Vierte Dimension

2/2021

Das Forth–Magazin

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

In dieser Ausgabe:

Dienstleistungen und Produkte

tematik GmbH
Technische
Informatik

Feldstraße 143
D–22880 Wedel
Fon 04103 – 808989 – 0
Fax 04103 – 808989 – 9
mail@tematik.de
`http://www.tematik.de`

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

Forth–Schulungen

Möchten Sie die Programmiersprache Forth erlernen oder sich in den neuen Forth–Entwicklungen weiterbilden? Haben Sie Produkte auf Basis von Forth und möchten Mitarbeiter in der Wartung und Weiterentwicklung dieser Produkte schulen?

Wir bieten Schulungen in Legacy–Forth–Systemen (FIG–Forth, Forth83), ANSI–Forth und nach den neusten Forth–200x–Standards. Unsere Trainer haben über 20 Jahre Erfahrung mit Forth–Programmierung auf Embedded–Systemen (ARM, MSP430, Atmel AVR, M68K, 6502, Z80 uvm.) und auf PC–Systemen (Linux, BSD, macOS und Windows).

Carsten Strotmann carsten@strotmann.de
`https://forth-schulung.de`

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

Cornu GmbH
Ingenieurdienstleistungen
Elektrotechnik

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

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

KIMA Echtzeitsysteme GmbH

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

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

FORTecH Software GmbH

Tannenweg 22 m D–18059 Rostock
`https://www.fortech.de/`

Wir entwickeln seit fast 20 Jahren kundenspezifische Software für industrielle Anwendungen. In dieser Zeit entstanden in Zusammenarbeit mit Kunden und Partnern Lösungen für verschiedenste Branchen, vor allem für die chemische Industrie, die Automobilindustrie und die Medizintechnik.

Ingenieurbüro
Klaus Kohl–Schöpe

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

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

Mikrocontroller–Verleih
Forth–Gesellschaft e. V.

Wir stellen hochwertige Evaluation–Boards, auch FPGA, samt Forth–Systemen zur Verfügung: Cypress, RISC–V, TI, MicroCore, GA144, SeaForth, MiniMuck, Zilog, 68HC11, ATMEL, Motorola, Hitachi, Renesas, Lego ...
`https://wiki.forth-ev.de/doku.php/mcv:mcv2`

Titelbild: In der Architektur ist Licht die vierte Dimension. Autor mk
Quelle: Irgendwo aus dem Internet und selbst modifiziert.

Impressum

Name der Zeitschrift
Vierte Dimension

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

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

Anzeigenverwaltung
Büro der Herausgeberin

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

Erscheinungsweise
1 Ausgabe / Quartal

Einzelpreis
4,00 € + Porto u. Verpackung

Liebe Leser,

das zweite Heft dieses Jahres war ungewöhnlich leicht zusammenzustellen. So macht Zeitung Spaß. Und der Trend hält an, sind doch schon zwei weitere Beiträge für das Folgeheft in der Warteschlange. Was nicht heißen soll, dass ihr euch nun alle zurücklehnen könntet. :)

In der Architektur bezeichnet man Licht als die *vierte Dimension* beim Bauen. Im Aufmacher, dem Titelbild, sah ich alle Aspekte dieses Heftes eingefangen: das Thema *Stapel* als Platinen–Stack oder die Stacks im Forth selbst, der Ausblick auf die helle Zukunft quelloffener moderner *Prozessoren im FPGA*, der Weg via *GPIO* auch aus starken Maschinen wie dem Pi4 heraus nach draußen, und alles zusammengefasst durch etwas Viertes, das Licht, das wir hinein bringen, wie *HolonForth* ins Programmieren. Wenn ihr das Titelbild um 90 Grad gegen den Uhrzeigersinn dreht, seht ihr in einen lichtdurchfluteten Gang. Und außerdem steht das Bild für die Hoffnung auf einen glücklichen Ausgang aus der Pandemie — Licht am Ende des Ganges, da geht es hinaus und zurück in die Freiheit.

Erich Wälde stellt seine *stapelbaren Platinen* vor. In dem besagten Rudel war ich übrigens zugegen und kann das bezeugen! Von komplexen Lösungen bis zum 1–Anschlussdraht–Modell war alles an Ideen dabei. Hier also die komplexere Lösung. Unser damaliger Elan verflog dann bald, weil auf einmal diese spottbilligen Entwicklungs–Kits wie der MSP430G2553 für nur 4,30 US$ auf den Markt kamen und eine Serie an Forth lostraten, wie das eForth, noForth, AmForth, Mecrisp und wohl noch einige andere. Aber Erich hat offensichtlich durchgehalten.

Zeit und Geschick hatten auch Matthias Koch und Bruno Levy, und haben es tatsächlich vollbracht, einen kleinen *RISC–V* im Icestick zu bauen. An sich ja kein Forth, sondern VERILOG, aber auf dem RISC–V läuft Forth! Fragt einfach danach.

Auf höchstem Niveau basteln mit Luxusprozessoren, die für Workstation&Server–OS (Linux) da sind? Ja, geht. Bernd Paysan hat sich den Pi4 mit seinem 40–poligen Pfostenstecker vorgenommen und kommt mit Gforth nun hinaus ins Freie! *GPIOs* hats da ja genug. Fette Forthsysteme kommen wieder direkt an die Peripherie heran. Lest selbst, wie es gelang.

Und zu meiner großen Freude kehrt ein wunderbares Werkzeug zurück, inzwischen in Tcl/Tk programmiert. Wolf Wejgaard hat *Holon* weiterentwickelt. Programmentwicklung, Bearbeiten, Laden, Testen und Debuggen erfolgen im *Browser*. In dieser stabilen Umgebung kann man sich ohne Ablenkung auf die Anwendung konzentrieren — wunderbar.

Eine traurige Nachricht gibt es auch. Unser Vereinsmitglied Elisabeth Rohrmayer ist im Juni verstorben, wie ihre Tochter mitteilte. Elisabeth, wir erinnern uns dankbar an deine treuen Besuche der Mitgliederversammlungen und deine stille Förderung des Vereins hinter den Kulissen all die Jahre.

Und was macht unsere diesjährige *Forthtagung*? Sie wird noch einmal eine Videokonferenz sein, ohne leibhaftige Zusammenkunft.

Bis bald, euer Michael

Die Quelltexte in der VD müssen Sie nicht abtippen. Sie können sie auch von der Web–Seite des Vereins herunterladen.
`http://fossil.forth-ev.de/vd-2021-02`

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

Ulrich Hoffmann
Bernd Paysan
Carsten Strotmann

Kontakt: Direktorium@Forth-ev.de

Mecrisp–Ice `pause`

Christof E. fragte:

> „In der 4d2021–01, im Artikel ‚Multitasking für Stackprozessoren', den ich sehr interessant fand, habe ich ziemlich lange gebraucht, um zu erahnen, was `pause` dort ist. Es muss wohl so eine Art ‚Hook–Wort' sein, das man mit `!` verändern kann. Ggf. wäre eine Erläuterung einen Nachtrag wert?"

Matthias Antwort:

Lieber Christof, gut beobachtet, hier kommt die Auflösung:

In *Mecrisp–Ice*, welches komplett im RAM läuft, ist `pause` eine ganz normale Definition — und enthält im Normalfall einfach nur einen Sprung zu `nop`:

```
see pause 037E : 01BA Jmp 0374 --> nop ok.
```

Wenn die Adresse einer Definition durch 2 geteilt wird, ergibt sich im Befehlssatz des Stackprozessors ein unbedingter Sprung zu ebendieser Definition, und so können die beiden mysteriösen Zeilen die Funktion von `pause` umschalten:

```
: multitask  ( -- ) ['] (pause) 2/ ['] pause ! ;
: singletask ( -- ) ['] nop 2/ ['] pause ! ;
```

Das Ergebnis sieht so aus:

```
multitask ok.
see pause 037E : 0FEA Jmp 1FD4 --> (pause) ok.
```

Der Trick kann natürlich auch verwendet werden, um versuchsweise jede beliebige Definition umzuleiten, es muss nur der jeweils erste Befehl der Definition gemerkt werden, falls die Änderung später rückgängig gemacht werden soll. Bei Neugier sei der ausgiebige Gebrauch des Disassemblers[1] empfohlen!

Viele Grüße, Matthias mk

Von Glyphen und Zeichen

Bei der Codierung von Schriften geht es darum, deren *Glyphen*[2] irgendwie runter in die Bits zu kriegen, damit sie mit Hilfe der kybernetischen Maschinen global verbreitet werden können. Und diese Projektion soll mit den gleichen Maschinen auch kulturspezifisch funktionieren, also trotz der so verschiedenen Kulturen auf unserer Erde. Dazu bedarf es kulturübergreifender Verabredungen[3], eine gemeinsame Meta–Kultur sozusagen und damit einer Verständigung auf die Begriffe dafür.

Im Forth, das aus dem englischen Sprachraum kam, waren wir lange zufrieden mit dem *Charakter* als Code für eine Glyphe.

In unseren Stackkommentaren steht noch immer das `c` als Abkürzung dafür.

```
KEY  ( -- c )
EMIT ( c -- )
```

Und Forthphrasen wie `CHAR A` oder `[CHAR] A` werden im Quellcode von Forthprogrammen benutzt, um den Code einer lateinischen Glyphe ins Programm zu compilieren.

Das hat bislang ganz gut geklappt, weil wir dabei von den lateinischen Glyphen, den Buchstaben, codiert in *ASCII*, ausgegangen sind. Mit Zeichensätzen wie *Latin–1*, dem zuletzt 1998 aktualisierter Standard für die Informationstechnik ... oder so.

