       channel @ sampel !                \ store sample
131       adifm pir1 mclr
```

```
132         channel @ 1+ dup channel !        \ increment A/D-Channel
133         dup
134         1 > if drop 0 dup channel ! -1 allad ! then \ only channel 8 & 9
135         $8 + 2 lshift                     \ only channel 8 & 9
136         adcon0 c@ $c1 and or adcon0 c!
137         [ adcon0 1 a, bsf, ]      \ go
138       then
139       i]
140       ;i
141
142   : start0   ( -- )
143     timerinit
144     cr ." tmroif" cr
145     adinit
146     [ pir2 0 a, bcf, ]            \  clear CCP2IF interrupt
147     [ pir1 2 a, bcf, ]            \  clear CCP1IF interrupt
148   \  [ pie2 0 a, bsf, ]           \ enable CCP2IE interrupt führt zu Absturz
149     [’] irqserv di is irq ei ." EI" cr
150     tmr0ifm intcon mclr
151     tmr3ifm pir2 mclr
152     adgo
153   \  cmpgo Absturz
154     cr ." GO"
155    ;
156
157   : under   ( n1 n2 -- n2 n1 n2 )   swap over ;
158
159   : m/mod   ( d n -- rem quot )   dup >r
160     abs over 0< if under + swap then um/mod r@ 0<
161     if negate over if swap r@ + swap 1- then then rdrop ;
162
163   : cp   ( -- n )                   \ compare @ for test
164        ccpr1h c@ 8 lshift ccpr1l c@ + ;
165
166   : cntreset   ( -- )             \ resets the hand-throttel
167        0 tunichts ! 0 frueher ! 0 spaeter ! 0 res1grad ! ;
168
169   : decode   ( -- )
170       1 sampel @ dup .
171       dup #200 < true case? if tunichts @ 1+ dup
172                                cnts > if drop cntreset
173                             else tunichts ! then
174                                drop exit then drop
175       dup #400 < true case? if frueher @ 1+ dup
176                                cnts > if drop cntreset  1 degrees +!
177                             else frueher !  then
178                                drop exit then drop
179       dup #700 < true case? if spaeter @ 1+ dup
180                                cnts > if drop cntreset -1 degrees +!
181                             else spaeter ! then
182                                drop exit then drop
183       drop res1grad @ 1+ dup
184       cnts > if drop cntreset 1 degrees !
185              else res1grad ! then
186    ;
187
188   : start ( -- )
189     cwd start0
190      begin
191       $4000 for next              \ ms funktioniert nicht mehr
```

```
192       allad @
193       if decode 0 allad ! then    \ so wait for new ADC-inputs
194    cr
195       period  @ u.
196       capt3   @ u.
197     key?
198     until
199      cp u. ;
200
201   : tit  ( -- n )                 \ timer @
202     tmr1h c@ 8 lshift tmr1l c@ + ;
203
```

e4thcom — Terminal für 4e4th, amforth und mecrisp

Manfred Mahlow

e4thcom ist ein minimalistisches Terminalprogramm für embedded Forth-Systeme, das das Hochladen von Quelltextdateien unterstützt. Das Hochladen wird durch Terminaldirektiven angestoßen, die am Terminal eingegeben oder in Quelltextdateien eingefügt werden können.

Forth auf Mikrocontrollern

Forth auf Mlkrocontrollern hat den großen Vorteil der Interaktivität. So reicht schon ein einfaches Terminalprogramm aus, direkten Kontakt zum System aufzunehmen und seine Struktur und Funktionalität zu erkunden.

Ist man dann aber so weit, dass man erste Programme schreiben und testen kann, vermisst man die Möglichkeit, den in Dateien gespeicherten Quelltext zum Forth-System hochladen zu können. Hier kommt nun e4thcom ins Spiel.

e4thcom Terminal

e4thcom ist ein Terminalprogramm für das Linux OS, das Terminaleingaben ohne lokales Echo direkt an das Forth-System (Zielsystem) sendet und vom Zielsystem empfangene Zeichen im Terminalfenster ausgibt. Das ist der Terminalmodus des Programms.

Gibt man im Terminalmodus als erstes Zeichen einer neuen Zeile # ein, gelangt man in den Steuerungsmodus und kann Terminaldirektiven eingeben. Die eingegebenen Zeichen werden dann nicht zum Zielsystem gesendet sondern zwischengespeichert und nach Drücken der Eingabetaste vom Terminalprogramm als Direktive interpretiert.

Terminaldirektiven

Für das Hochladen von Quelltext werden in Analogie zu den Wörtern include und require des Forth Standards die Direktiven #include und #require unterstützt. Sie erwarten die Angabe der zu ladenden Quelltextdatei als einzigen Parameter:

#include <Datei> \ unbedingtes Hochladen

#require <Datei> \ bedingtes Hochladen

Beide Direktiven können auch in Quelltextdateien eingefügt werden - eine Direktive pro Zeile, der Rest der Zeile wird ignoriert - so dass beim Hochladen einer Datei automatisch weiterer Quelltext aus anderen Dateien eingefügt werden kann.

Die angegebene Quelltextdatei wird im Pfad *cwd:cwd/target:cwd/lib* gesucht. Dem liegt die Idee zugrunde, projektspezifischen Quelltext im cwd zu gespeichern, targetspezifischen Quelltext unter cwd/target und target- und projektunabhängigen Quelltext unter cwd/-lib. cwd ist das Verzeichnis, das zum Zeitpunkt des Programmstarts das aktuelle Verzeichnis (current working directory) war.

Werden die Direktiven am Terminal eingegeben, können auch die Kurzformen #i und #r verwendet werden.

Neben den Terminaldirektiven für das Hochladen von Quelltextdateien können je nach Zielsystem weitere Direktiven definiert sein. Angaben dazu finden sich im Hilfetext für das jeweilige Zielsystem.

Hochladen von Quelltext

Beim Hochladen wird der Text zeilenweise aus der Quelltextdatei gelesen und entweder als Terminaldirektive interpretiert oder als Quelltext zum Forth-System übertragen.

Nach dem Senden eines Zeichens wird auf ein Echo gewartet und dann das empfangene Zeichen im Terminalfenster ausgegeben. Bleibt das Echo aus, wird die Übertragung mit einer Fehlermeldung abgebrochen.

Nach der Übertragung einer Zeile wartet das Terminal auf die ok-Meldung des Zielsystems. Kommt diese, wird das Hochladen mit der nächsten Zeile fortgesetzt.

Kommt sie nicht, wird das Hochladen mit einer Fehlermeldung abgebrochen.

Reine Kommentarzeilen werden nicht hochgeladen sondern nur im Terminalfenster ausgegeben.

Wird das Ende einer Quelltextdatei erreicht, wird diese geschlossen und zur aufrufenden Ebene zurückgekehrt. Sind alle Dateien geschlossen, wird wieder der Terminalmodus aktiviert.

Bedingtes Hochladen

Das bedingte Hochladen von Quelltext wurde, ausgehend von der Forderung, dass alle dafür erforderlichen Informationen nicht vom Terminal sondern im Zielsystem gespeichert werden und das Zielsystem möglichst wenig belastet wird, wie folgt implementiert:

1. Vor dem Hochladen wird geprüft, ob im Wörterbuch des Zielsystems bereits ein Wort mit dem Namen der hochzuladenden Datei existiert. Ist das der Fall, wird die Direktive ignoriert. Ist das nicht der Fall, wird die Datei hochgeladen.

2. Nach dem Hochladen wird erneut geprüft, ob ein Wort mit dem Namen der Datei im Wörterbuch existiert. Ist das nicht der Fall, wird im Wörterbuch ein NOOP-Wort mit dem Namen der Datei erzeugt .

Man kann also die Zahl der ins Wörterbuch einzutragenden Dateinamen minimieren, indem man, immer dann, wenn es Sinn macht, als Dateinamen den Namen eines Wortes verwendet, das in der Datei definiert ist.

e4thcom starten

Zum Starten von e4thcom öffnet man ein Terminal im jeweiligen Projektverzeichnis und gibt dort das Kommando

/path/to/e4thcom [-i target [options]]

ein.

Folgende Argumente werden unterstützt:

-i [4e4th, amforth, mecrisp] Angabe des Zielsystems

-b [B9600, B19200, B38400, B57600, B115200] Baudrate für die Datenübertragung

-d [device] Serielles Device für die Datenübertragung, z.B. ttyS0, ttyUSB0, ttyACM0

-h Hilfetext anzeigen

Abbildung 1: e4thcom im Standardterminal

Für die Optionen sind im Programm Standardwerte voreingestellt, die durch die Angaben auf der Kommandozeile überschrieben werden können.

Unmittelbar nach dem Programmstart wird die Verbindung zum Zielsystem hergestellt. Schlägt das fehl, weil das angegebene Device nicht existiert oder noch von einem anderen Prozess (z.B. dem Modemmanager des OS) belegt ist, wird das Programm mit einer Fehlermeldung beendet.

Nach Beheben der Fehlerursache oder kurzem Warten, bis der Modemmanager aufgibt, sollte dann ein erfolgreicher Programmstart möglich sein.

Babel ...

Da Forth-Systeme bei der Datenübertragung unterschiedliches Antwortverhalten haben, ist das Programm in einen Forth-Kern (e4thcom) und mehrere Imagedateien gegliedert; je eine Imagedatei pro Zielsystem (4e4th.i, amforth.i, mecrisp.i).

With a little help ...

e4thcom bietet keine erweiterten Editiermöglichkeiten für die Kommandozeile und auch keine Kommandozeilenhistorie. Es ist bewusst einfach gehalten. Man kann es aber - ganz in UNIX-Tradition - mit einem Tool kombinieren, das die fehlenden Eigenschaften hinzufügt. Ein solches Tool ist z.B. die ForthBox.

ForthBox

Die ForthBox ist ein GTK+ GUI für Forth-Interpreter und Forth-Terminals, ein Terminalemulator, ergänzt durch Dateiauswahldialoge für das Editieren und Ausführen/Hochladen von Forth-Quelltextdateien.

In einem Einstellungsdialog kann der Anwender die zu startende Terminalanwendung, das Projektverzeichnis, den zu verwendenden Editor und die Parameter für eine serielle Verbindung auswählen.

Um e4thcom in einer ForthBox zu starten, ist der Befehl

/path/to/e4thcom -i forthbox

einzugeben.

Unmittelbar nach dem Programmstart wird der Einstellungsdialog geöffnet. Nach Auswahl der gewünschten Optionen und Drücken des OK-Knopfes wird e4thcom für das ausgewählte Zielsystem im Terminalfenster gestartet. Schlägt das aus den oben genannten Gründen fehl, kann man, gegebenenfalls nach Anpassen der Einstellungen, das Terminalprogramm erneut starten, wie nachfolgend beschrieben.

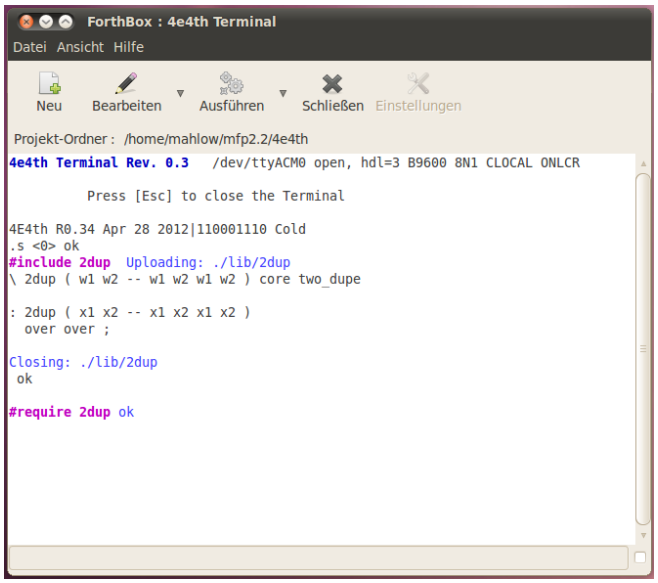


Abbildung 2: e4thcom in der ForthBox

Zur Verwendung der ForthBox hier nur noch ein paar kurze Hinweise:

- Über die Werkzeugleiste kann man mit den Knöpfen *[Bearbeiten]* bzw. *[Ausführen]* eine Datei zum Bearbeiten bzw. zum Ausführen/Hochladen auswählen.
- Neben den Knöpfen *[Bearbeiten]* und *[Ausführen]* gibt es jeweils einen Menü-Knopf, über den man auf die vom OS geführte globale Liste der zuletzt benutzten Dateien zugreifen kann. Hier erscheinen auch alle Dateien, die zum Bearbeiten ausgewählt werden.
- Eine zum Bearbeiten ausgewählte Datei wird nach dem Schließen des Auswahldialogs an den externen Editor übergeben, der im Einstellungsdialog ausgewählt wurde.
- Über den ersten Eintrag in der Liste zum Knopf Ausführen kann man den jeweiligen Terminal-Client neu starten.
- Unter dem Terminalfenster kann eine separate Kommandozeile mit automatischer Textauswahl (Historie) und automatischer Textvervollständigung aktiviert werden (Check-Button rechts unten).

Weitere Informationen zur ForthBox bleiben einem separaten Artikel vorbehalten.

Anmerkungen:

e4thcom ist freie Software gemäß der GNU General Public License Version 3. Es ist in MINFORTH Plus implementiert, einem MINFORTH Derivat und wurde mit folgenden Zielsystemen getestet:

- Ardunino Duemilanove: amforth 4.9 und 5.1
- TI MSP430 Launchpad: mecrisp 1.0, 4e4th 0.34.

Mit Veröffentlichung dieses Artikels werde ich das Programm als e4thcom-x.y.tar.gz Archiv über die WEB-Seite der Forth-Gesellschaft www.forth-ev.de zur Verfügung stellen.

Fragen, Fehlerhinweise und Verbesserungsvorschläge zum Programm, auch Vorschläge für weitere Zielsysteme, bitte an folgende E-Mailadresse senden: manfred.mahlow@forth-ev.de .

Matrix–Tastaturscanner

Rafael Deliano

Häufig benötigt, eigentlich nicht komplex, aber Details wollen berücksichtigt werden.

Belegt typisch einen kompletten 8–Bit–Port des Controllers. Bis 8 Pins, also typisch 4x4 Tasten, kommt man ohne weiteres IC aus (Bild 1). Eine Tastatur mit 5x5 Tasten (Bild 2, 3) benötigt ein Schieberegister als Erweiterung. Die Widerstands-Arrays sind hier zwar eingezeichnet. Sie können im Layout aber oft entfallen, da der GP32 Controller intern zuschaltbare pullups hat.

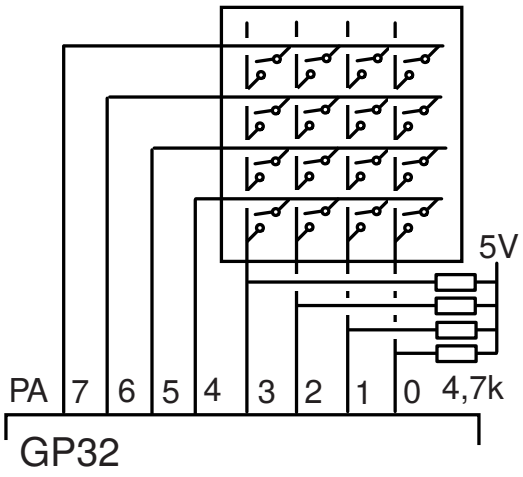


Abbildung 1: kleine Tastatur

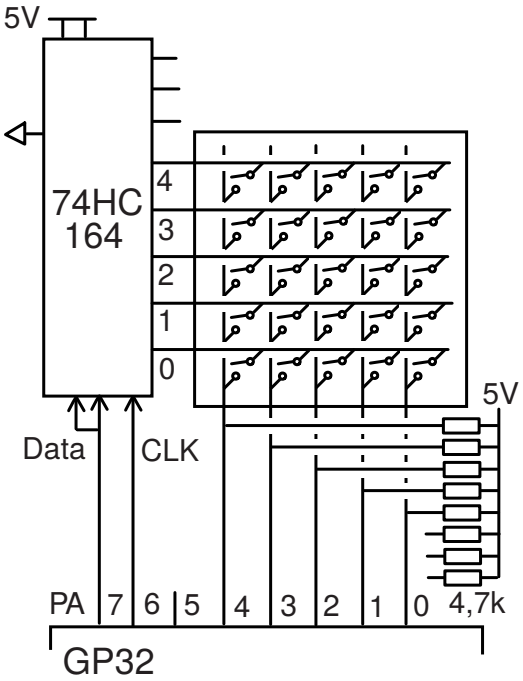


Abbildung 2: größere Tastatur

Abbildung 3: 5x5–Tastatur

Decodierung

Die Zeilen der Tastatur werden von Output-Pins angesteuert, die im Ruhezustand alle auf 00000 geschaltet sind. Die Spalten sind Inputs, die dann durch die pullups den Pegel 11111 haben. Man kann also Aktivität an der Tastatur direkt erkennen, wenn der gelesene Wert der Eingänge von 11111 abweicht. Decodierung der Taste erfolgt dann durch eine Routine GET-KEY die sequentiell die Muster 11110, 11101, 11011, 10111, 01111 ins Schieberegister (Bild 2) schreibt und anhand der Werte am Eingang die betätigte Taste bestimmt. Gültige Werte sind auch dort wieder 11110, 11101, 11011, 10111, 01111. Anhand der Positionsinformation der Taste in der Matrix holt man dann aus einer etwa 50 Byte langen Tabelle den eigentlichen Tastaturcode, meist ein 7 Bit ASCII-Zeichen. GET-KEY erkennt auch ghosting, also den Zustand, in dem mehrere Tasten betätigt sind, und liefert dann den Fehlercode 00 zurück.

Ausgabe

Auch wenn die meisten Taster nur maximal 10msec Prellzeit haben, sollte man bis zu 40msec tolerieren können. Zu lange darf die Entprellzeit aber auch nicht sein. Bei rascher Eingabe auf Computertastaturen dauert die Pause zwischen Tastendruck typisch nur 120msec [1] (Bild 7). Zudem will man eine unverzögerte Reaktion des Geräts auf Tastendruck. Die Timerfunktion sollte also passend konfigurierbar sein.

Meist lässt man die Tastaturroutine mit dem System-Tick des Geräts mitlaufen. Hier wird sie alle 0,5msec aufgerufen und mit LIMIT = 80 , d.h. 128 Ticks, ist der Timer CNTR auf 64msec eingestellt (Bild 4).

Es gibt zwar eine Variable SYM, die den ASCII-Code der betätigten Taste enthält. Aber öfters benötigt man in der Variable SEMA an den Flanken einen key-on- und key-off-Token. ON Ist hier der ASCII-Code mit Bit 7 = 0, OFF mit Bit 7 = 1. Die Zustandsmaschine (Bild 6) sollte auch auf den Fall getestet werden, dass Tasten ohne Pause ineinander übergehen (Bild 5). Bzw. dass auch bei ghosting eine plausible key-on/key-off-Sequenz erzeugt wird.

Auf einem GP32 in Assembler ist der Speicherverbrauch inklusive Tabelle ca. 600 Byte. Bei 2,45MHz CPU-Takt benötigt die Routine 15usec, wenn keine Taste betätigt ist. Bzw. 280usec, wenn Taste gedrückt.

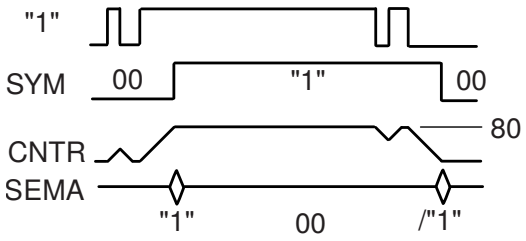


Abbildung 4: Signale in Variablen

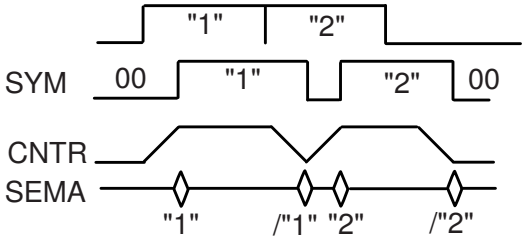


Abbildung 5: Betätigung ohne Pause

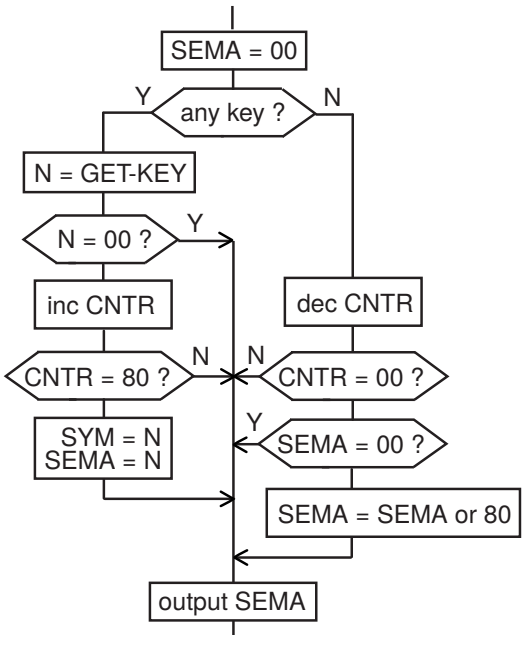


Abbildung 6: Flussdiagramm

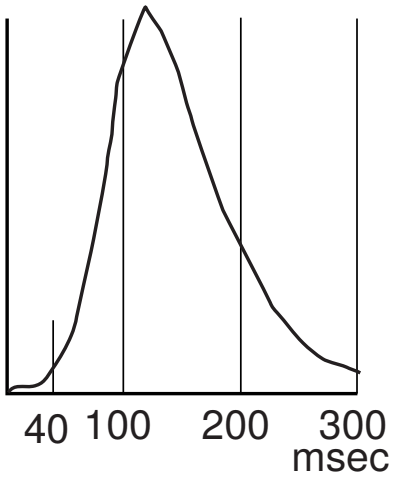


Abbildung 7: Häufigkeit der Pausen zwischen Tastendruck auf Computertastatur

Test-Modus

Hier wurde nur eine gedrückte Taste als gültige Eingabe zugelassen. In manchen Geräten sind undokumentiert Test- und Diagnosefunktionen realisiert, die durch gleichzeitige Betätigung von zwei Tasten ausgelöst werden. Da nicht sicher auszuschließen ist, dass der Benutzer in diese Betriebsarten stolpert, wird das hier nicht empfohlen. Kompromiss wäre, im power-up die Tastatur auf solche Doppeltasten abzufragen.

Referenzen

[1] Sherr Input Devices, Academic Press 1988

Listing