Abbildung 1: Setzkasten

Die Älteren unter uns erinnern sich noch an die Probleme mit der Darstellung unserer deutschen Umlaute auf den englischen Maschinen. Die kamen im ASCII nun mal nicht vor. Man schrieb dann `ae` statt `ä` und `oe` statt `ö`, `ss` statt `ß` und solcherlei Spachverbiegungen in seinem Forthquellcode. Die englische Maschine zwang uns zur Anpassung an ihre Beschaffenheit.

Doch die Leute haben nicht aufgehört, ihre eigene Sprache und Schrift zu fordern, ihre eigene Kultur gegen die englische Maschine zu stellen. Und hatten Erfolg damit. Heute gibt es UTF–8, worin alle Glyphen dieser Welt abgebildet sind. Die unterschiedlichsten heiligen Schriften sind daher nun digitalisiert. Und wer will, dass nur ein kleiner Kreis Eingeweihter ihr Geschriebenes lesen kann, macht seine eigenen Hieroglyphen[4].

Die Schriftzeichen, die Glyphen, wenn nicht handgemalt, sind seit Gutenberg als bewegliche Lettern[5] in Setzkästen (Abb. 1) aufgehoben. Die aus Blei gegossenen Lettern haben je nach Charakter der Schrift, ihrem Typ, einen eigenen Setzkasten. Im Computer sind die Zeichencodes ebenfalls zu Zeichensätzen zusammengefasst[6]. Und weil der Zeichenvorrat, der in ein Byte passt, also 256 Stück,

[1] Welcher ja dabei ist bzw. nachgeladen werden kann.

[2] Die grafische Darstellung eines einzelnen Schriftzeichens heißt Glyphe

[3] Engl.: character encoding, kurz: encoding)

[4] Griechisch „hieros" für „heilig" und „glyphos" für „eingeritzt"

[5] Von französisch „lettre", aus lateinisch „littera", „Buchstabe"; deutsch: Drucktypen bzw. Typen; englisch: character, also der „Charakter", die Eigenart der Glyphen.

[6] ccs, coded character set, oder codepage

für alle Glyphen dieser Welt und alle, die noch kommen werden, nicht reicht, wurde auf UTF–8 erweitert. Man hängt bei Bedarf einfach weitere Bytes dran.

Begrifflich kommt man dabei aber in Schwierigkeiten. Die Glyphe entspricht nicht mehr 1:1 dem `CHAR`, hat also nicht mehr nur ein Element, sondern deren Code kann jetzt eine kleine Reihe bytegroßer Elemente sein. Statt mit `CHARs` haben wir es nun mit Reihen von *codeunits* (cu) zu tun.

Und wenn man die ganze Reihe meint, also den ganzen *string* der Glyphe, wie soll das dann genannt werden? Der Programmierer stellt sich das als *array* im Speicher vor, und da interessiert nur der Startpunkt und die Länge davon. Darum spricht er auch vom *codepoint* (cp), dem Codepunkt, wenn er eigentlich das ganze Array meint.

Der Codepunkt `cp` repräsentiert also nun den ganzen *encoded character*, somit unsere Glyphe.

```
cp = {cu0, cu1, cu2 ... }
```

In welcher Form der Codepunkt codiert worden ist, muss ebenfalls mitgeteilt werden. Weiß man nichts über die CEF[7] und das CES[8], macht alles keinen Sinn.

Hier noch schnell ein Encoding–Beispiel. Das chinesische Schriftzeichen für Berg ist shān.[9] Es hat im *Unicode* den Codepunkt `U+5C71` und benötigt zur Darstellung 15 Bit. Nimmt man als CEF das UTF–16, passt es in eine Codeeinheit. Ist sein CES big–endian, steht `5C 71` im Speicher, mit little–endian `71 5C`. Bei UTF–8 hingegen stehen drei Codeeinheiten im Speicher: `E5 B1 B1` (Wieso?). Die resultierende Glyphe ist bei allen 山. Soviel mal zur Begriffsklärung.

Im letzten Heft (4d2021–01) hat Anton Ertel ausführlich dargelegt, welche xchar–Wörter[10] Forth nun hat. Nun fehlen mir einige schöne Beispiele, wo das im Forth auch zum Tragen kommen könnte. Vielleicht hilft mir jemand aus Übersee auf die Sprünge? mk

`https://de.wikipedia.org/wiki/Zeichenkodierung`

Retro Forth Web–Terminal

Rick Carlino hat *Websocketd*[11] mit Retro Forth kombiniert, um Retro–Forth–Sitzungen mittels eines Docker–Containers in einem Browser via Websockets anzuzeigen. Websocketd kann die Bildschirmausgaben und Tastatureingaben beliebiger Unix/Linux–Programme auf eine Webseite umlenken. Diese Technik ist nicht auf Retro Forth beschränkt, Ricks Beispiel–Docker–Container kann als Vorlage für Web–Versionen anderer Forth–Systeme genommen werden (GNU/Forth, lina, muForth etc.). Aber Achtung: Wer ein Forth–System über das Internet für beliebige Benutzer freigibt, sollte wissen, dass die Benutzer volle Kontrolle über das System (bzw. über den Docker–Container) erlangen können. Daher sind für solche Anwendungen eine gute Firewall und Authentisierung der Benutzer, z. B. über einen Reverse–Web–Proxy, empfehlenswert.

Retro Forth — was war das nochmal?

> „Retro is a modern, pragmatic Forth drawing influences from many sources. It's clean, elegant, tiny, easy to grasp, and adaptable to many tasks.
>
> It's not a traditional Forth. Drawing influence from colorForth, it uses prefixes to guide the compiler. From Joy and Factor, it uses quotations (anonymous, nestable functions) and combinators (functions that operate on functions) for much of the stack and flow control. It also adds vocabularies for working with strings, arrays, and other data types.
>
> Retro runs on Nga, a tiny virtual machine emulating a MISC style processor. Implementations of this are included in 65c816 Assembly, C, C++, C#, JavaScript, Nim, Pascal, Python, Swift, and Retro.“

- Retro Forth via Websocketd `https://github.com/RickCarlino/retroforth-web-terminal`
- Websocketd `http://websocketd.com/`
- Retro Forth `https://retroforth.org/`

mk

Fortsetzung der Leserbriefe auf Seite 14

[7] cef : character encoding form

[8] character encoding scheme, ces; little– oder big–endian

[9] Ich nehme mal wieder Chinesisch, weil mir das so weit weg von unseren Buchstaben vorkommt.

[10] Das *xchar wordset* handhabt diese erweiterten Codeformen.

[11] Websocketd ist ein WebSocket daemon in Linux

TheStack — stapelbare Platinen

Erich Wälde

Der erste Teil dieses Artikels[1] *beschreibt meine neuste Errungenschaft: ganz forthsch stapelbare Platinen für eigene Bastelprojekte. Dabei wird der bekannte ATmega644PA–Kontroller verwendet, mit allerlei praktischen Ergänzungen.*

Im zweiten Teil stelle ich dann eine Konfiguration von `AmForth` *vor und Beispielprogramme, um auf dem ausgesuchten 20x4–Buchstabendisplay Text anzuzeigen.*

Geschichte

Vor langer Zeit (2009–07–25 vielleicht) traf sich ein Rudel Forth–Kundiger bei Martin Bitter im Wohnzimmer. Es ging darum, eine Platine zu entwerfen, mit der man Anfänger bespaßen könnte, und die auch für eigene Experimente/Projekte taugen sollte. Die Vorstellungen gingen von „eine nackte Adapterplatine“ bis hin zu „ein komfortables Ding“, bevorzugt mit nur einem Kabel — unvereinbar. Die Geschichte hat uns dann locker und mit links überholt, als die Arduino–Platinen in größeren Stückzahlen auf den Markt kamen. Dass deren Bootloader nicht direkt mit Am– oder einem anderen Forth zusammenarbeiten wollte, ist eine ganz andere Geschichte.

Auf der Forth–Tagung 2019 in Worms habe ich erste Entwürfe dieses Projekts vorgestellt. In einer Abendveranstaltung habe ich der Audienz erste Schritte in KiCAD [5] vorgeführt. Es gibt davon sogar Videos, siehe [17]. Dass ich jetzt dem unübersichtlichen Platinen–Universum eine weitere, mit nichts kompatible Dimension hinzugefügt habe, geht ganz klar auf das Konto: „Because I can!“

Teil I

Vorrede

TheStack [1] stellt eine Mikrokontroller–Platine vor, die einige meiner Wünsche vereint, auch wenn das Ergebnis vielleicht etwas ungewöhnlich ausfällt.

Es gibt Leute, die gerne mit dem Lötkolben arbeiten und dabei wunderbare Dinge (z. B. [3]) erschaffen, die aber SMD–Bauteile nicht anfassen. „Zu klein!“, höre ich da. Eine Lupe, gute Beleuchtung, Pinzette und ein Bauteiletester sind da sehr hilfreich. Weniger hilfreich ist ein Teppichboden und Nies–Anfälle. SMD–Bauteile haben auch Vorteile — zumindest verbrauchen sie nicht so viel Platz in der Bastelkiste. Und manche Bauteile sind nur in SMD–Bauformen lieferbar. Und Reflow–Löten mit einem ausgedienten Bügeleisen [16] hat einen ordentlichen Nerdfaktor.

Vielfalt

TheStack ist eine SMD–Löt–Übung mit einer gewissen Komplexität — und nicht nur ein Blinki, das dann in der Schublade verstaubt. TheStack bietet eine Kontroller–Platine mit einem der bekannten AVR–Kontroller, genauer: ein ATmega644PA [4]. Das ist der größere Bruder vom ATmega328A auf den berühmten Arduino–Platinen.

Der Name des Projektes TheStack bezieht sich zum einen darauf, dass sich die Platinen stapeln lassen sollen. Zum anderen ist es eine Anspielung auf die wichtigste Einheit eines Forth–Systems, den Daten–Stapel — meine bevorzugte Programmiersprache für diese Kontroller ist `AmForth` [12].

TheStack macht wenig konkrete Vorgaben. Mikrokontroller–Projekte gibt es viele, allerdings muss sich der Erfinder immer irgendwie festlegen, welche Pins für welche Aufgabe belegt werden. Dabei gibt es immer Kompromisse. Daher habe ich beschlossen, die Anschlussbelegung für jegliche Peripherie eben nicht vorzugeben. Sogar die Status–LEDs müssen mit Litze oder Fädeldraht mit einem Pin verbunden werden. Damit bleibe ich flexibel, falls eine Platine Überarbeitung erfährt oder zusätzliche Aufgaben übernehmen soll.

TheStack steht unter der CERN OpenHardwareLizenz [6]. Die KiCAD–Dateien sind verfügbar. Es lassen sich Änderungen an der Schaltung und am Layout vornehmen. Man kann die Platinen selbst fertigen oder fertigen lassen (total schick mit Lötstopplack und Bestückungsaufdruck).

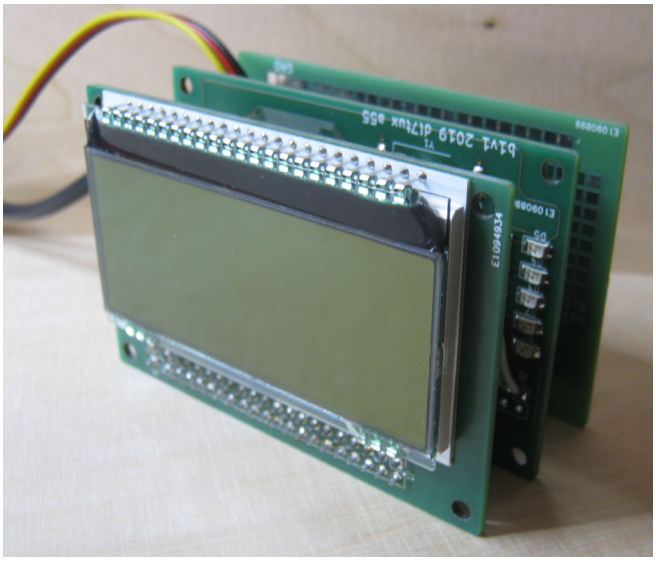


Abbildung 1: TheStack

[1] Dieser Teil wurde für eine andere Publikation geschrieben, wurde aber verworfen. Allerdings ist das zu schade für die Tonne.

TheStack ist auch eine Plattform, um Programmieren zu lernen. Denn so ein Mikrokontroller ist unnütz, solange er nicht programmiert wurde. Programmieren ist eine komplett andere Angelegenheit, die zwar ohne Lötkolben auskommt, aber nicht ohne einen Rechner, Programme und Programmieradapter.

Im einfachsten Fall bekommt man zu einem Projekt einen fertig programmierten Kontroller. Dann kann man dem Ding aber keine neuen Kunststücke beibringen! Und das ist doch der große Spaß an der Sache. So ein Kontroller kann alles, was man ihm beibringt! Also nur Mut und frisch ans Werk. Vielleicht findet sich auch jemand, der oder die für ihr Leben gern programmiert! Die Kosten für den Programmieradapter sind den Spaß jedenfalls wert.

Dieser Teil der Welt ist mindestens so groß und vielfältig, wie die Elektronik selbst. Es gibt verschiedene Programmiersprachen für Mikrokontroller: Assembler[10], C[11], Forth[12], Bascom[14], Lisp[13], Ada[15] und mehr. TheStack ist also auch eine Plattform für vielfältige Programmier–Experimente.

Design–Entscheidungen

Ich habe lange über dem Formfaktor gebrütet. Als wesentliche Inspiration darf das Parallax–Quickstart–Projekt [9] gelten, aber auch das miniSKS von DL1ZAX [8] und die AATiS BB56–Platinen [7].

Die Platinen sind 2 x 3 Zoll groß (ca. 51 x 76 mm) und haben zwei Lochreihen mit je 2 x 20 Kontakten. In die eine werden PC104–Press–Fit–Stapelstifte gepresst. An der anderen sind sämtliche Pins des Kontrollers zugänglich. Sämtliche Peripherie muss explizit verbunden werden. Die SMD–Bauteile sind von der Größe mindestens 0805, besser 1206. Die Lötinseln sind alle so groß, dass man sie von Hand löten kann. Die Platinen sind zweilagig und nur auf einer Seite bestückt (Ausnahme: Display).

Um sinnvoll arbeiten zu können, entschied ich mich, zum Start drei Platinen zu erstellen: eine Kontroller–Platine, eine Platine für ein Display, sowie eine Platine mit Lötinseln inklusive zweier SOT–Bestückungsplätze.

Platine 1: Kontroller

Die Schaltung stellt die notwendigen Bauteile wie Entkoppelkondensatoren, Beschaltung des Reset–Pins, sowie einen Quarz mit Lastkondensatoren zur Verfügung. Zusätzlich gibt es einen Bestückungsplatz für einen Uhrenquarz. Dieser ist hilfreich, will man den Kontroller die meiste Zeit schlafen legen, was eine Menge Strom sparen kann. Die Platine stellt außerdem Platz für vier Status–LEDs zur Verfügung, für eine weitere LED, welche die Stromversorgung anzeigt, Zugang zu einer seriellen Schnittstelle mit Pins sowie den Programmierstecker. Und weil ich nicht widerstehen konnte, fügte ich einen Bestückungsplatz für den Uhrenbaustein DS3231 hinzu.

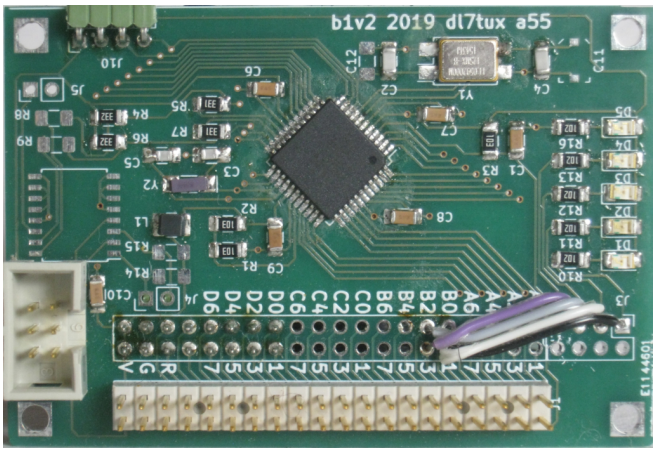


Abbildung 2: Bestückte Kontrollerplatine. Die Litzen verbinden die Status–LEDs mit den Kontroller–Pins B0 .. B3.

Platine 2: DOG–EAM–Display

Displays gibt es viele, Wünsche und Vorlieben ebenfalls. Ich habe mich für ein Buchstabendisplay mit vier Zeilen zu je 20 Zeichen entschieden. Die Hintergrundbeleuchtung ist separat zu beschaffen und in verschiedenen Farben erhältlich. Das Display kann man auf verschiedene Weise ansprechen: mit 8– oder 4–Bit parallelen Daten, per SPI– oder I2C–Bus. Allerdings begrenzt das gewählte Display die Versorgungsspannung auf 3,3 Volt! Außerdem musste ich lernen, dass eine schaltbare Reset–Verbindung zwar einen weiteren Pin am Kontroller beansprucht, aber deutlich zum Gelingen beiträgt.

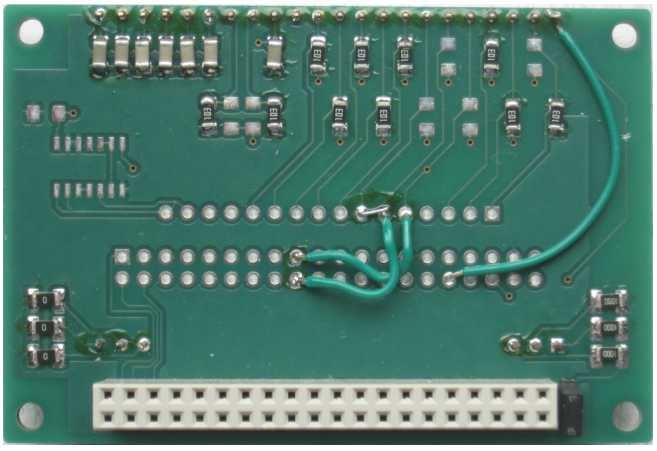


Abbildung 3: Bestückte Display–Platine, I2C–Variante. Version 1, die Reset–Verbindung ist noch geflickt. Das Display selbst befindet sich auf der Rückseite.

Platine 3: Prototyping–Fläche

Diese Platine hat mich eine Menge gelehrt, was ihre Erstellung in KiCAD betrifft. Jede Insel ist ein Bauteil mit einem Pin. Diesem Pin habe ich eine von zwei selbst erstellten Lötinseln zugewiesen. Zum einen lediglich die Insel auf der Vorderseite, zum anderen je eine Insel auf Vorder– und Rückseite, verbunden mit einer durchkontaktierten Bohrung. Diese sind dafür gedacht, Kabel, Pinleisten oder kleine Klemmblöcke anzulöten.

Auf der Rückseite ist eine durchgehende Masseflächе vorhanden, im Bereich der Lötinseln ohne Lötstopplack. Dann kann man bei Bedarf eine Insel durchbohren und auf kurzem Weg mit Masse verbinden. Zwei SOT–Bestückungsplätze habe ich für den Komfort spendiert, ebenso zwei Reihen Inseln, die eine mit Masse, die andere mit der Versorgungsspannung verbunden. In diesem Layout steckt also mehr Arbeit, als man auf den ersten Blick vermuten könnte.