```
1   <| \ SRC0
2
3   \ keyboard decoder
4
5   \ port PA pin 0 ... 4
6   PA 5 DCONSTANT TO    \ out 0
7   PA 6 DCONSTANT CLK   \ out 0
8   PA 7 DCONSTANT DATA  \ out 0
9
10  :CODE CLK     \ puls CLK-pin high
11  CLK MBS, NOP, NOP, NOP,
12  CLK MBC, RTS, CODE;
13  :CODE 4xCLK   \ puls CLK-pin 4x high
14  ' CLK JSR, ' CLK JSR,
15  ' CLK JSR, ' CLK JMP, CODE;
16  :CODE SR=00000   \ clear shift register
17  DATA MBC,  ' CLK JSR,
18  ' 4xCLK JMP, CODE;
19  :CODE SR=11110   \ load shift register
20  DATA MBS,  ' 4xCLK JSR,
21  DATA MBC, ' CLK JMP, CODE;
22  :CODE SR=<<0     \ shift shift register
23  DATA MBS,  ' CLK JMP, CODE;
24
25  : INIT-KB       \ (  --  ) setup ports
26  B% 11100000 DPA C!
27  B% 00000000 PA C!  SR=00000 ;
28
29  TABLE CHARS \ 00 = invalid
30  00 C, 21 C, 22 C, 23 C, 24 C, 25 C, 00 C,
31  00 C, 00 C, 31 C, 32 C, 33 C, 34 C, 35 C,
```

```
00 C, 00 C, 00 C, 41 C, 42 C, 43 C, 44 C,
45 C, 00 C, 00 C, 00 C, 51 C, 52 C, 53 C,
54 C, 55 C, 00 C, 00 C, 00 C, 61 C, 62 C,
63 C, 64 C, 65 C, 00 C, 00 C, 00 C, 71 C,
72 C, 73 C, 74 C, 75 C, 00 C, 00 C,

:CODE GET-CLMN    \ out N 1+: 1,2,3,4,5 ;
XSAVE           SHX,  \ save stackpointer
              X. CLR,
    PA          LDA,
    B% 11111 #. AND,
                INX,
    B% 11110 #. CMP,
1 $             BEQ,
                INX,
    B% 11101 #. CMP,
1 $             BEQ,
                INX,
    B% 11011 #. CMP,
1 $             BEQ,
                INX,
    B% 10111 #. CMP,
1 $             BEQ,
                INX,
    B% 01111 #. CMP,
1 $             BEQ,
              X. CLR,
1 $:            TXA,
XSAVE           LHX,
              A. TST,
2 $             BEQ,  \ skip if ghosting
     N          ORA,  \ insert column code
     N 1+       STA,  \ store as output
     N 2+       INC,  \ inc CNTR
2 $:            RTS,
CODE;