Es sind natürlich viele weitere Platinen denkbar, etwa für Kommunikation (von RS485 bis LoRa), Sensorik oder Stromversorgung (Solarpanel, Akku und Ladekontroller). Das hängt vom konkreten Projekt ab und kann mit Hilfe der Vorlage (separates KiCAD–Projekt) realisiert werden.

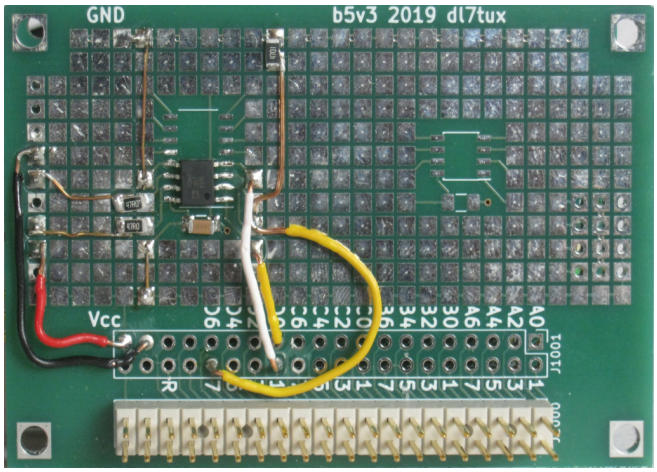


Abbildung 4: Bestückte Prototype–Platine. Ein RS485–Transceiver. Die Klemmblöcke sind am linken Rand auf der Rückseite. Darüber wird der Kontroller auch mit Spannung versorgt.

Zukunft

Ich selbst habe daraus zunächst eine Stationsuhr gebaut, um zu sehen, ob die beiden Platinen auch zusammenspielen. Experimente mit einer Neuauflage meiner Hausbus–Stationen sind noch im Gange. Derzeit vielversprechendster Kandidat ist eine autarke *Außenstation* mit Solarpanel, Laderegler, Batterie und einer Reihe Sensoren. Die Messwerte werden alle 64 Sekunden mit kleiner Leistung in den Äther geflüstert. Eine entsprechende Empfangsstation sammelt die Daten ein und leitet sie weiter.

Was genau man aus diesen Platinen macht, ist jedem selbst überlassen. Der Fantasie sind letztlich keine Grenzen gesetzt. Vom COVID19–Abstandsmesser über Taschen–Geigerzähler bis zum IoT–Sensorknoten, von Nachbarschaftswetterstation bis Citizen–Science–Projekt — alles ist möglich.

Teil II

Mein `AmForth`–System erzeuge ich immer selbst aus Quellen. Ich kopiere das Verzeichnis `appl/template`, ändere die Dateinamen und den Inhalt der Datei `Makefile`, und dann kann's losgehen.

`AmForth`–Konfiguration

Die wichtigsten Einträge in der Datei `main.asm` sind schnell erklärt:

```
.equ F_CPU = 11059200
.set BAUD=115200
.include "drivers/usart_0.asm"

.set WANT_IGNORECASE = 0
```

Die ersten drei Zeilen teilen dem System die Quarzfrequenz und die gewünschte Baud–Rate auf der seriellen Schnittstelle `usart0` mit. Ich bevorzuge Baudratenquarze mit *krummen* Frequenzen, damit die Geschwindigkeit der seriellen Schnittstelle höher gewählt werden kann.

Der letzte Eintrag ist Geschmacksache: Ich bevorzuge die Unterscheidung von Groß– und Kleinbuchstaben.

Es gibt ein paar Funktionen, die ich meinem `AmForth`–System fast immer hinzufüge. Zum einen über die Datei `dict_appl.inc`:

```
.include "words/notequalzero.asm"
.include "words/no-jtag.asm"
.include "words/2spirw.asm"
.include "words/spirw.asm"
.include "words/n-spi.asm"
```

Und zum anderen als `include`–Anweisungen auf Forth–Ebene:

```
include lib-common/builds.frt
include lib-common/forth2012/core/erase.frt
include lib-common/dot-base.frt
include lib-avr8/imove.frt
include lib-avr8/bitnames.frt
include lib-avr8/forth2012/core-ext/marker.frt
include lib-avr8/forth2012/core/environment-q.frt
include lib-avr8/dot-res.frt
include lib-avr8/forth2012/core/avr-values.frt
include lib-common/forth2012/core/is.frt
include lib-common/forth2012/tools/dumper.frt
include lib-avr8/hardware/interrupts.frt
include atmega644p.fs
include lib-common/forth2012/double/2-fetch.frt
include lib-common/forth2012/double/2-store.frt
include lib-common/quotations.frt
include lib-common/uzerodotr.frt
```

Dabei ist `lib-common` ein symbolischer Link auf das Verzeichnis `common/lib` im Verzeichnisbaum des verwendeten `AmForth`. Sinngemäß zeigt `lib-avr8` auf das Verzeichnis `avr8/lib`. Die Datei `atmega644p.fs` ist ein Extrakt aus `avr8/devices/atmega644p/atmega644p.frt`. Darin sind nur die Definitionen, die ich auch wirklich haben will. Der Grund ist eher die Zeit, die es braucht, alles über die serielle Schnittstelle zu programmieren, als angestrengter Minimalismus. Zusammen ergibt das ein relativ komfortables System, das dann je nach Projekt weiter ergänzt wird.

Display

Jetzt habe ich mich quasi aus dem Fenster gelehnt und behauptet, es sei doch eine praktische Sache, wenn man die Displays auf verschiedene Arten ansteuern kann. Ich habe sogar extra das Platinenlayout so gemacht, dass man alle diese Fälle auch verdrahten kann. Dann sollte ich auch demonstrieren, dass das wirklich funktioniert, oder? Am besten als Beschreibung der nötigen Komponenten (das sind auch Dateien), deren Inhalte hübsch übereinandergestapelt werden.

Ebene 0: Pins und Register

Die für das konkrete Projekt gültige Pinbelegung wird in der Datei des Hauptprogramms definiert. Die Namen stammen allerdings aus Ebene 1. Das sieht für ein per I2C oder SPI angeschlossenes Display etwa so aus:

```
include lib-avr8/bitnames.frt
\ ssd1803a display i2c-mode
PORTC  0 portpin: _scl
PORTC  1 portpin: _sda

\ ssd1803a display spi-mode
PORTA  0 portpin: ssd1803a-/cs    \ chip select
PORTD #7 portpin: ssd1803a-/reset
PORTB #4 portpin: /ss
PORTB #5 portpin: _mosi
PORTB #6 portpin: _miso
PORTB #7 portpin: _clk
```

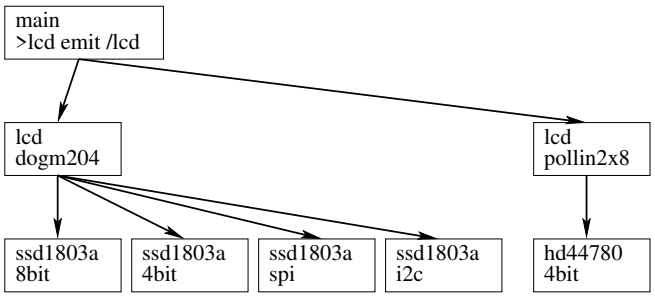


Abbildung 5: Komponenten zur Ansteuerung der Displays

Ebene 1: Befehle und Daten

Egal welche Ansteuerung oder Pinbelegung nun ausgewählt wurde, man muss immer einen Weg finden, dem Displaykontroller zu sagen, was er anzeigen soll. Die beiden Kontroller, die ich in Gebrauch habe, unterscheiden also zwischen Befehlen und Daten. Auf dieser untersten Ebene braucht es mindestens folgende Funktionen (`$ctrl` steht für `ssd1803a` oder `hd44780`):

- `$ctrl-init` — konfiguriere die benutzten Ein–/Ausgabe–Pins am Mikrokontroller, setze den /Reset–Pin für in diesem Fall 5 Millisekunden, um den Displaykontroller zu starten.
- `$ctrl-cmd-tx` — übertrage ein Byte im Befehls–Modus.
- `$ctrl-data-tx` — übertrage ein Byte im Daten–Modus.

Jegliche Hilfsworte, um die Datenübertragung auszuführen, sind ebenfalls in dieser Ebene zu implementieren, etwa das Zerlegen der Datenbytes in Nibble, oder die korrekte Ausführung *einer* Übertragung.

Dieser Teil muss getrennt für alle Displaykontroller (s. o.) und Ansteuerungen (8–Bit, 4–Bit, I2C, SPI) implementiert werden. Die nächsthöhere Ebene benutzt dann nur noch diese Schnittstelle, ohne über irgendwelche Details Bescheid wissen zu müssen.

Ebene 2: Das Display

Die Aufgaben in dieser Ebene beziehen sich immer auf ein bestimmtes Display. Die Größe der Anzeige in Zeilen+Spalten ist hier vorgegeben. Die konkrete Anbindung wird aus Ebene 1 ausgewählt. Die Schnittstelle auf dieser Ebene beinhaltet ungefähr diese Funktionen:

- `+lcd` — das Display einschalten
- `lcd-cmd` — einen Befehl senden
- `lcd-emit` — ein Zeichen (Datenbyte) senden und anzeigen
- `lcd-clear` — den aktuellen Anzeigeninhalt löschen
- `lcd-home` — den Cursor an die erste Position setzen
- `lcd-pos` — den Cursor an eine gewünschte Position setzen
- `lcd-cursor-{blink,on,off}` — den Cursor anzeigen, oder auch nicht
- `lcd-display-{on,off}` — ggf. kann man das Display komplett aus– und wieder einschalten
- `lcd-orient-{up,down}` — ggf. kann man den Text auf der Anzeige auch *einfach so* mal um 180° drehen.
- `lcd-contrast` — ggf. kann man den Kontrast der Anzeige per Befehl verändern (nicht per Beschaltung)

Das sind Funktionen, die das Leben in den höheren Ebenen einfacher machen. In diese Ebene gehören auch Funktionen, die nicht bei allen Displays gleichermaßen verfügbar sind, etwa das Drehen des Inhalts auf der Anzeige.

Dieser Teil muss getrennt für verschiedene Displays implementiert werden.

Ebene 3: `emit` umschalten

Die in Ebene 2 definierte Funktion `lcd-emit` wird natürlich über den normalen `emit` aufgerufen, welcher bekanntlich als *deferred word* implementiert wird. Es braucht also noch die Möglichkeit, zwischen den verschiedenen Ausgabemöglichkeiten umzuschalten:

- `>lcd` — emit auf das LCDisplay
- `/lcd` — emit auf die Voreinstellung, z. B. serielle Schnittstelle

Solchermaßen ausgestattet sollte es kein Problem mehr sein, die Anzeige für ein neues Projekt zu wechseln. In den Listings ist ein kurzgehaltenes Beispiel abgedruckt, welches die Variante mit der I2C–Anbindung demonstriert.

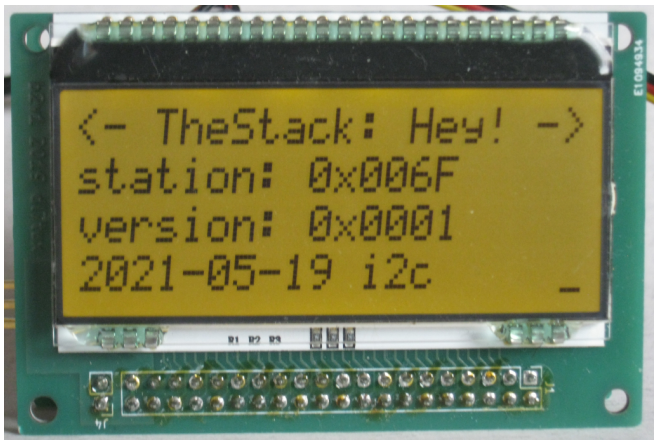


Abbildung 6: Anzeige des Demo–Programms

Ausblick und so Sachen

Derzeit sind auf meinem Schreib–/Basteltisch mehrere dieser Platinen in Gebrauch. Eine Station ist die Vorstufe für einen Außensensor, der die Temperatur und Luftfeuchte misst und diese Messwerte alle 64 Sekunden als Funkwellen (434 MHz) in die Gegend flüstert. Weitere Sensoren sind geplant. Eine zweite Station nimmt diese Messungen auf, stellt sie einem Perl–Programm auf einem Raspberry Pi zur Verfügung, welches die Daten alle 10 Minuten in die hauseigene Datenbank schickt. Per Grafana und Webbrowser kann man die Daten dann schick und bunt ansehen. Eine dritte Station nimmt die Daten ebenfalls auf, zeigt diese dann aber auf einem Display an. Das wäre die Empfangsstation für die Küche, quasi. Zwei weitere Stationen werden mit wechselnden Aufgaben beschäftigt. Ich kann also sagen, dass die Platinen für mich sehr praktisch sind. Von diesen Experimenten wird noch zu berichten sein, der Editor scharrt schon ungeduldig mit den Hufen.

Den kompletten Quelltext abzudrucken, schreckt mich etwas. Quelltext–Lesungen sind nur in kleinen Kreisen populär. Und abtippen will das sowieso keiner. Daher lege ich den kompletten Quelltext auf meinem öffentlichen Git–Repository [2] ab.

Von den hier vorgestellten Platinen habe ich noch einige im Schrank, auch Komponenten sind noch teilweise vorhanden. Bei Interesse gebe ich gerne welche ab. Kontakt: `erich.waelde@forth-ev.de`

Referenzen

1. TheStack (Schaltpläne, Layout) — `https://git.sr.ht/~ew/TheStack`
2. TheStack (Quelltext) — `https://git.sr.ht/~ew/TheStack-Code`
3. “The Clock” von Gislain Benoit — `http://www.techno-logic-art.com/clock.htm`
4. Datenblatt ATmega644PA — `https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega164A_PA-324A_PA-644A_PA-1284_P_Data-Sheet-40002070B.pdf`
5. KiCAD — `http://kicad-pcb.org/`
6. CERN Open Hardware License — `http://ohwr.org/cernoh`
7. AATiS BB56 — `https://www.aatis.de/`
8. SKS von DL1ZAX — `http://dl1zax.selfhost.de/ATMega/index.html`
9. Quickstart Propeller — `https://www.parallax.com/product/40000`
10. avr–asm — `http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf`
11. avr–gcc (z. B. Debian Linux Paket)— `https://packages.debian.org/buster/gcc-avr`
12. AmForth — `http://amforth.sourceforge.net`
13. uLisp — `http://www.ulisp.com/`
14. BasCom (proprietär) — `https://avrhelp.mcselec.com`
15. Ada — `https://wiki.kucia.net/doku.php?id=projects:avrada`
16. Reflow–Bügeleisen — `https://github.com/MakeMagazinDE/Loeten-mit-dem-Buegeleisen`
17. Wikiseite der Forthtagung 2019 in Worms — `https://forth-ev.de/wiki/events:ft2019:start`

Listings

Der gesamte Quelltext ist natürlich wieder mal ziemlich lang, daher ist hier nur ein Auszug abgedruckt. Der vollständige Quelltext befindet sich auf meiner Projektseite [2] zum Herunterladen.

Ebene 1

```
\ 2019-03-20 ew ewlib/ssd1803a-i2c.fs
\
\ driver for EA-DOGM 204 Display with
\ ssd1803a controller
\
\ requires
\     avr8/lib/hardware/i2c-twi-master.frt
\     common/lib/hardware/i2c.frt
\ needs
\     PORTC 1 portpin: _scl
\     PORTC 2 portpin: _sda
\
\     PORTD 7 portpin: ssd1803a-/reset
\ provides
\     ssd1803a-init     ( -- )
\     ssd1803a-cmd-tx   ( c -- )
\     ssd1803a-data-tx  ( c -- )

```

```
                                                  \     include ewlib/ssd1803a-4bit.fs
                                                  \     include ewlib/ssd1803a-spi.fs
                                                  \     include ewlib/ssd1803a-i2c.fs

\ I2C mode handles Transfers of 1 data/cmd        \ provides
\ Byte through the following sequence             \     +lcd -- configure pins, communication
\ - start condition                               \     lcd-init -- set all details,
\ - slave addr | R/W                              \                 display on, cursor blink on
\ - control byte: 0x80 (Co, continuation bit)     \                 clear content
\                 0x40 (D/C#) 1: data, 0: cmd     \     lcd-emit
\ - data byte                                     \     lcd-pos
\                                                 \     lcd-cmd
\ The sequence control byte, data byte may be     \     lcd-clear
\ repeated if Co=1                                \     lcd-home
\ There maybe more than 1 data byte, after Co=0   \     lcd-contrast ( 0..63 -- ) set contrast
\                                                 \     lcd-rompage-{A,B,C}  select glyph table
\ - stop condition                                \     lcd-orient-{up,down} rotate text

$3c constant lcd_addr                             \ 8-bit spi with chip select
$80 constant ssd1803a-cb-cont                     : +lcd     ( -- )    ssd1803a-init ;
$40 constant ssd1803a-cb-data                     : lcd-emit ( n -- )
                                                    $ff and  ssd1803a-data-tx ;
\ i2c mode is selected by IM1=low, IM2=low        : lcd-cmd  ( n -- )
\ pullup/down resistors on board!                   $ff and  ssd1803a-cmd-tx ;
\ configure pins, reset controller,               : lcd-init
: ssd1803a-init                                     ssd1803a-set.RE=1.IS=0 \ --- RE=1 IS=0
  ssd1803a-/reset  pin_output                       $09 lcd-cmd            \ 4 lines
  ssd1803a-/reset  high #5 ms  \ idle               $06 lcd-cmd            \ orient-up
  ssd1803a-/reset  low  #5 ms                       $1E lcd-cmd            \ BS1=1
  ssd1803a-/reset  high #5 ms
;                                                   ssd1803a-set.RE=0.IS=1 \ --- RE=0 IS=1
                                                    $1B lcd-cmd    \ BS0=1 -> Bias=1/6
                                                    $6E lcd-cmd    \ Divider on and set value
: ssd1803a-cmd-tx   ( n -- )                        $57 lcd-cmd    \ Booster on, contrast C5:4
  ( n )             \ data byte                     $72 lcd-cmd    \ set contrast C3:0
  $00               \ control byte
  #2                \ N                             ssd1803a-set.RE=0.IS=0 \ --- RE=0 IS=0
  lcd_addr          \ addr (7bit)                   $0F lcd-cmd    \ display,cursor,blink on
  i2c.n!                                          ;
;                                                 : lcd-clear        ( -- )  $01 lcd-cmd ;
: ssd1803a-data-tx  ( n -- )                      : lcd-home         ( -- )  $02 lcd-cmd ;
  ( n )             \ data byte                   : lcd-cursor-blink ( -- )  $0F lcd-cmd ;
  ssd1803a-cb-data  \ control byte                : lcd-cursor-on    ( -- )  $0E lcd-cmd ;
  #2                \ N                           : lcd-cursor-off   ( -- )  $0C lcd-cmd ;
  lcd_addr          \ addr (7bit)                 : lcd-display-on   ( -- )  $0C lcd-cmd ;
  i2c.n!                                          : lcd-display-off  ( -- )  $08 lcd-cmd ;
;                                                 : lcd-pos ( row col -- )
                                                    \ $80 + 7bit addr register
\ this is "8bit" specific,                          \ $80 + row*$20 + col
\ "4bit" would be $2A instead of $eA                $1F and \ trim col
: ssd1803a-set.RE=1.IS=0 ( -- )                     swap
  $3A ssd1803a-cmd-tx                               $03 and \ trim row
;                                                   $20 * +
: ssd1803a-set.RE=0.IS=1 ( -- )                     $80 + lcd-cmd
  $39 ssd1803a-cmd-tx                             ;
;                                                 : lcd-contrast    ( n -- )
: ssd1803a-set.RE=0.IS=0 ( -- )                     $3F and dup \ limit to 0..63
  $38 ssd1803a-cmd-tx                               ssd1803a-set.RE=0.IS=1 \ --- RE=0 IS=1
;                                                   #4 rshift $54 or lcd-cmd
                                                    $0f and   $70 or lcd-cmd
                                                    ssd1803a-set.RE=0.IS=0 \ --- RE=0 IS=0
                                                  ;
                                                  : lcd-(rompage)   ( n -- )
                                                    $0C and \ limit to %000_1100

                                                    ssd1803a-set.RE=1.IS=0 \ --- RE=1 IS=0
```