:CODE GET-KEY         \ out accu: ASCII ; 00 = invalid
     N 1+      CLR,   \ output register
     N 2+      CLR,   \ CNTR
' SR=11110 JSR,  N B% 000000 #. MOV, \ SR=11110
' GET-CLMN JSR,
' SR=<<0 JSR,    N B% 001000 #. MOV, \ SR=11101
' GET-CLMN JSR,
' SR=<<0 JSR,    N B% 010000 #. MOV, \ SR=11011
' GET-CLMN JSR,
' SR=<<0 JSR,    N B% 011000 #. MOV, \ SR=10111
' GET-CLMN JSR,
' SR=<<0 JSR,    N B% 100000 #. MOV, \ SR=01111
' GET-CLMN JSR,
' SR=00000 JSR,
     N 2+        DEC, \ CNTR to 00 if it was 01
1 $             BEQ,
     N 1+        CLR, \ output = 00 if ghosting
1 $: N 1+       LDA,
XSAVE           SHX,
                TAX, H. CLR,
CHARS         ,X LDA,
XSAVE           LHX,
                RTS,
CODE;

3 ZVARIABLE KY
\ 0 + SYM
\ 1 + CNTR
\ 2 + SEMA     00 =idle/invalid

:CODE KB?   \ ( -- UC1 ) UC1 = 00000000 idle
            \            UC1 = 0bbbbbbb on
            \            UC1 = 1bbbbbbb off
    KY 2 +      CLR,    \ sema = idle
    PA          LDA,    \ test if keyboard activity
    B% 11111 #. AND,
    B% 11111 #. CMP,
1 $             BEQ,
' GET-KEY       JSR,
    N           STA,
1 $             BEQ,    \ branch if ghosting


2 $   KY 1+  7  BBS,    \ inc CNTR if not LIMIT 80h
      KY 1+     INC,
2 $   KY 1+  7  BBC,    \ if CNTR reaches LIMIT output
                        \ key-on sema
    KY    N     MOV,    \ load SYM
    KY 2+ N     MOV,    \ load SEMA
2 $             BRA,

1 $:                    \ idle
      KY 1+     LDA,    \ dec CNTR if not 00
2 $             BEQ,
      KY 1+     DEC,
      KY 1+     LDA,    \ if CNTR reaches 00 output
                        \ key-off sema
2 $             BNE,
      KY        LDA,
2 $             BEQ,    \ skip if 00 : no code 80
          80 #. ORA,
      KY 2+     STA,
      KY        CLR,    \ clear SYM

2 $:                    \ output sema
      KY 2+     LDA,
                DEX, 0,X STA,
        A. CLR, DEX, 0,X STA,
                RTS,
CODE;

\ -------------------------------------------------

:CODE GET-KEY"          \ (  -- UC1 )
' GET-KEY       JSR,    \ get key
                DEX, 0,X STA,
        A. CLR, DEX, 0,X STA,
                RTS,
CODE;

: TEST1       \ (  --  ) scan and print every second
INIT-KB BEGIN CR GET-KEY" CH. D% 1000 MSEC AGAIN ;

: TEST2       \ (  --  ) print key-on and key-off
INIT-KB
BEGIN KB? DUP
IF DUP 80 AND LNOT IF CR THEN CH. ELSE DROP THEN
1 MSEC AGAIN ;