Ebene 2

```
\ 2019-03-16  ew  ewlib/lcd_dogm204.fs

\ this file requires one of the low-level
\ libraries to access the DOGM204 Display.
\ Load one of them prior to this file
\ needs one of
```

```
    $72   lcd-cmd
    ( n ) lcd-emit
    ssd1803a-set.RE=0.IS=0 \ --- RE=0 IS=0
  ;
  : lcd-rompage-A   ( -- )  $00 lcd-(rompage) ;
  : lcd-rompage-B   ( -- )  $04 lcd-(rompage) ;
  : lcd-rompage-C   ( -- )  $08 lcd-(rompage) ;

  : lcd-(updown)    ( n -- )
    ssd1803a-set.RE=1.IS=0 \ --- RE=1 IS=0
    ( n ) lcd-cmd
    ssd1803a-set.RE=0.IS=0 \ --- RE=0 IS=0
  ;
  : lcd-orient-up   ( -- ) $06 lcd-(updown) ;
  : lcd-orient-down ( -- ) $05 lcd-(updown) ;
```

Demo

```
\ 2021-05-19 ew
\ Station $6F - test i2c display connection

marker --main--

variable stationID
$006F Evalue EEstationID
$0001 Evalue swVersion
variable old-emit

include lib-common/forth2012/double/2-fetch.frt
include lib-common/forth2012/double/2-store.frt
include lib-common/quotations.frt

\ --- pins ----------------------------------

PORTA  1 portpin: led.1
PORTA #2 portpin: led.2
PORTA #3 portpin: led.3
PORTA #4 portpin: led.0
: lederr     [: led.0 ;] execute ;
: ledsec     [: led.1 ;] execute ;
: ledsensor  [: led.2 ;] execute ;
: leddata    [: led.3 ;] execute ;


PORTC 0 portpin: _scl
PORTC 1 portpin: _sda

PORTD #7 portpin: ssd1803a-/reset

: pin_highZ ( pinmask portaddr -- )
  over over pin_input low ;

\ --- famous includes -----------------------
include lib-common/uzerodotr.frt
include lib-ew/stationid.fs
include lib-common/forth2012/facility/ms.frt
include lib-ew/leds.fs

\ --- display -------------------------------

\ EA DOGM204 via i2c
include lib-avr8/hardware/i2c-twi-master.frt
include lib-common/hardware/i2c.frt
include lib-common/hardware/i2c-detect.frt
: +i2c  ( -- )
  _scl pin_pullup_on
  _sda pin_pullup_on
  0  \ prescaler
  #6 \ bit rate --- 400kHz @ 11.0592 MHz
  i2c.init
;

: i2c.scan
  base @ hex
  $79 $7 do
    i i2c.ping? if i 3 .r then
  loop
  base !
  cr
;

include lib-ew/ssd1803a-i2c.fs
include lib-ew/lcd_dogm204.fs
: >lcd
  ['] lcd-emit to emit
;
: /lcd
  old-emit @ is emit
;



\ --- main ----------------------------------

: heyhey ." <- TheStack: Hey! ->" ;
: lcd-msg
  lcd-clear lcd-home  >lcd
  heyhey
  #1 0 lcd-pos
  ." station: 0x" stationID @ #4 hex u0.r
  #2 0 lcd-pos
  ." version: 0x" swVersion #4 hex u0.r
  #3 0 lcd-pos
  ." 2021-05-19  i2c"
  /lcd
  #3 #19 lcd-pos
;

: init
  ['] emit defer@ old-emit !
  +stationid
  +leds leds-intro

  +i2c
  +lcd lcd-init lcd-clear
  #50 lcd-contrast
  lcd-msg
;
```

Pin–Belegung

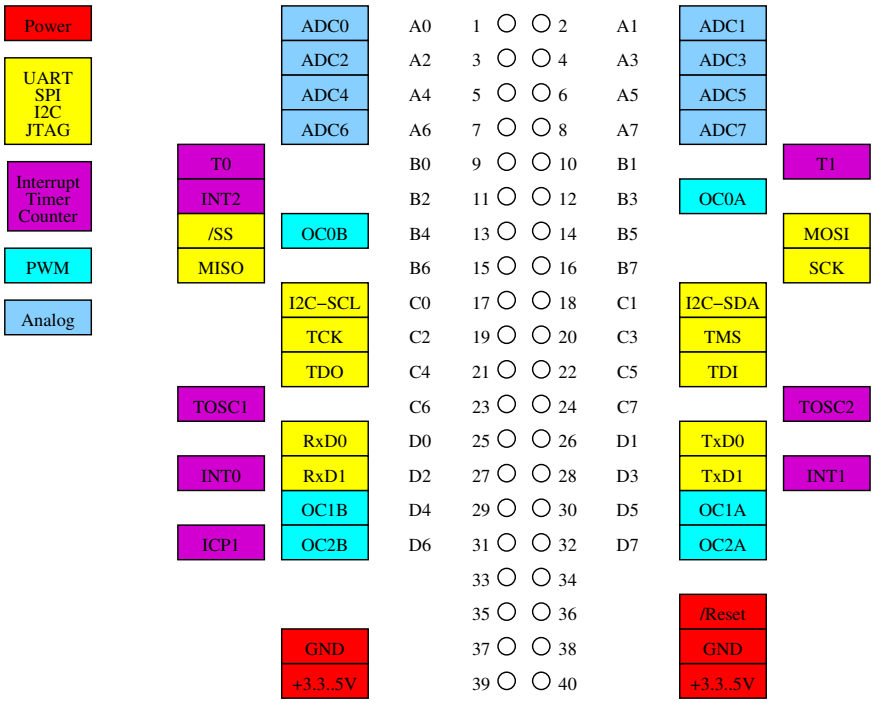


Abbildung 7: Pinbelegung der Stecker J1 und J2 auf den Platinen. Die Portpins des Kontrollers sind alle herausgeführt. Dazu ein Auszug der Funktionen aus dem Datenblatt des verwendeten Atmel ATmega644PA. Die Pins 33 bis 35 sind nicht belegt und können für zusätzliche Verbindungen zwischen den Platinen verwendet werden.

Fortsetzung der Leserbriefe von Seite 6

MiniForth — noch ein Forth im Bootsektor

MiniForth von Jakub Kądziołka ist ein minimales Forth für x86 PCs in nur 504 Byte. Damit passt dieses Forth in den Bootsektor einer Festplatte oder eines USB–Sticks. Trotz der geringen Größe ist MiniForth benutzbar. In den 504 Byte sind die Wörter `+ - ! @ c! c@ dup drop swap emit u. >r r> [ ] : ; load` definiert. Per `load ( blk - )` können Forth–Quellcode–Blöcke zu je 1024 Byte von Festplatte oder USB–Laufwerk geladen werden. Über diese Quellcode–Blöcke kann aus dem MiniForth ein ausgewachsenes Forth erzeugt werden.

In dem zugehörigen ausführlichen Blog–Posting erklärt Jakub die Design–Entscheidungen innerhalb von MiniForth und erklärt den Assembler–Quellcode. Der Prozessor–Stack wird für den Forth–Daten–Stack verwendet, der Return–Stack per Software implementiert — nicht schnell, aber dies spart Speicherplatz. Das `NEXT` der Forth–Wörter wurde komprimiert und wird nach dem Laden im Speicher entpackt. Fortgeschrittenes Code–Golf!

Aus dem Assembler–Quellcode wird mittels des YASM–Assemblers der Boot–Sektor gebaut. Dieser wird mit den Quellcode–Blöcken zu einem Disketten–Abbild verbunden und kann sektorweise auf einen USB–Stick oder auf eine Festplatte geschrieben werden (Achtung: Dabei werden alle Daten auf dem Speichermedium gelöscht). Gefahrlos kann MiniForth in einem Emulator getestet werden. Im Quellcode–Repository kann der QEMU–Emulator mit dem Script `run.sh` mit dem erstellten Disketten–Abbild gestartet werden.

MiniForth kommt ohne (Fehler–)Meldungen aus (dafür ist kein Platz) und startet mit einem schwarzen Bildschirm mit `BASE` 16 (HEX). Das abgedruckte Test–Programm kann helfen, die eigene MiniForth–Installation zu prüfen.

MiniForth ist ein Highlight für Freunde von hoch–optimierten Assembler–Forth Systemen.

1. MiniForth `https://github.com/NieDzejkob/miniforth`
2. Blog–Post “Fitting a FORTH in 512 bytes“ `https://niedzejkob.p4.team/bootstrap/miniforth/`

Kleines MiniForth Test–Programm

```
>r >r dup a swap ! r> r>     : cr 13 emit 10 emit ;
: say-hi 72 emit 101 emit 108 dup emit emit 111 emit  44 emit 32 emit
                119 emit 111 emit 114 emit 108 emit 100 emit 33 emit cr ; say-hi
```

FemtoRV32–Quark — ein RISC–V in 400 Zeilen Verilog

Matthias Koch und Bruno Levy

Aus einer ganz besonderen Zusammenarbeit mit Bruno Levy*, in der wir fast ein Jahr lang miteinander nachgedacht haben, ist ein neuer RISC–V–Prozessor hervorgegangen. Winzigklein, das war das Ziel, um damit den Icestick zu erobern. Das Ziel haben wir erreicht — aber noch etwas anderes: einen Prozessor zu entwickeln, dessen Quelltext gut lesbar und dessen Funktionsweise gut verständlich ist. Von mir stammen vor allem viele Ideen zur kompakten Implementierung, während Bruno sich die Lesbarkeit seines nach dem Elementarteilchen benannten „FemtoRV32–Quark" auf die Fahnen geschrieben hat.*

Außen herum

Nach außen hin ist die Beschaltung übersichtlich, neben Takt und Reset sind die nötigen Leitungen für Lese– und Schreibzugriffe vorhanden. Für einen Lesezugriff legt der Prozessor die Adresse an `mem_addr`, signalisiert den gewünschten Lesezugriff für einen Taktzyklus lang in `mem_rstrb` und liest die entsprechenden Daten im darauffolgenden Taktzyklus aus `mem_rdata`, sofern nicht vorher auf `mem_rbusy` gewartet werden muss. Für einen Schreibzugriff legt der Prozessor die Daten an `mem_wdata` bereit, setzt für einen Taktzyklus lang die Schreibmaske `mem_wmask` auf die gewünschte Schreibbreite und prüft anschließend, ob auf `mem_wbusy` gewartet werden muss. Das war es schon!

Und innen drin

Innen drin lassen sich die klassischen Bestandteile eines Prozessors fein säuberlich sortiert wiederfinden. Lese– und Schreibzugriffe benötigen, wenn nicht auf etwas drumherum gewartet werden muss, vier Taktzyklen. Fast alle anderen Befehle werden in drei Taktzyklen ausgeführt — mit Ausnahme der Schübe, die aus Platzgründen in der Logik jeweils nur um ein Bit pro Taktzyklus schieben können und so eine variable Zahl von Taktzyklen benötigen.

Einen Preis hat die Einfachheit jedoch: Mit ungültigen Befehlen jenseits des implementierten Befehlssatzes RV32I gefüttert wird dieser Prozessor auch Unsinn anstellen — ganz wie die Klassiker *Z80* oder *6502*, mit denen einige von uns groß geworden sind. Da es keine Interrupts gibt, liefern zum Beispiel alle möglichen Systembefehle den Taktzyklenzähler zurück.

Natürlich lässt sich der Prozessor mit einem Barrel–Shifter ausstatten, so dass auch Schiebebefehle in drei Takten ausgeführt werden. Ebenso lassen sich Interrupts, Multiplikation und Division einfügen — nur in den *Icestick* passt das Ergebnis dann leider nicht mehr. Wer nach der Lektüre des Quelltextes Lust hat, ein bisschen tiefer einzusteigen, sollte einfach mal in dem Git von Bruno Levy [2] oder in Mecrisp–Quintus stöbern.