CR ." -> SRC1"
CR |>
```

Atmendes LED

M. Kalus

Am Beispiel des MSP430G2553–LaunchPads wird gezeigt, was man alles so falsch machen kann auf dem Wege hin zu einem „atmenden" LED. Ein eigentümliches Flackern im stetigen An– und Abschwellen der Helligkeit der LED brachte uns dann auf die richtige Lösung dieser Steuerungsaufgabe. Der Code dafür wird vorgestellt, in Assembler, um zu zeigen, wie wenig wirklich nötg ist für die atmende LED. Der Aufbau ist schnell beschrieben. Es braucht nur das LaunchPad am PC, sonst nichts. An Software braucht es natürlich den passenden Assembler. Bei diesem Assembler-Projekt wurde die IDE von IAR benutzt — Plattform: Windows XP.

Aufgabenstellung

Entstanden ist das atmende LED in einer Reihe kleinerer Schulungsprojekte. Der Student sollte einen RISC–Prozessor in einer MCU näher kennen lernen. Nachdem der Instruktionssatz und dann der Instruktionsaufbau erarbeitet worden waren, galt es zu ermitteln, wie die CPU mit der auf dem Chip integrierten Peripherie zusammenwirkt. Das Konzept, solche integrierten Funktionsgruppen über Register I/O und Kontroll-Register zu steuern, sollte erkundet werden. In einer weiteren Serie soll dann der Übergang zum Bau eines Compilers (Forth) bewältigt werden.

Bei der Suche nach einer geeigneten Hardware fiel die Wahl auf das LaunchPad, weil es klein, handlich, billig und — vor allem — gerade verfügbar war, und zudem über USB einfach am PC angeschlossen werden konnte. Motiviert durch diese Schulungsaufgabe entstand eine kleine Assembler–Projektreihe für das LaunchPad. Bedingung für alle Projekte dieser Reihe war es, das LaunchPad zu benutzen „wie es ist". Also ohne weitere Hardware daran anschließen zu müssen. Lediglich beim Projekt 7 wurde ein Potientiometer (Schiebesteller) hinzugenommen. Daraus entstanden folgende Übungen:

Projekt 0	Blinkendes LED
Projekt 1	Blinkendes LED II
Projekt 2	Tasterstellung anzeigen. (I/O Übung)
Projekt 3	Taster und Flanken-Interrupt
Projekt 4	Terminal und LEDs
Projekt 5	PWM an Pin P1.6 (grüne LED)
Projekt 6, 6a	Atmendes LED
Projekt 7	Dimmer (mit Potentiometer und LED)
Projekt 8	Ausgabe einer WAV-Datei

Wer sich dafür interessiert, findet diese im unten angegebenen Link.

Vorgabe

Das LauchPad hat drei LEDs auf dem Board, zwei davon ansteuerbar, eine rote an Port-Pin P1.2 und eine grüne an P1.6. Die dritte ist lediglich Anzeige für die Versorgungsspannung. Die Vorgabe für das atmende LED war so: Ein LED in der Helligkeit lansam an- und abschwellen lassen, ungefähr in der menschlichen Atemfrequenz von etwa 10 Atemzügen pro Minute. So dass ein vollständiger Zyklus rund 6 Sekunden dauert. An- und Abschwellen sollte einem sinusförmigen Verlauf folgen. Übungshalber wurden drei Dateien mit Sinus-Werten zwischen 0°-360° erzeugt, je eine Tabelle mit Werten in 1°, 5° und 10°-Schritten. Diese Tabellen wurden einmal in Excel, und dann auch mit Phython erzeugt, in der Notation für den Assembler ausgegeben, und so als Datei eingebunden.

Vorbereitung

Zum Programmablauf wurden zunächst Ideen gesammelt, Fragen geklärt, und sodann ein Flussdiagramm erstellt (Abb. 1). Das beschrieb den Ablauf so: Ein Taktgenerator erzeugt ein schnelles periodisches PWM-Signal, mit dem ein bestimmter Helligkeitswert am LED erreicht, und so schnell wiederholt wird, dass kein Flimmern mehr zu sehen ist (Bildwiederholrate); dazu wurden mindestens 24 Hz (Kino), besser 50Hz bis 70Hz (wie flimmerfreie Monitore) angesetzt. Dieser TA0 genannte Taktgenerator löst Interrupts aus, und die ISR (engl.: iterrupt service routine) setzt den Helligkeitswert jedesmal neu. Und ein anderer Taktgenerator TA1 wechselt den Helligkeitswert so, dass die Wertetabelle ca. alle 6s einmal durchlaufen wird. Dessen ISR holt den nächsten Helligkeitswert aus der Sinustabelle. Nach der Initialisierung hat die main-Routine somit keine weitere Aufgabe, die eigentliche Arbeit erledigen die beiden ISR. Hier sei schon verraten, dass wir es versäumt hatten, die Schnittstelle zwischen den beiden ISR zu diskutieren, und festzulegen wie die Werte übergeben werden sollen.

Programmierung — und zwei typische Fehler

Dann ging es ans Werk. Der Blick in das Handbuch der MCU zeigte, dass an P1.6 statt des einfachen I/O auch ein PWM–Signal ausgegeben werden kann (2nd function), erzeugt vom Timer–A0 der MCU. So nahmen wir für TA0 den Timer-A0. Der andere Takgeber brauchte keine Verbindung zur Außenwelt. TA1 soll ja lediglich den Trigger liefern, mit dem die Werte aus der Sinus–Tabelle periodisch an den PWM–Generator TA0 übertragen werden. Nach den Übungen aus den vorherigen Projekten war bald klar, was in den Timer–Registern initial eingestellt werden musste, das Listing zeigt, wie das geht.

Doch obschon der Ablauf der Interrupts funktionierte, schimmerte das grüne LED nur schwach. Dass die Interrupts funktionierten, wurde folgendermaßen überprüft.

Durch die zusätzliche Instruktion `XOR.B #red,&P1` wurde die rote LED getoggelt, also immer, wenn der Prozessor dort „vorbei“ kam, wechselte die rote LED ihren Zustand zwischen *an* und *aus*. Um das mit dem Auge verfolgen zu können, wurde der Prozess stark verlangsamt. Der Prozessortakt wurde auf 1MHz herabgesetzt, und die Taktteiler der Timer wurde maximal gesetzt (prescaler 1/8). Das führte dazu, dass man dem *An* und *Aus* der LEDs schon folgen konnte, und sah was passierte. Indikator war jeweils die rote LED, siehe Listing. Also die Interrupts arbeiteten korrekt.

Was wir übersehen hatten, war, dass unsere Sinustabelle mit einer Auflösung von 8Bit generiert worden war, also Werte zwischen 0 — 255 hatte. Doch das Timer–Register hat 16bit–Breite, und die Wiederholung der Periode geschah langsam, nach $FFF0 Takten. Das hatte zur Folge, dass immer nur ein 8Bit schmales SET, und dann ein um so breiteres RESET des PWM erfolgte. Was natürlich die LED nur so gerade eben schimmern ließ. Immerhin, im Prinzip funktionierte es schon mal. So wurden die Tabellen neu skaliert auf Werte 0 — $FFFF. Nun atmete die LED schon sichtlich. Sichtlich war aber auch, dass am hellsten Punkt ein kurzes Verlöschen zu sehen war. Der Tabellenwert $FFFF, übertragen ins Comparator-Register TACCR1, war größer als der Timer-Reset-Wert im Comparator-Register TACCR0, was dazu führte, dass es dann gar nicht zum SET kam, die LED also einige Perioden lang ausgeschaltet blieb, was ein sichtbares Erlöschen zur Folge hatte. Die Tabellen wurde erneut angepasst, jetzt mit Maximalwerten passend zum Wert der PWM-Periode, so wie im Listing notiert. Damit war dieses Phänomen behoben. Nun lief eigentlich alles sehr schön glatt, das LED atmete.

Doch ab und zu funkelte es unstet. Zuerst sah das wie zufällig aus. Das LED ging manchmal für einen kurzen Augenblick einfach aus. Nach längerer Betrachtung fanden wir heraus, dass dahinter eine feste Sequenz lag. Diese Regelmäßigkeit blieb aber zunächst rätselhaft. Nachdem das Internet befragt worden war (siehe Link), hatten wir endlich einen Verdacht, woran es liegen könnte. Das Phänomen hatten wir unter uns dann „Rückfall“ genannt. Der Timer zählt ja kontinuierlich hoch bis zu seinem Umkehrpunkt, definiert durch den Wert im Register TACCR0, springt da wieder auf null, und setzt dabei den Pegel an P1.6 *high* — `SET`. Unterwegs erreicht der Zähler den Wert im Register TACCR1, wodurch der Pegel an P1.6 wieder auf *low* gesetzt wird — `RESET`. Wenn nun der Wert im TACCR1 absichtlich hinter den Zählerstand TAR des Timers zurück gesetzt wird, gibt es in dieser Periode kein `RESET` mehr. In den Skizzen wird dies veranschaulicht. Diese „Rückfall-Lücke“ war tatsächlich sichtbar für das menschliche Auge, glücklicherweise, jedenfalls bei der Wiederholrate die wir eingestellt hatten. Sonst wäre uns dieses Phänomen verborgen geblieben.

So ein „Rückfall“ passierte also immer dann, wenn durch die Phasenverschiebung zwischen der Wiederholrate und der PWM-Frequenz das Register TACCR1 gerade dann mit einem kleineren Wert neu beschrieben wurde, wenn TAR kurz davor war, den vorherigen Wert von TACCR1 zu erreichen. Die Register vom Timer-A haben keinen Puffer. Schreibt man dort einen neuen Wert hinein, wird er **sofort** wirksam, und es kommt dann ab und zu zu diesem Flackern des LED. Indem wir ein Puffer-Register dazu nahmen, verschwand das Phänomen. So übernahm nun die PWM-Routine ihren Stellwert immer am Ende ihrer Periode aus diesem Puffer. Und die Routine für die Wiederholrate schrieb dort hin (s. Listing).

So entstand schließlich ein schön gleichmäßig atmendes LED. Die Listings im Link beinhalten auch die Fehler, die wir gemacht haben, und auch Testhilfen, allerdings auskommentiert. Die nicht auskommentierten Zeilen enthalten die Lösungen. Viel Spaß beim Ausprobieren der Phänomene.

Skizzen

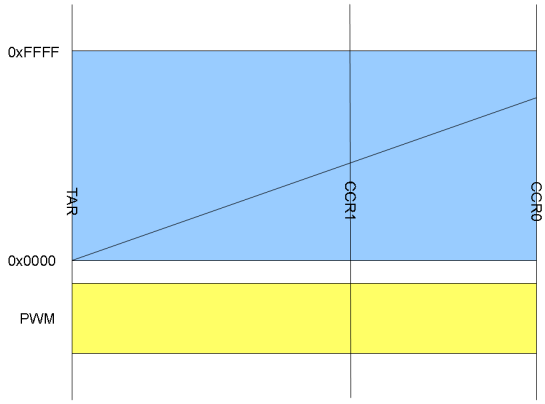


Abbildung 1: Der Timer A (blau) startet einen PWM-Zyklus. Timer–A–Register TAR=0.

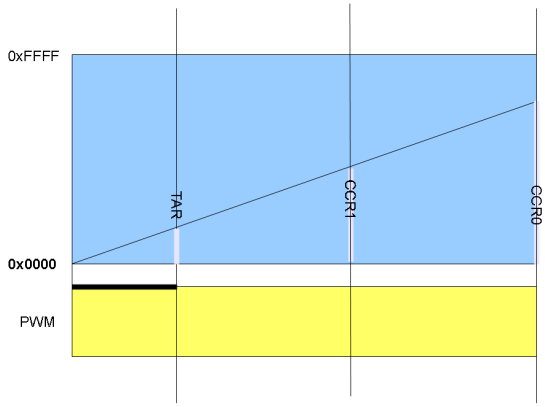


Abbildung 2: TAR zählt hoch. Die Spannung am PWM-Pin ist Vcc=SET.

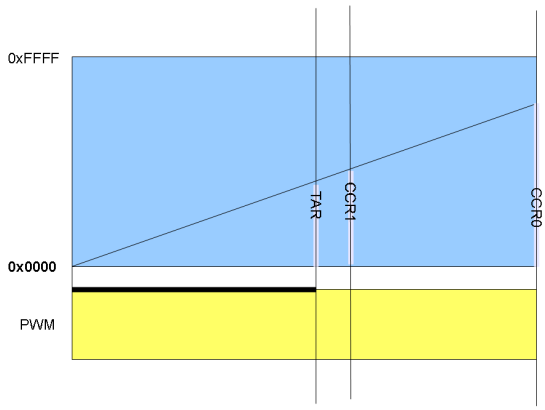


Abbildung 3: Der Wert in TAR nähert sich dem Wert im Timer–A–Register CCR1.

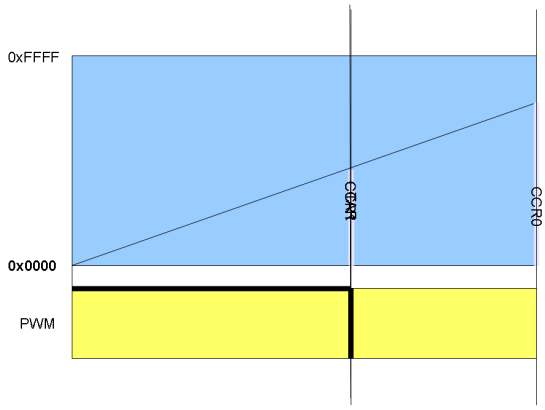


Abbildung 4: TAR=TACCR1. Der Pegel am PWM-Pin springt auf Vss=RESET

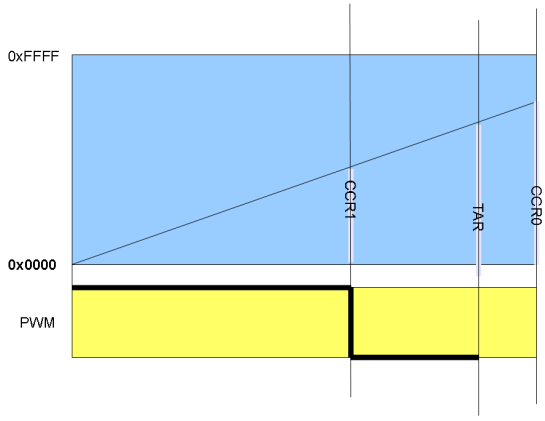


Abbildung 5: TAR zählt weiter hoch.

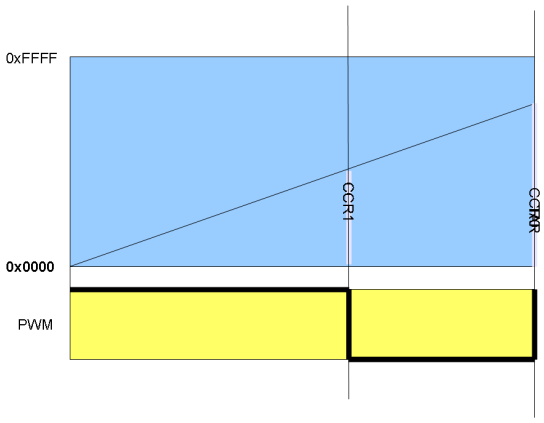


Abbildung 6: TAR=TACCR0. Es erfolgt ein SET des PWM-Signals, sowie TAR=0 (Im upmode setzt der Zähler hier TAR auf NULL zurück.)

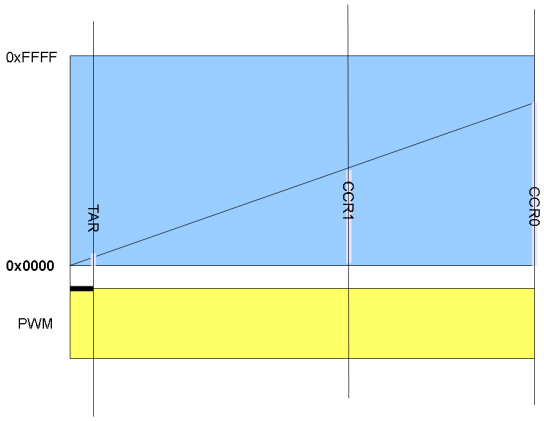


Abbildung 7: TAR zählt erneut hoch. PWM == SET.

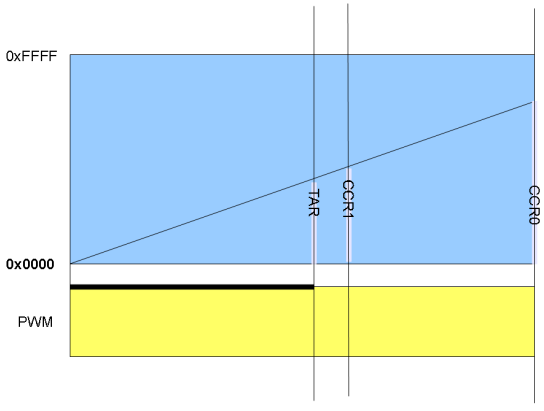


Abbildung 8: Der Wert in TAR nähert sich wieder dem Wert in CCR1...

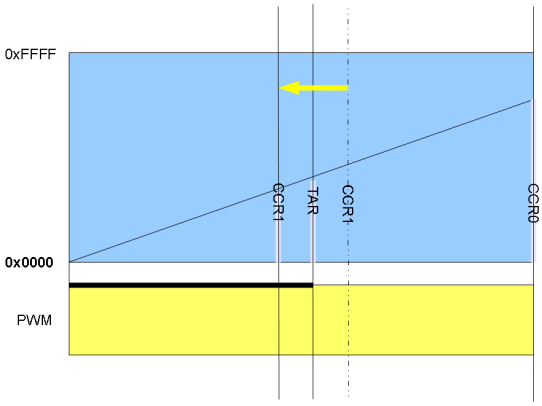


Abbildung 9: ... und in dem Moment wird der Wert in CCR1 von der anderen ISR herunter gesetzt, tiefer als der Wert, den TAR schon erreicht hatte. ...

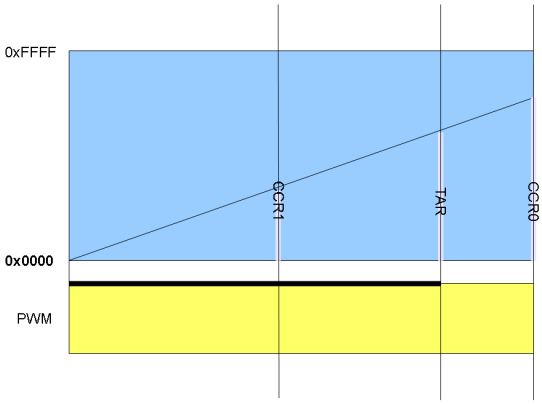


Abbildung 10: ... TAR zählt weiter hoch, es kommt aber nicht mehr zu einem Gleichstand mit CCR1. Der Zustand SET des PWM bleibt daher erhalten. Es fällt daher die RESET-Phase aus.

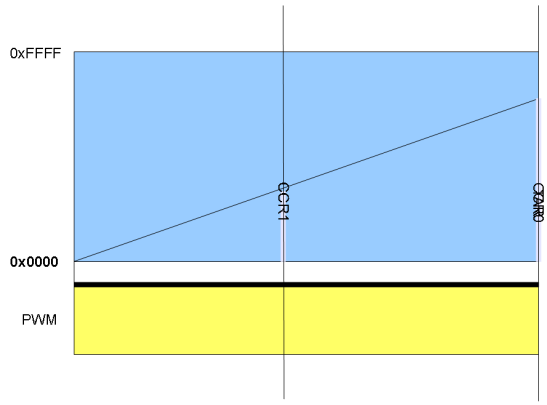


Abbildung 11: Bei TAR=CCR0 wird ebenfalls ein SET des PWM gemacht, PWM bleibt also weiterhin im Zustand SET. TAR=0, ein neuer Zyklus beginnt.

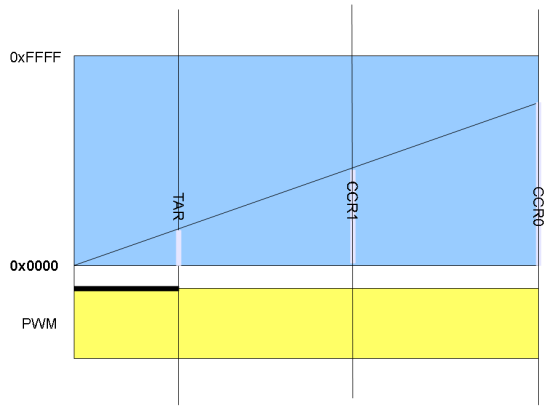


Abbildung 12: TAR zählt wieder hoch, erreicht den neuen CCR1-Wert, RESET des PWM usw. Es ist also eine Periode ausgefallen, weil Schreiben eines neuen Wertes nach CCR1 sofort wirksam wird. Die Pufferung muss also außerhalb des Zählers in Software gemacht werden.

Links

Die kleinen Projekte sind hier zusammengefasst:
https://dl.dropboxusercontent.com/u/1170761/8_projekte.zip

Den rettenden Beitrag fanden wir hier:
„PWM using TimerB in Continuous mode“; in: msp430@yahoogroups.com

Listing

```
1   ; projekt 6a - Atmendes LED II
2   #include "msp430.h"  ; #define controlled include file
3
4   ;             76543210
5   #define gruen 01000000b  /* Maske für gruenes LED */
6   #define rot   00000001b  /* Maske für rotes LED   */
7   #define Prozessortakt CALBC1_8MHZ
8   #define pwmPeriode  0xFFFE /* Dauer der PWM--Periode */
9   #define bildperiode 0xA2C2 /* Dauer für einen Schritt aus der Sinustabelle. */
10
```

```
11          NAME    main                    ; module name
12          PUBLIC  main,isr_TA1,isr_TA0,init,isr_null
13          EXTERN  sinStart,sinEnd
14          RSEG    CSTACK                  ; pre-declaration of segment
15          RSEG    CODE                    ; place program in 'CODE' segment
16  init:   MOV.W   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
17          MOV     #SFE(CSTACK), SP        ; set up stack
18          MOV.B   &Prozessortakt, &BCSCTL1   ; Set DCO = Prozessortakt
19          MOV.B   &Prozessortakt, &DCOCTL
20          MOV.B   #(gruen+rot),&P1DIR     ; p1.0 und p1.6 auf output setzen
21          BIS.B   #gruen,&P1OUT           ; gruene LED anmachen
22          BIC.B   #rot,&P1OUT             ; rote LED ausmachen
23          BIS.B   #gruen,&P1SEL           ; second funktion von P1.6 = TA0.1
24          MOV     #sinStart,R4            ; Tabellenanfang nach R4 schreiben
25          MOV     @R4,R5                  ; Buffer R5 mit Startwert laden
26          ;TA0 init
27          BIS     #CCIE,&TA0CCTL0            ; Interrupt mode aktivieren
28          BIS     #OUTMOD_7,&TA0CCTL1        ; set/reset mode anmachen
29          MOV     #pwmPeriode,&TA0CCR0       ; set 122/s
30          MOV     R5,&TA0CCR1 ; set initialen Wert = tebellen anfang
31          BIS     #(TASSEL_2+MC_1),&TA0CTL   ; use smclk fuer TA0, upMode
32          ;TA1 init
33          BIS     #CCIE,&TA1CCTL0                ; Interrupt mode
34          MOV     #bildperiode,&TA1CCR0          ; set bildperiode
35          BIS     #(TASSEL_2+MC_1+ID_3),&TA1CTL   ; smclk, upMode, /8
36
37          BIS.B   #GIE,SR  ; Global Interrupt an
38  main:   JMP main  ; (endless loop)
39
40  isr_null: RETI
41  isr_TA0:  MOV R5,&TA0CCR1             ; helligkeitswert aendern
42            RETI
43  isr_TA1:  MOV @R4+,R5                 ; Tabellenwert in Buffer R5 schreiben
44            CMP #sinEnd,R4
45            JN  isr_TA1_end
46            MOV #sinStart,R4
47  isr_TA1_end: RETI
48          END
49
50  (Für den Druck wurde die Sinustabelle gekürzt. Bitte Projekt herunterladen.)
51  PUBLIC sinStart,sinEnd
52  RSEG    CODE
53  ;Sinus--Tabelle in 5--Grad--Schritten
54  sinStart:
55          DW 32767        ; 0 Grad
56          DW 35623        ; 5 Grad
57          DW 38457        ; 10 Grad
58          DW 41248        ; 15 Grad
59              ...
60  sinEnd:
61      END
```

Wave Engine (8)

Hannes Teich

Die im letzten Beitrag besprochene Programmversion war im Stande, vier abklingende Akkorde ertönen zu lassen, die allerdings von unerwünschten Nebengeräuschen begleitet waren. Dieses Rauschen zu beseitigen, war nicht ganz trivial, und davon soll heute berichtet werden.

Die Fehlersuche

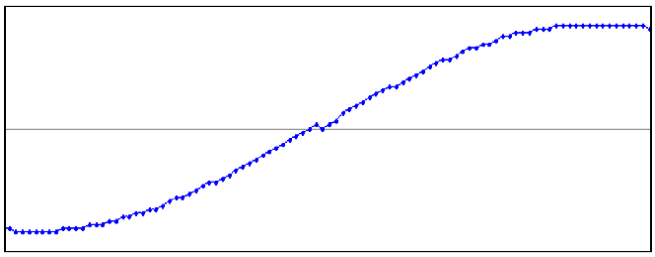


Abbildung 1: Das leidige Rauschen, mit Audacity sichtbar gemacht

So wie in **Abb. 1** sah es vor der Reparatur aus. Der erste Blick lässt den Verdacht aufkommen, es gehe hier um floored/symmetric division, aber so etwas ist nicht beteiligt. Auch sonst ist die Kurve nicht eben „smooth“, und das ist halt deutlich zu hören. Um die Ursache zu finden, ist ein tieferer Blick ins Geschehen nötig.

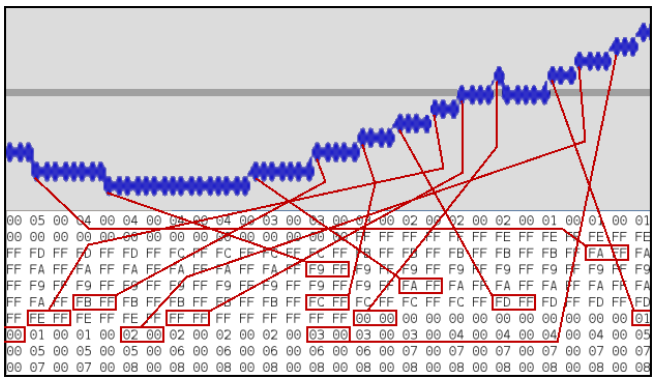


Abbildung 2: Protokoll einer Fehlinterpretation

Abb. 2 zeigt die durch das Programm berechneten Werte rund um einen Nulldurchgang, sowie deren Auswirkungen, mit der Anwendung *Audacity* sichtbar gemacht. Der Übergang von FF–FF zu 00–00 scheint die Doppelbytegrenzen anzuzeigen. Doch das stellte sich als Irrtum heraus: Die Kästchen sind allesamt um ein Byte nach rechts verrutscht. Der Ausreißer nach oben wird durch die Folge $FFFF – $00FF – $0000 ausgelöst (in Dezimalzahlen -1 – 255 – 0), und damit verhält sich der Ausreißer korrekt. Warum allerdings $FFFF und $0000 (zumindest bei Audacity) auf gleicher Höhe landen, ist ungeklärt, aber auch ziemlich unwichtig. Denn wie man sieht, ist durch die Verwechslung der Bytes die ganze Angelegenheit um den Faktor 256 vergröbert worden.

Es lag also nahe, die Funktion *swap–bytes* ins Spiel zu bringen. Das allerdings führte auch nicht zum gewünschten Verhalten, es gab immer noch Nebengeräusche. Des Rätsels Lösung war schließlich: Irgendwo vorher war ein Byte zu viel eingefügt, so dass alle Werte um ein Byte nach rechts verschoben waren — und damit hohe und tiefe Bytes vertauscht erschienen.

Im Nachhinein muss ich mir sagen, Abb. 1 hätte mich auf die Verwechslung / Verschiebung schon aufmerksam machen sollen. Die grobe Darstellung war mir zwar aufgefallen, aber ich hielt das für ein weiteres Problemchen, das ich anschließend anzugehen gedachte.

Weitere Aspekte

Die vier (jetzt sechzehn) Akkorde klingen jetzt ausreichend sauber. Es handelt sich schließlich um eine temperierte Stimmung, die immer etwas rauer klingt als eine reine, vor allem, wenn es sich um eine sterile Tonerzeugung handelt. Den Tönen tut ihre Oberquinte (2. Oberton) gut. Ohne sie klingt es härter, wie der geneigte Leser (m/w) selbst feststellen kann, wenn er (Gforth und Wave–Engine–Ver3–03 geladen habend) die Obertöne mal probeweise abschaltet. Dazu wären in Datei `singen-2.fs` (ziemlich unten) die Lautstärkewerte der vier Obertöne von 1000 auf 0 zu setzen.

Der letzte Akkord wollte nicht ausklingen, er wurde vorzeitig beendet. Der von der Partitur gelieferte Zweibyte-Wert für die Tonlänge kann die Anzahl benötigter Frames nicht fassen. Der Maximalwert 65536 ergibt mit 44100 Frames pro Sekunde nicht die ausreichende Zeit. Deshalb wird für den aktuellen Fall in Funktion `loops` mit `2*` ein 17–bit–Wert erzeugt.

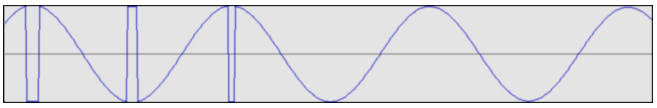


Abbildung 3: Übersteuerung eines abklingenden Tons

Abb. 3 zeigt, was bei Übersteuerung geschieht. Mit den Zweibyte–Werten ist das Maximum der Amplitude ±32767. Die entstehenden bösen Nägel ließen sich zwar durch eine passende Begrenzung vermeiden, aber sie sind ein deutliches Indiz für Korrekturbedarf. Überhaupt sollte man mit der Amplitude vorsichtig verfahren, denn gesetzt der Fall, man hätte vier gleiche Töne gleichzeitig, so würden diese sich gnadenlos addieren. Was war noch? Die erklingende Kadenz (C–Dur — F–Dur — G–Dur — C–Dur) war in der letzten Version einen Halbton zu hoch — nicht wichtig, aber dennoch erwähnenswert.

Aktuelle Version WE3–03 zum Runterladen

Wie schon bei den Versionen WE3–01 und WE3–02 wird auch diesmal aller Code zur Verfügung gestellt. Gforth gibt es (unter anderem) für Linux und Windows, ich habe beides getestet. Abb. 4 zeigt den Aufruf unter Linux. (Um genau zu sein: Point Linux, das auf Debian aufsetzt.)

```
jgt@point:~$ cd Wave_Engine_8
jgt@point:~/Wave_Engine_8$ gforth WAVER3-03.fs

included: goodies.fs
included: freqtab-1.fs
included: wavgen.fs
included: partitur.fs
included: singen-2.fs

WAVER3-03   >>>--->  Mit 'go' starten!  <---<<<

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

go ..... fertig! <0> <0>
ok
```

Abbildung 4: Protokoll des Aufrufs und Ablaufs mit Linux

Man sieht in **Abb. 4**, ich hatte die relevanten Dateien in einem Ordner „Wave_Engine_8“, der im „Persönlichen Ordner“ steckt. Im Umgang mit meinem neuen Ultrabook bin ich noch nicht sattelfest, mit Windows 8 noch weniger. Es ist mir nicht gelungen, den Ordner „Wave_Engine_8“ in den Ordner „Hannes“ zu verschieben, also musste er für diesmal auf dem Desktop bleiben. **Abb. 5** zeigt, wie mir der Aufruf in der PowerShell gelang.

```
PS C:\Users\Hannes> cd Desktop\Wave_Engine_8
PS C:\Users\Hannes\Desktop\Wave_Engine_8>
                  \"Program Files (x86)"\gforth\gforth WAVER3-03.fs
```

Abbildung 5: Aufruf mit Windows 8.1

Für Windows 7 dürfte „Program Files (x86)“ durch „Programme“ oder „Programs“ zu ersetzen sein. Das Forth steckt jedenfalls im Ordner „gforth“, und das Ergebnis ist eine abspielbare Datei „`WE-001.wav`“. Mein Windows zierte sich beim Abspielen, aber mit „Audacity“ (das gibt's kostenlos für Linux und Windows) geht es allemal. Diese Anwendung zeigt die Kurven optisch und übernimmt auch die Wandlung nach „mp3“.

Ich habe in den Dateien auf Umlaute verzichtet wegen der besseren Lesbarkeit unter Windows. Der Windows–Editor kommt mit Linux–Text nicht zurecht, Wordpad aber schon.

Die Dateien stecken in „Wave-Engine-Ver3-03.zip“, Fundort am Ende des Artikels. Dieses Zip-Paket ist auch auf meiner Homepage zu finden.

Ausblick auf Künftiges

Mir war wichtig, an einem einfachen Beispiel (2 Manuale, je 2 Töne mit je 2 Obertönen) das Prinzip durchzuspielen, das beim allmählichen Aufblähen auf volles Werk (4 Manuale, je 6 Töne mit je 16 Obertönen) erhalten bleiben kann. Leider ist meine alte (solchermaßen aufgeblähte) Version nicht mehr lauffähig, aber die Neufassung wird allemal besser und übersichtlicher als der zusammengebastelte alte Klotz.

Ich habe „Response“ aus der Leserschaft erhalten, deshalb werde ich weiter berichten. Hanno Schwalm hat vorgeschlagen, die unzähligen Sinus-Berechnungen durch eine Tabelle zu ersetzen, um das Programm zu beschleunigen. Obgleich die Wave Engine nicht in Echtzeit funktioniert, wäre es doch recht angenehm, nicht gar so lange auf das Ende einer Kompilation warten zu müssen. Gute Idee!

Das große Thema „Syntaxerkennung“ muss ich noch ein bisschen aufschieben, obwohl ohne sie das Abfassen einer „Partitur“ mit mehr als sechzehn Akkorden zur Fleißarbeit wird.

Referenzen

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

Kapazitätsmessung und kapazitive Fühler

Matthias Koch

Wer schon immer einmal sein eigenes Touchpad oder einen kapazitiven Sensor bauen wollte, für den ist dieser Artikel gedacht. Es gibt verschiedene bekannte Möglichkeiten, Kapazitäten zu bestimmen, zum Beispiel über die Resonanzfrequenz in einem Schwingkreis mit bekannter Induktivität oder über die Entladezeit parallel zu einem bekannten Widerstand. Doch da in der Vierten Dimension auch programmiert werden soll, stelle ich diesmal ein Verfahren vor, welches die Kapazität eines Fühlers durch Vergleich mit einem wesentlich größeren, wohlbekannten Referenzkondensator bestimmt.

Prinzip

Das grundlegende Prinzip besteht darin, mit der kleinen Fühlerkapazität Ladung zu schöpfen und zu prüfen, wie oft geschöpft werden muss, bis die Referenzkapazität geladen ist.

Der elektrische Aufbau, der mit einem MSP430-Launchpad ausprobiert werden kann, ist sehr einfach:

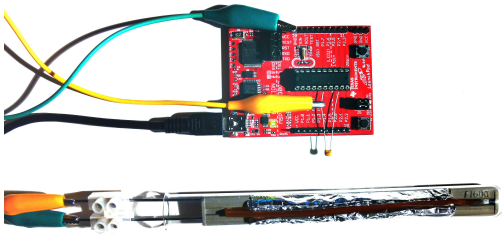


Abbildung 1: LaunchPad mit Fühler — Übersicht des Aufbaus. Matthias zeigt hier einen Fühleraufbau, der auch in Wasser eingetaucht werden kann. Beachte den kleinen Referenz–Kondensator am LaunchPad.

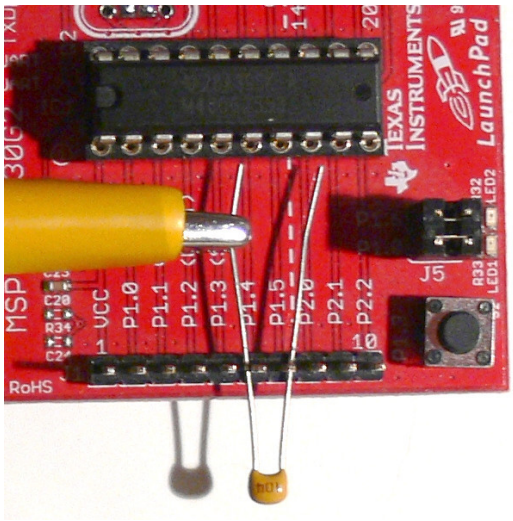


Abbildung 2: Der Referenz–Kondensator am LaunchPad.

Ablauf der Messung

Zu Beginn werden beide Anschlüsse vom Mikrocontroller als Ausgang–Low geerdet — damit ist der Referenzkondensator und der Fühlerkondensator entladen und alles für die Messung bereit. Es beginnt eine Schleife über die folgenden beiden Schritte, wobei in der Implementation stets über „Beide–Anschlüsse–Eingang–Zwischenzustände“ geschaltet wird, damit nicht beide Anschlüsse kurzfristig zugleich Ausgänge sein können:

Erster Schritt: Während der Referenzanschluss Eingang ist und frei schwebt, wird der Fühleranschluss auf Ausgang–High gesetzt. Dadurch wird der Fühlerkondensator geladen. An dieser Stelle wird der Referenzanschluss–Eingang gelesen, denn während der Fühleranschluss getrieben wird, können nicht so leicht Störungen einkoppeln. Um zu verstehen, was dieses gelesene Bit bedeutet, müssen wir den Referenzkondensator und die analogen Eigenschaften der digitalen Eingänge näher betrachten: Der Referenzkondensator ist an der einen Seite über den Fühleranschluss mit der positiven Betriebsspannung verbunden, während die andere Seite schwebt. Ist der Kondensator leer, gibt es keine Spannungsdifferenz zwischen seinen Beinchen, und auch der schwebende Eingang liegt auf High–Pegel. Ist allerdings bereits Ladung im Referenzkondensator vorhanden, so ergibt sich an dieser Stelle eine Spannungsdifferenz, und der schwebende Eingang spürt eine Spannung, die mit zunehmender Ladung des Referenzkondensators immer weiter sinkt. Da die digitalen Eingänge eine Hysterese haben, springt der gelesene Zustand auf Low, wenn der Referenzkondensator weit genug geladen ist, um mit seiner Spannungsdifferenz zur positiven Betriebsspannung hin die untere Schaltschwelle zu erreichen.

Zweiter Schritt: Der Fühleranschluss wird zum Eingang, während der Referenzanschluss auf Ausgang–Low gesetzt wird. An dieser Stelle sind beide Kapazitäten parallel geschaltet und verteilen ihre Ladung so, dass an beiden Kondensatoren schließlich die gleiche Spannung anliegt. Da die kleine Fühlerkapazität im ersten Schritt auf die volle Betriebsspannung aufgeladen worden ist und der Referenzkondensator zu Beginn entladen ist, steigt so die Spannung im Referenzkondensator mit jedem Schritt ein klein wenig an und nähert sich nach vielen Schritten in Form einer Sättigungskurve der Betriebsspannung. Auch in diesem Schritt könnte theoretisch der Ladezustand des Referenzkondensators abgefragt werden, da hier jedoch der Fühleranschluss „offen“ ist, könnten Störungen leichter einkoppeln.

Aus der Anzahl an Ladeschritte, die nötig gewesen sind, um den Referenzkondensator bis zum Auslösen der Hystereseschaltschwelle zu laden, kann die Fühlerkapazität berechnet werden. Details dazu können gerne bei mir erfragt werden, doch praktisch genügt es zu wissen, dass

eine kleine Zahl von Ladeschritten eine große Fühlerkapazität bedeutet. Sind sehr viele Ladeschritte nötig, ist die Fühlerkapazität im Vergleich zur Referenzkapazität klein. Um eine feine Auflösung zu erreichen, sollte die Größe der Referenzkapazität so gewählt werden, dass viele Ladeschritte nötig sind, aber nicht zu viele, denn mit der Zahl der Ladeschritten steigt leider auch der Einfluss von Leckströmen und Schwankungen in der Temperatur oder bei der Betriebsspannung. Viele Faktoren spielen eine Rolle, so dass es am besten ist, mit dem fertigen Aufbau Messungen zur Kalibration durchzuführen. Bewährt hat sich ein Bereich von einigen hundert bis hin zu einigen tausend Ladeschritten pro Kapazitätsmessung, für einen Berührungsfühler mit typischerweise einigen hundert Picofarad ist ein 100 nF–Folienkondensator eine gute erste Wahl zum Ausprobieren. Sehr viel größer sollten die Kapazitäten jedoch nicht werden, da sonst größere Ströme zum schnellen Laden nötig wären, welche die Pins des Launchpads nicht liefern können.

Experimente mit der Kapazitätsmessung

Zum Ausprobieren ist es praktisch, einen 100 nF–Kondensator mit kurzen Leitungen zwischen P1.4 und P1.5 an das Launchpad anzuschließen und den Fühleranschluss P1.4 sowie GND mit Krokodilklemmenkabel herauszuführen. Daran könnten unterschiedliche „Fühler“ angeklemmt und ausprobiert werden, zum Beispiel ein Stück zweiadriges Kabel, zwei Aluminiumfolien unter einem Plastikdeckel, oder auch einmal probeweise ein einzelnes Stück Rohr, dass dann gegen Erde eine sehr kleine Kapazität bildet.

Michael Kalus hat bereits erste Experimente durchgeführt, so dass hier Ergebnisse zu bestaunen sind (Abbildungen 3 und 4).

Aus dieser einfachen Kapazitätsmessung lassen sich leicht interessante Kombinationen bilden — beispielsweise kann mit zwei von diesen Fühlern bereits eine einfache Positionserkennung aufgebaut werden ! Hier lohnt es sich, der Phantasie freien Lauf zu lassen, Anwendungen gibt es genug — beispielsweise die Messung des Füllstandes in Trinkgläsern, siehe Link.

`www.merl.com/papers/docs/TR2002-21.pdf`

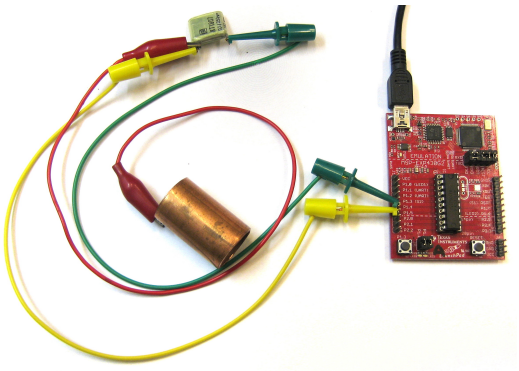


Abbildung 3: Experiment „mika“: Aufbau eines kapazitiven Fühlers — mal eben so — am LaunchPad. Das Stück Kupferrohr wird mit der Hand berührt, ist der Sensor.

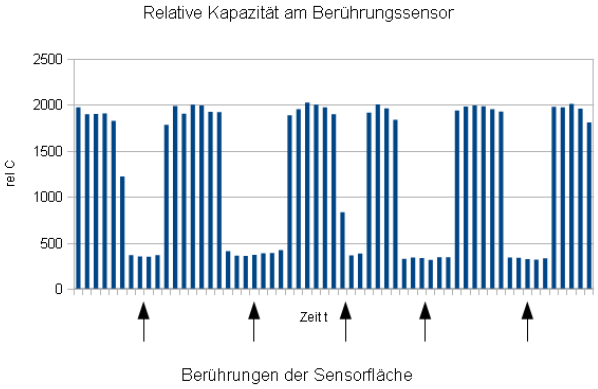


Abbildung 4: Die Messwerte der Kapazität aus dem Experiment „mika“ wurden in ein Tabellenkalkulationsprogramm übertragen und graphisch dargestellt..

Quelltext

```
1   \ Der Quelltext ist verfasst für mecrisp für das TI MSP430 Launchpad.
2
3   \ Aufbau:
4   \ P1.4 und P1.5 tragen den Messkondensator,
5   \ P1.4 ist mit der einen Sensorfläche verbunden,
6   \ die andere Sensorfläche liegt an GND.
7
8   compiletoflash
9
10  16 constant Fuehler     \ o--+---------------
11                          \    |               |
12                          \   +|              +|
13                          \   ===  Cm         === Cf
14                          \   -|              -|
15                          \    |               |
16  32 constant MessC       \ o---              GND
17
```

```
18  \ Dabei ist es notwendig, dass Cm >> Cf.
19  \ Ein Kondensator Cm ~ 100nF ist ein guter Anfang.
20
21  : Kapazitaet ( -- Kapazitaetswert )
22
23    0 \ Durchlaufzaehler. Am Anfang ist noch kein Zyklus geschehen.
24
25    \ Initialisierung
26    Fuehler MessC or P1OUT cbic! \ Vorbereitung, Entladen. F low, M low.
27    Fuehler MessC or P1DIR cbis!
28
29    begin                \ Erster Einlauf: Übergangszustand. F (low), M in
30      MessC P1DIR cbic! \ Später:         Übergangszustand. F (in),  M in
31
32      \ Erster Schritt:
33      Fuehler P1OUT cbis! \ Laden und Messen. F high, M (in)
34      Fuehler P1DIR cbis!
35
36    MessC P1IN cbit@ while \ Falls MessC Low, Ende.
37      1+ \ Ein Zyklus mehr...
38
39      \ Zweiter Schritt:
40      Fuehler P1DIR cbic! \ Übergangszustand. F in, M (in)
41      MessC   P1DIR cbis! \ Mischzustand. F (in), M low
42                          \ MessC ist in P1OUT noch von Schritt (1) low.
43    repeat
44
45    Fuehler MessC or P1OUT cbic! \ Stelle Grundzustand wie bei der Initialisierung
46    Fuehler MessC or P1DIR cbis! \ wieder her. Beide entladen: F low, M low.
47
48  ; \ Kapazitaet
49
50  compiletoram
51
52  : messen ( -- ) begin cr kapazitaet . 1000 ms ?key until ;
```

... Fortsetzung der Meldung zum Forth Day 2013

Abbildung 1: Gruppenfoto vom SVFIG–Meeting

net2o — das Internet neu erfinden, Teil 3: Signaturen

Bernd Paysan

Ein Ding fehlte im Teil 2 noch: Signaturen. Die reiche ich nach, und liefere ein paar Erklärungen, wie das denn so funktioniert. Und wie man das gleiche Schlüsselpaar sowohl für Signaturen als auch für den Diffie–Hellman–Exchange nutzen kann.

Wozu signieren?

Den Verbindungsaufbau, haben wir ja beim letzten Mal [1] gesehen, kriegt man ohne Signaturen hin. Auch die Authentizität — ja, das ist zwar symmetrisch, aber wenn Alice weiß, dass sie die Message selbst nicht geschrieben hat, muss es Bob sein. Für Bob ist diese Situation am besten, denn er kann sich im Zweifelsfall immer damit herausreden (also z.B. beim Auffinden kompromittierender Liebesbriefe, die von der eifersüchtigen Eve entdeckt werden), dass Alice diese selbst geschrieben hätte.

„Unabstreitbarkeit“ nennt man dieses Feature von digitalen Signaturen, das Fehlen derselben hilft dann bei der Abstreitbarkeit. Wann also will ich etwas Unabstreitbares haben? Oder einen Beweis für den Besitz des geheimen Schlüssels, denn mehr ist das ja nicht?

Distributed Hash Table

Ein P2P–Netz braucht eine verteilte Datenbank, um Dinge zu speichern. Dinge wie „wohin sende ich Messages für Bob?“ oder „wer hat dieses oder jenes Dokument?“. Das packt man in eine Hash–Table, die die Hashes von Dokumenten und öffentlichen Schlüssel enthalten (die Schlüssel selbst nochmal zu hashen bringt allerdings wenig).

Nun kann ja rein theoretisch jeder Beliebiges in so eine Hash–Table eintragen oder aus ihr löschen. Das wollen wir eher nicht. Es muss also eine Möglichkeit geben, solche Eintragungen zu verifizieren. Bei der „wer ist wo“–Frage ist klar, wer da unterschreiben muss: Der, der seine Adresse(n) angibt. Beim Eintragen selbst kann der empfangende Server das mit der Authentizitätsprüfung verbinden, das reicht — der Server weiß, wer ihn von wo aus anspricht. Aber diese Server replizieren ihre Daten, weshalb das Ganze ja „distributed“ heißt. Der nächste Server kann zwar sehen, woher diese Information kommt, aber nicht verifizieren, ob sie korrekt ist.

Und genau hier wird's Zeit für eine digitale Signatur. Die erlaubt es nämlich, derartige Prüfsummen weiterzuleiten.

Ed25519 von Dan Bernstein

Etwas Mathematik

Die Curve25519 von Dan Bernstein hat inzwischen ein kleines Geschwister bekommen, nämlich eine Kurve in Edwards–Form. Diese Form hat Harold M. Edwards 2007 entdeckt. Die Kurve ist vierfach symmetrisch, und sieht je nach Parameter aus wie in Abbildung 1.

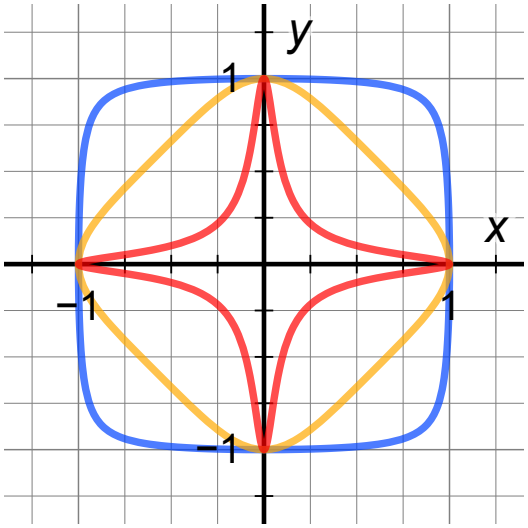


Abbildung 1: Edwards–Kurve der Gleichung $x^2+y^2=1-d\cdot x^2\cdot y^2$ über die reellen Zahlen für $d=300$ (rot), $d=\sqrt{8}$ (gelb) und $d=-0.9$ (blau) [2]

Zumindest über die reellen Zahlen sieht die Kurve so aus... aber wenn man Kryptographie macht, rechnet man nicht mit reellen Zahlen. Das dient also nur der Vorstellung. Über diese Kurve definiert man eine Additions–Operation, die der Addition zweier Zeiger auf einem Kreis (also Winkeladdition) formal entspricht. Wenn wir einen Kreis vom Radius 1 hätten, wäre $d=0$, also $x^2+y^2=1$. Zwei Punkte auf dem Kreis mit den Winkeln α_1 und α_2 hätten also die Koordinaten $x_1,y_1=\sin\alpha_1,\cos\alpha_1$ und $x_2,y_2=\sin\alpha_2,\cos\alpha_2$. Um die beiden Winkel zu addieren, müssen wir nur in die verstaubte Formelsammlung aus dem Abitur gucken (oder in Wikipedia), und sehen:

$$\sin(\alpha_1+\alpha_2)=\sin\alpha_1\cos\alpha_2+\sin\alpha_2\cos\alpha_1=x_1y_2+x_2y_1$$

$$\cos(\alpha_1+\alpha_2)=\cos\alpha_1\cos\alpha_2-\sin\alpha_1\sin\alpha_2=y_1y_2-x_1x_2$$

Auf einer Edwards–Kurve ist die Formel nur etwas anders, schließlich liegen die Punkte mal weiter innen oder weiter außen, abhängig von d:

$$(x_1,y_1)+(x_2,y_2)=\left(\frac{x_1y_2+y_1x_2}{1+dx_1x_2y_1y_2},\frac{y_1y_2-x_1x_2}{1-dx_1x_2y_1y_2}\right)$$

Edwards–Kurven haben hier gegenüber anderen elliptischen Kurven einen Vorteil: Das Gesetz für die Addition hat keine Sonderregelungen, es funktioniert sowohl für das neutrale Element als auch für die Verdopplung einwandfrei. Natürlich kann man bei der Verdopplung „Abkürzungen“ nutzen, weil Terme doppelt vorkommen, aber man muss nicht.

Wenn wir eine Addition haben, mit neutralem Element (dem Winkel 0, also dem Punkt $(0,1)$), dann gibt es auch ein Skalarprodukt (wiederholte Addition), also die Multiplikation mit einer natürlichen Zahl. Dieses Skalarprodukt ist die Basisoperation für die Kryptographie mit elliptischen Kurven. Allerdings nicht über Kurven, die reelle Zahlen nutzen, sondern über Kurven in einem Restklassenring. Wir haben also (sehr lange) Integer, die modulo einer großen Primzahl genommen werden, in diesem Fall ist das $2^{255}-19$, die Zahl ist auch Namensgeber der Kurve. Wenn man so nah an einer Zweierpotenz ist, lässt sich der Rest sehr viel einfacher berechnen.

Die Punkte auf der Kurve haben ebenfalls einen Zyklus, der etwas kürzer ist als die der Koordinate, und zwar $l = 2^{252} + 27742317777372353535851937790883648493$. Auch hier hilft die Nähe an einer Zweierpotenz, den Rest schneller auszurechnen — und auch diese Zahl ist eine Primzahl.

Schlüsselpaare

Als geheimen Schlüssel wählen wir eine zufällige Zahl $[sk] \in \mathbb{Z}_l$, wobei es da völlig reicht, wenn man z.B. dafür sorgt, dass sie kleiner als 2^{252} ist, aber nicht zu viel kleiner (das oberste Bit sollte gesetzt sein). Unser öffentliche Schlüssel ist dann eine Skalarmultiplikation vom Basis–Punkt der Kurve mit diesem geheimen Schlüssel. Zur Nomenklatur: Punkte auf der Kurve sind Variablen ohne Kennzeichnungen, Skalare in eckigen Klammern:

$$pk = [sk] * base$$

Diffie–Hellman–Schlüsselaustausch

Wie beim letzten Mal erklärt, multipliziert man beim Schlüsselaustausch einfach seinen geheimen Schlüssel mit dem öffentlichen Schlüssel des anderen, was in beiden Fällen den gleichen Punkt auf der Kurve gibt, weil

$$[sk_2] * pk_1 = [sk_2] * [sk_1] * base = [sk_1] * pk_2$$

ist. Den öffentlichen Schlüssel codiert Dan Bernstein als x–Koordinate, also ebenfalls in 32 Bytes, wie auch den privaten Schlüssel. Das ist schön kompakt, man muss den öffentlichen Schlüssel dafür beim Empfangen expandieren, also die zugehörige y–Koordinate ausrechnen. Inzwischen nutze ich auch für den Schlüsselaustausch die Edwards–Form der Kurve; das ist genauso sicher, aber etwas schneller (wegen Optimierungen, die nur in dieser Form möglich sind). Vorteilhaft ist hier, dass man den gleichen Schlüssel sowohl für Signaturen als auch für Verschlüsselung verwenden kann, und damit der ganze Schlüssel sehr kompakt untergebracht werden kann — also auch z.B. einfach auf eine Visitenkarte gedruckt.

Signatur

Um eine Signatur zu berechnen, braucht man den Hash der signierten Daten. Ich nutze auch hier Keccak, und hashe zunächst die Nachricht. Für die Signatur wird der komplette State *state* genutzt, und zwar zweimal, denn es reicht nicht einfach, nur den Hash durch die Gegend zu schicken. Zunächst braucht man für so eine Signatur nämlich eine unterschiedliche pseudo–zufällige Zahl pro Message, die außer dem Signierer keiner kennen darf. Damit wir nicht unsere Zufallsquelle strapazieren müssen (die langsam oder schlecht sein könnte), schlägt Dan Bernstein vor, diese Zufallszahl aus dem Hash über die Message und den privaten Schlüssel zu bilden. Ich absorbiere also den privaten Schlüssel in den Hash–State: $k := \mathrm{hash512}(\mathrm{absorb}(sk, state))$. Das ist eine 64–Byte–Zahl, die reduziert wird in $\mathbb{Z}_l$. Wir rechnen damit einen Punkt r auf der Kurve aus, also $r := [k] * base$.

Diesen Punkt komprimieren wir auf 32 Bytes (nur die x–Koordinate), hängen unseren öffentlichen Schlüssel dran (macht 64 Bytes), absorbieren das wieder in den übergebenen State *state*, und bekommen so die Zahl $z := \mathrm{hash512}(\mathrm{absorb}(r||pk, state))$. Daraus können wir den zweiten Teil der Signatur berechnen, das Skalar $[s] := [z * sk + k]$. Beide Zahlen r, s zusammen ergeben die Signatur (wo nötig, muss noch der Public Key angehängt werden, damit man weiß, von wem die Signatur ist).

Zur Verifikation berechnet der Empfänger z wie oben, und dann vergleicht er

$$r \equiv [s] * base - [z] * pk$$

$$= [z * sk] * base + [k] * base - [z] * [sk] * base = [k] * base$$

Wie man sieht, muss auf der rechten Seite das Gleiche übrig bleiben wie auf der linken steht — dann ist die Signatur gültig. Das Signieren einer Message ist schneller als das Verifizieren, da man für das Signieren nur eine komplexe Operation braucht, nämlich die Skalarmultiplikation mit der *base*. Die kann man mit einer Multiplikationstabelle gut beschleunigen. Trotzdem ist Ed25519 so schnell, dass man auf einem Core i7 mit 3GHz mit einem Thread 14000 einzelne Signaturen in der Sekunde verifizieren kann (Batch–Verifikation geht noch schneller).

Die Implementierung der Mathematik ist in einer schnellen C–Library ausgelagert, die man bei Github herunterladen kann (geforkt von mir, damit der Diffie–Hellman–Schlüsselaustausch auch geht) [3].

Referenzen

[1] Bernd Paysan, *net2o — Das Internet neu erfinden, Teil 2,* VD 2013/03

[2] Wikipedia, *Edwards Curve,* `http://en.wikipedia.org/wiki/Edwards_curve`

[3] Bernd Paysan, *ed25519-donna fork,* `git clone https://github.com/forthy42/ed25519-donna.git -b bernd`

Listings

```
\ Interface to the ed25519 primitives from donna
\ The high level stuff is all in Forth

\ dummy load for Android
[IFDEF] android
    s" /data/data/gnu.gforth/lib/libed25519-prims.so"
    open-lib drop
[THEN]

c-library ed25519_donna
    "ed25519-prims" add-lib
    \c #include <stdint.h>
    \c #include <ed25519-prims.h>
    \c int str32eq(uint64_t* a, uint64_t* b) {
    \c    uint64_t diff=((a[0]^b[0])|(a[1]^b[1])|
    \c                   (a[2]^b[2])|(a[3]^b[3]));
    \c    return -(diff==0);
    \c }

    c-function raw>sc25519 expand_raw256_modm
               a a -- void ( sc char[32] -- )
    c-function nb>sc25519 expand256_modm
               a a n -- void ( sc char[64] -- )
    c-function sc25519>32b contract256_modm
               a a -- void ( char[32] sc -- )
    c-function sc25519* mul256_modm
               a a a -- void ( r x y -- )
    c-function sc25519+ add256_modm
               a a a -- void ( r x y -- )

    c-function ge25519*base ge25519_scalarmult_base
               a a -- void ( ger x -- )
    c-function ge25519-pack ge25519_pack
               a a -- void ( r ger -- )
    c-function ge25519-unpack-
    ge25519_unpack_negative_vartime a a -- n
               ( r p -- flag )
    c-function ge25519*+
    ge25519_double_scalarmult_vartime a a a a -- void
               ( r p s1 s2 -- )
    c-function ge25519*v
    ge25519_scalarmult_vartime a a a -- void
               ( r p s -- )
    c-function ge25519*
    ge25519_scalarmult a a a -- void
               ( r p s -- )
    c-function 32b= str32eq a a -- n
               ( addr1 addr2 -- flag )
end-c-library

: 32b>sc25519 32 nb>sc25519 ;
: 64b>sc25519 64 nb>sc25519 ;

$20 Constant KEYBYTES

$40 12 * $60 + $40 + 200 + $10 + Constant edbuf#

here edbuf# allot $F + -$10 and \ align for SSE

dup Constant sct0 $40 +
dup Constant sct1 $40 +
dup Constant sct2 $40 +
dup Constant sct3 $40 + \ can be between $20 and $30
dup Constant get0 $100 +
dup Constant get1 $100 + \ can be between $80 and $C0
dup Constant sigbuf $60 +
dup Constant hashtmp $40 +
Constant keccaktmp

: gen-sk ( sk -- ) >r
    \G generate a secret key with the
    \G right bits set and cleared
    $20 rng$ r@ swap move  r@ c@ $F8 and r@ c!
    r> $1F + dup c@ $7F and $40 or swap c! ;

: sk>pk ( sk pk -- )
    \G convert a secret key to a public key
    sct0 rot raw>sc25519
    get0 sct0 ge25519*base
    get0 ge25519-pack ;

: ed-keypair ( sk pk -- )
    \G generate a keypair
    over gen-sk sk>pk ;

: >hash ( addr u -- )
    \G absorb a short string, perform a hash round
    \G and output 64 bytes to hashtmp
    >keccak keccak* hashtmp $40 keccak> ;

: ed-sign { sk pk -- sig u }
    \G sign a message: the keccak state
    \G contains the hash of the message.
    @keccak keccaktmp keccak# move
    \ we need this twice - move away
    sk $20 >hash
    \ gen "random number" from secret to hashtmp
    keccaktmp @keccak keccak# move \ restore state
    sct3 hashtmp 64b>sc25519
    get0 sct3 ge25519*base   \ sct3 is k
    sigbuf get0 ge25519-pack   \ sct0 is r=k*base
    pk sigbuf $20 + $20 move
    sigbuf $40 >hash           \ z=hash(r+pk+message)
    sct1 hashtmp 64b>sc25519 \ sct1 is z
    sct2 sk raw>sc25519      \ sct2 is sk
    sct1 sct1 sct2 sc25519*
    sct1 sct1 sct3 sc25519+  \ s=z*sk+k
    sigbuf $20 + sct1 sc25519>32b
    sigbuf $40 ; \ r,s

: ed-check? { sig pk -- flag }
    \G check a message: the keccak state
    \G contains the hash of the message.
    \G The unpacked pk is in get0, so this
    \G word can be used for batch checking.
    sig hashtmp $20 move  pk hashtmp $20 + $20 move
    hashtmp $40 >keccak keccak* hashtmp $40 keccak>
    \ z=hash(r+pk+message)
    sct2 hashtmp 64b>sc25519       \ sct2 is z
    sct3 sig $20 + raw>sc25519     \ sct3 is s
    get1 get0 sct2 sct3 ge25519*+  \ base*s-pk*z
    sigbuf $40 + get1 ge25519-pack         \ =r
    sig sigbuf $40 + 32b= ;

: ed-verify { sig pk -- flag }
    \ message digest is in keccak state
    get0 pk ge25519-unpack-
    0= IF  false EXIT  THEN \ bad pubkey
    sig pk ed-check? ;

: ed-dh { sk pk -- secret len }
    get0 pk ge25519-unpack- 0= !!no-ed-key!!
    sct2 sk raw>sc25519
    get1 get0 sct2 ge25519*
    sct0 get1 ge25519-pack
    sct0 $20 ;

: ed-dhv { sk pk -- secret len }
    get0 pk ge25519-unpack- 0= !!no-ed-key!!
    sct2 sk raw>sc25519
    get1 get0 sct2 ge25519*v
    sct0 get1 ge25519-pack
    sct0 $20 ;
```

EuroForth 2013

Bernd Paysan

Dieses Jahr fand die EuroForth im nicht so bezaubernden, aber genauso teuren Haus Rissen in Hamburg statt, organisiert von Klaus Schleisiek (der mit dem Preis–Leistungsverhältnis auch nicht zufrieden ist) und Ulrich Hoffmann. Wir werden uns für das nächste Mal in Deutschland anderswo umsehen müssen. Zumindest die abendlichen Diskussionen sind im Haus Rissen problemlos bis 2 Uhr nachts möglich. Da das Haus Rissen über die Woche Seminare für unsere Streitkräfte anbietet, können wir davon ausgehen, dass alles belauscht wurde, und zwar nicht nur von einem Geheimdienst.

Das ist allerdings fast egal, denn ich habe alle Vorträge aufgezeichnet und online gestellt.

Forth 200x Meeting

Der laufende Standard, der in Oxford ein Datum bekommen hat, und deshalb Forth 2012 heißt, ist in die nächste Revision gegangen: Es wurde weiter an den Details gefeilt, und ein neuer Release Candidate (RC2), also das Ergebnis unseres Meetings, ist am 13. November veröffentlicht worden [1]. Die Review–Periode beträgt auch dieses Mal wieder 3 Monate; im März werden wir uns (wahrscheinlich elektronisch) wieder treffen, um das zu beraten. Der Standard ist aber so gut wie fertig, wenn sich die Änderungen mehr auf Vorwörter und so konzentrieren.

Manche Sachen sind im Standard nicht so schön geworden, wie sie hätten sein sollen, und das beheben wir natürlich ebenso im weiterhin „rollenden Dokument“, von dem am 9. November die Version 13–1 veröffentlicht wurde (die rollenden Dokumente enthalten immer die Änderungen mit Changebars):

Throw IORs Die Rückgabewerte, die Fehler darstellen, sind jetzt immer THROWable. Das war auch in Forth 2012 so beabsichtigt, damit das aber tatsächlich so im Standard steht, muss der Text an ganz anderen Stellen geändert werden; der Datatyp ior ist nun schon in Abschnitt 3 definiert, und damit eine feste Beziehung zwischen den Rückgabewerten und THROW hergestellt.

TRAVERSE–WORDLIST Viele Forth–Systeme enthalten eine higher order Function, die Wordlists durchläuft. Das ist jetzt standardisiert.

Xchar IO Wie sich halbe Xchars beim IO verhalten, war eigentlich schon länger klar, der Text hat es aber irgendwie nicht ins Dokument geschafft.

EuroForth Konferenz

Das Programm war dicht gepackt mit Vorträgen, von denen ich hier aus Platzgründen einfach nur die Titel aufzähle. Alle Vorträge kann man auf Video ansehen, und sich bei vielen die Slides herunterladen [2].

Freitag, 27.9.

16:00 Anton Ertl “Region-based Memory Allocation”

16:45 Nick Nelson “Forth Query Language (FQL)”

17:30 Andrew Read “Optimizing memory access design for a 32 bit FORTH processor”

18:15 Gerald Wodni “Forth to .NET”

Samstag, 28.9.

9:30 Anton Ertl “Standardize Strings Now!”

10:15 Ulrich Hoffmann “Forth Literate Programming with IPython notebook”

11:30 Willi Stricker “Forth Floating Point WordSet without Floating Point Stack”

12:15 Sergey Baranov “Forth in Russia” Video 12:45 Klaus Schleisiek “Impromptu Talk: Vector multiplication”

14:00 Bernd Paysan “Net2o application layer”

Sonntag, 29.9.

9:30 Anton Ertl “PAF: A portable Forth Assembler”

10:00 Peter Knaggs “Impromptu talk about benchmarks”

10:15 Peter Knaggs “Impromptu talk Standard in LaTeX”

10:30 Klaus Schleisiek “Impromptu talk ’You broke the Internet’”

10:45 Gerald Wodni “Impromptu talk Phaseout”

11:00 Stephen Pelc “Impromptu talk Forth20xx”

Referenzen

[1] Forth 200x Standards Committee #9, *Forth 200x draft documents,* `http://www.forth200x.org/documents/`

[2] *EuroForth 2013 Proceedings,* `http://www.complang.tuwien.ac.at/anton/euroforth/ef13/papers/`

Forth–Gruppen regional

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

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

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

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

Gruppengründungen, Kontakte

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

μP–Controller Verleih

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

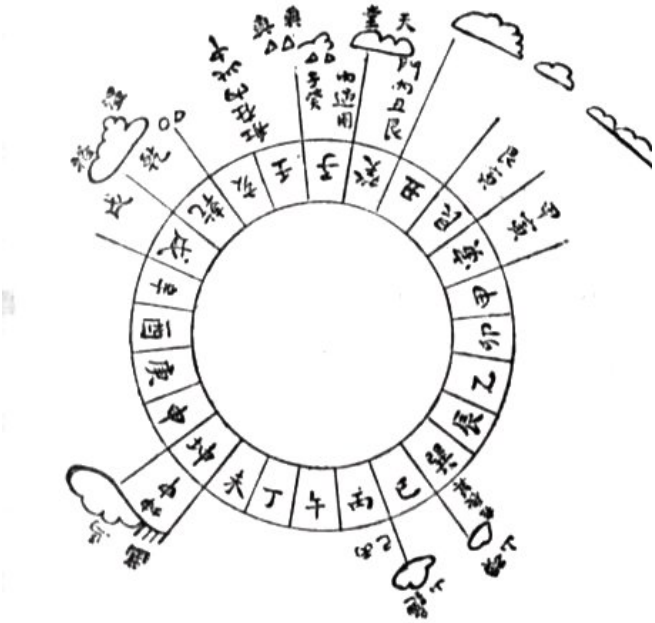
Spezielle Fachgebiete

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

Termine

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

27.–30. März 2014: Forth–Tagung, Bad Vöslau bei Wien
25.–28. Juni 2014: LinuxTag, Berlin



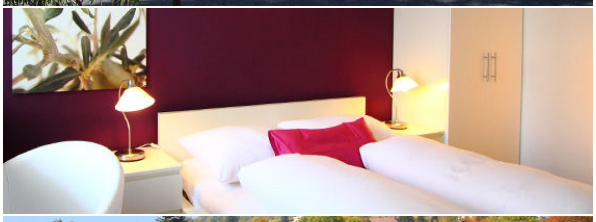
Möchten Sie gerne in Ihrer Umgebung eine lokale Forthgruppe gründen, oder einfach nur regelmäßige Treffen initiieren? Oder können Sie sich vorstellen, ratsuchenden Forthern zu Forth (oder anderen Themen) Hilfestellung zu leisten? Möchten Sie gerne Kontakte knüpfen, die über die VD und das jährliche Mitgliedertreffen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

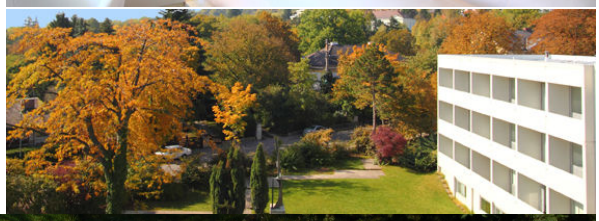
Hinweise zu den Angaben nach den Telefonnummern:
Q = Anrufbeantworter
p = privat, außerhalb typischer Arbeitszeiten
g = geschäftlich
Die Adressen des Büros der Forth–Gesellschaft e.V. und der VD finden Sie im Impressum des Heftes.

Einladung zur

Forth–Tagung 2014 vom 27. bis 30. März in Bad Vöslau

Unterbringung und Tagung im College Garden Hotel in der Johann–Strauß–Straße 2, 2540 Bad Vöslau, Österreich.





Anreise

Bad Vöslau liegt südlich von Wien und ist mit dem Auto über die A2 erreichbar. Das Hotel bietet einen Taxiservice vom Bahnhof, oder auch vom Flughafen.

Anmeldung

Auf `http://tagung.forth-ev.de` finden sich noch viele weitere Informationen, das aktuellste Programm sowie die elektronische Anmeldung.

Programm

Donnerstag

ab 13:00	Frühankommer(–Workshops)

Freitag

vormittags	Frühankommer(–Workshops)
nachmittags	Begin der Tagung, Vorträge und Workshops

Samstag

vormittags	Vorträge und Workshops
nachmittags	Exkursion

Sonntag

09:00	Mitgliederversammlung
13:00	Ende der Tagung