Links

[1] `https://github.com/BrunoLevy/learn-fpga/issues/1`

[2] `https://github.com/BrunoLevy/learn-fpga/tree/master/FemtoRV/RTL/PROCESSOR`

Verilog–Quelltext des FemtoRV32–Quark

```
1   /***************************************************************/
2   // FemtoRV32, a collection of minimalistic RISC-V RV32 cores.
3   // This version: The "Quark", the most elementary version of FemtoRV32.
4   //             A single VERILOG file, compact & understandable code.
5   //             (200 lines of code, 400 lines counting comments)
6   //
7   // Parameters:
8   //  Reset address can be defined using RESET_ADDR (default is 0).
9   //
10  //  The ADDR_WIDTH parameter lets you define the width of the internal
11  //  address bus (and address computation logic).
12  //
13  // Macros:
14  //    optionally one may define NRV_IS_IO_ADDR(addr), that is supposed to:
15  //              evaluate to 1 if addr is in mapped IO space,
16  //              evaluate to 0 otherwise
17  //    (additional wait states are used when in IO space).
18  //    If left undefined, wait states are always used.
19  //
20  //    NRV_COUNTER_WIDTH may be defined to reduce the number of bits used
21  //    by the ticks counter. If not defined, a 32-bits counter is generated.
```

```
22  //    (reducing its width may be useful for space-constrained designs).
23  //
24  //    NRV_TWOLEVEL_SHIFTER may be defined to make shift operations faster
25  //    (uses a two-level shifter inspired by picorv32).
26  //
27  // Bruno Levy, Matthias Koch, 2020-2021
28  /***************************************************************/
29
30  module FemtoRV32(
31     input          clk,
32
33     output [31:0] mem_addr,  // address bus
34     output [31:0] mem_wdata, // data to be written
35     output  [3:0] mem_wmask, // write mask for the 4 bytes of each word
36     input  [31:0] mem_rdata, // input lines for both data and instr
37     output        mem_rstrb, // active to initiate memory read (used by IO)
38     input         mem_rbusy, // asserted if memory is busy reading value
39     input         mem_wbusy, // asserted if memory is busy writing value
40
41     input         reset      // set to 0 to reset the processor
42  );
43
44     parameter RESET_ADDR       = 32'h00000000;
45     parameter ADDR_WIDTH       = 24;
46
47     localparam ADDR_PAD = {(32-ADDR_WIDTH){1'b0}}; // 32-bits padding for addrs
48
49    /**************************************************************************/
50    // Instruction decoding.
51    /**************************************************************************/
52
53    // Extracts rd,rs1,rs2,funct3,imm and opcode from instruction.
54    // Reference: Table page 104 of:
55    // https://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf
56
57    // The destination register
58    wire [4:0] rdId = instr[11:7];
59
60    // The ALU function, decoded in 1-hot form (doing so reduces LUT count)
61    // It is used as follows: funct3Is[val] <=> funct3 == val
62    (* onehot *)
63    wire [7:0] funct3Is = 8'b00000001 << instr[14:12];
64
65    // The five immediate formats, see RiscV reference (link above), Fig. 2.4 p. 12
66    wire [31:0] Uimm = {    instr[31],   instr[30:12], {12{1'b0}}};
67    wire [31:0] Iimm = {{21{instr[31]}}, instr[30:20]};
68    /* verilator lint_off UNUSED */ // MSBs of SBJimms are not used by addr adder.
69    wire [31:0] Simm = {{21{instr[31]}}, instr[30:25],instr[11:7]};
70    wire [31:0] Bimm = {{20{instr[31]}}, instr[7],instr[30:25],instr[11:8],1'b0};
71    wire [31:0] Jimm = {{12{instr[31]}}, instr[19:12],instr[20],instr[30:21],1'b0};
72    /* verilator lint_on UNUSED */
73
74      // Base RISC-V (RV32I) has only 10 different instructions !
75      wire isLoad    =  (instr[6:2] == 5'b00000); // rd <- mem[rs1+Iimm]
76      wire isALUimm  =  (instr[6:2] == 5'b00100); // rd <- rs1 OP Iimm
77      wire isAUIPC   =  (instr[6:2] == 5'b00101); // rd <- PC + Uimm
78      wire isStore   =  (instr[6:2] == 5'b01000); // mem[rs1+Simm] <- rs2
79      wire isALUreg  =  (instr[6:2] == 5'b01100); // rd <- rs1 OP rs2
80      wire isLUI     =  (instr[6:2] == 5'b01101); // rd <- Uimm
81      wire isBranch  =  (instr[6:2] == 5'b11000); // if(rs1 OP rs2) PC<-PC+Bimm
82      wire isJALR    =  (instr[6:2] == 5'b11001); // rd <- PC+4; PC<-rs1+Iimm
83      wire isJAL     =  (instr[6:2] == 5'b11011); // rd <- PC+4; PC<-PC+Jimm
84      wire isSYSTEM  =  (instr[6:2] == 5'b11100); // rd <- cycles
85
86      wire isALU = isALUimm | isALUreg;
87
88      /**************************************************************************/
89      // The register file.
90      /**************************************************************************/
91
92      reg [31:0] rs1;
93      reg [31:0] rs2;
94      reg [31:0] registerFile [31:0];
95
96      always @(posedge clk) begin
97        if (writeBack)
```

```
98          if (rdId != 0)
99            registerFile[rdId] <= writeBackData;
100     end
101
102     /***************************************************************************/
103     // The ALU. Does operations and tests combinatorially, except shifts.
104     /***************************************************************************/
105
106     // First ALU source, always rs1
107     wire [31:0] aluIn1 = rs1;
108
109     // Second ALU source, depends on opcode:
110     //    ALUreg, Branch:     rs2
111     //    ALUimm, Load, JALR: Iimm
112     wire [31:0] aluIn2 = isALUreg | isBranch ? rs2 : Iimm;
113
114     reg  [31:0] aluReg;       // The internal register of the ALU, used by shift.
115     reg  [4:0]  aluShamt;     // Current shift amount.
116
117     wire aluBusy = |aluShamt; // ALU is busy if shift amount is non-zero.
118     wire aluWr;               // ALU write strobe, starts shifting.
119
120     // The adder is used by both arithmetic instructions and JALR.
121     wire [31:0] aluPlus = aluIn1 + aluIn2;
122
123     // Use a single 33 bits subtract to do subtraction and all comparisons
124     // (trick borrowed from swapforth/J1)
125     wire [32:0] aluMinus = {1'b1, ~aluIn2} + {1'b0,aluIn1} + 33'b1;
126     wire        LT  = (aluIn1[31] ^ aluIn2[31]) ? aluIn1[31] : aluMinus[32];
127     wire        LTU = aluMinus[32];
128     wire        EQ  = (aluMinus[31:0] == 0);
129
130     // Notes:
131     // - instr[30] is 1 for SUB and 0 for ADD
132     // - for SUB, need to test also instr[5] to discriminate ADDI:
133     //    (1 for ADD/SUB, 0 for ADDI, and Iimm used by ADDI overlaps bit 30 !)
134     // - instr[30] is 1 for SRA (do sign extension) and 0 for SRL
135
136     wire [31:0] aluOut =
137       (funct3Is[0]  ? instr[30] & instr[5] ? aluMinus[31:0] : aluPlus : 32'b0) |
138       (funct3Is[2]  ? {31'b0, LT}                                     : 32'b0) |
139       (funct3Is[3]  ? {31'b0, LTU}                                    : 32'b0) |
140       (funct3Is[4]  ? aluIn1 ^ aluIn2                                 : 32'b0) |
141       (funct3Is[6]  ? aluIn1 | aluIn2                                 : 32'b0) |
142       (funct3Is[7]  ? aluIn1 & aluIn2                                 : 32'b0) |
143       (funct3IsShift ? aluReg                                         : 32'b0) ;
144
145     wire funct3IsShift = funct3Is[1] | funct3Is[5];
146
147     always @(posedge clk) begin
148        if(aluWr) begin
149           if (funct3IsShift) begin  // SLL, SRA, SRL
150              aluReg <= aluIn1;
151              aluShamt <= aluIn2[4:0];
152           end
153        end
154
155   `ifdef NRV_TWOLEVEL_SHIFTER
156        else if(|aluShamt[3:2]) begin // Shift by 4
157           aluShamt <= aluShamt - 4;
158           aluReg <= funct3Is[1] ? aluReg << 4 :
159                     {{4{instr[30] & aluReg[31]}}, aluReg[31:4]};
160        end  else
161   `endif
162        // Compact form of:
163        // funct3=001               -> SLL  (aluReg <= aluReg << 1)
164        // funct3=101 &  instr[30] -> SRA  (aluReg <= {aluReg[31], aluReg[31:1]})
165        // funct3=101 & !instr[30] -> SRL  (aluReg <= {1'b0,       aluReg[31:1]})
166
167        if (|aluShamt) begin
168           aluShamt <= aluShamt - 1;
169           aluReg <= funct3Is[1] ? aluReg << 1 :              // SLL
170                     {instr[30] & aluReg[31], aluReg[31:1]};  // SRA,SRL
171        end
172     end
173
```

```
174     /***************************************************************************/
175     // The predicate for conditional branches.
176     /***************************************************************************/
177
178     wire predicate =
179          funct3Is[0] &  EQ  | // BEQ
180          funct3Is[1] & !EQ  | // BNE
181          funct3Is[4] &  LT  | // BLT
182          funct3Is[5] & !LT  | // BGE
183          funct3Is[6] &  LTU | // BLTU
184          funct3Is[7] & !LTU ; // BGEU
185
186     /***************************************************************************/
187     // Program counter and branch target computation.
188     /***************************************************************************/
189
190     reg  [ADDR_WIDTH-1:0] PC; // The program counter.
191     reg  [31:2] instr;        // Latched instruction. Note that bits 0 and 1 are
192                               // ignored (not used in RV32I base instr set).
193
194     wire [ADDR_WIDTH-1:0] PCplus4 = PC + 4;
195
196     // An adder used to compute branch address, JAL address and AUIPC.
197     // branch->PC+Bimm    AUIPC->PC+Uimm    JAL->PC+Jimm
198     // Equivalent to PCplusImm = PC + (isJAL ? Jimm : isAUIPC ? Uimm : Bimm)
199     wire [ADDR_WIDTH-1:0] PCplusImm = PC + ( instr[3] ? Jimm[ADDR_WIDTH-1:0] :
200                                              instr[4] ? Uimm[ADDR_WIDTH-1:0] :
201                                                         Bimm[ADDR_WIDTH-1:0] );
202
203     // A separate adder to compute the destination of load/store.
204     // testing instr[5] is equivalent to testing isStore in this context.
205     wire [ADDR_WIDTH-1:0] loadstore_addr = rs1[ADDR_WIDTH-1:0] +
206                     (instr[5] ? Simm[ADDR_WIDTH-1:0] : Iimm[ADDR_WIDTH-1:0]);
207
208     assign mem_addr = {ADDR_PAD,
209                        state[WAIT_INSTR_bit] | state[FETCH_INSTR_bit] ?
210                        PC : loadstore_addr
211                       };
212
213     /***************************************************************************/
214     // The value written back to the register file.
215     /***************************************************************************/
216
217     wire [31:0] writeBackData  =
218        /* verilator lint_off WIDTH */
219        (isSYSTEM            ? cycles              : 32'b0) |  // SYSTEM
220        /* verilator lint_on WIDTH */
221        (isLUI               ? Uimm                : 32'b0) |  // LUI
222        (isALU               ? aluOut              : 32'b0) |  // ALUreg, ALUimm
223        (isAUIPC             ? {ADDR_PAD,PCplusImm} : 32'b0) |  // AUIPC
224        (isJALR   | isJAL    ? {ADDR_PAD,PCplus4  } : 32'b0) |  // JAL, JALR
225        (isLoad              ? LOAD_data           : 32'b0);   // Load
226
227     /***************************************************************************/
228     // LOAD/STORE
229     /***************************************************************************/
230
231     // All memory accesses are aligned on 32 bits boundary. For this
232     // reason, we need some circuitry that does unaligned halfword
233     // and byte load/store, based on:
234     // - funct3[1:0]:  00->byte 01->halfword 10->word
235     // - mem_addr[1:0]: indicates which byte/halfword is accessed
236
237     wire mem_byteAccess     = instr[13:12] == 2'b00; // funct3[1:0] == 2'b00;
238     wire mem_halfwordAccess = instr[13:12] == 2'b01; // funct3[1:0] == 2'b01;
239
240     // LOAD, in addition to funct3[1:0], LOAD depends on:
241     // - funct3[2] (instr[14]): 0->do sign expansion   1->no sign expansion
242
243     wire LOAD_sign =
244          !instr[14] & (mem_byteAccess ? LOAD_byte[7] : LOAD_halfword[15]);
245
246     wire [31:0] LOAD_data =
247          mem_byteAccess ? {{24{LOAD_sign}},     LOAD_byte} :
248       mem_halfwordAccess ? {{16{LOAD_sign}}, LOAD_halfword} :
249                            mem_rdata ;
```

```
250
251     wire [15:0] LOAD_halfword =
252                 loadstore_addr[1] ? mem_rdata[31:16] : mem_rdata[15:0];
253
254     wire  [7:0] LOAD_byte =
255                 loadstore_addr[0] ? LOAD_halfword[15:8] : LOAD_halfword[7:0];
256
257     // STORE
258
259     assign mem_wdata[ 7: 0] = rs2[7:0];
260     assign mem_wdata[15: 8] = loadstore_addr[0] ? rs2[7:0]  : rs2[15: 8];
261     assign mem_wdata[23:16] = loadstore_addr[1] ? rs2[7:0]  : rs2[23:16];
262     assign mem_wdata[31:24] = loadstore_addr[0] ? rs2[7:0]  :
263                               loadstore_addr[1] ? rs2[15:8] : rs2[31:24];
264
265     // The memory write mask:
266     //    1111                     if writing a word
267     //    0011 or 1100             if writing a halfword
268     //                                (depending on loadstore_addr[1])
269     //    0001, 0010, 0100 or 1000 if writing a byte
270     //                                (depending on loadstore_addr[1:0])
271
272     wire [3:0] STORE_wmask =
273                 mem_byteAccess      ?
274                       (loadstore_addr[1] ?
275                             (loadstore_addr[0] ? 4'b1000 : 4'b0100) :
276                             (loadstore_addr[0] ? 4'b0010 : 4'b0001)
277                       ) :
278                 mem_halfwordAccess ?
279                       (loadstore_addr[1] ? 4'b1100 : 4'b0011) :
280                 4'b1111;
281
282     /***************************************************************************/
283     // And, last but not least, the state machine.
284     /***************************************************************************/
285
286     localparam FETCH_INSTR_bit     = 0;
287     localparam WAIT_INSTR_bit      = 1;
288     localparam EXECUTE_bit         = 2;
289     localparam WAIT_ALU_OR_MEM_bit = 3;
290     localparam NB_STATES           = 4;
291
292     localparam FETCH_INSTR     = 1 << FETCH_INSTR_bit;
293     localparam WAIT_INSTR      = 1 << WAIT_INSTR_bit;
294     localparam EXECUTE         = 1 << EXECUTE_bit;
295     localparam WAIT_ALU_OR_MEM = 1 << WAIT_ALU_OR_MEM_bit;
296
297     (* onehot *)
298     reg [NB_STATES-1:0] state;
299
300     // The signals (internal and external) that are determined
301     // combinatorially from state and other signals.
302
303     // register write-back enable.
304     wire writeBack = ~(isBranch | isStore ) &
305                      (state[EXECUTE_bit] | state[WAIT_ALU_OR_MEM_bit]);
306
307     // The memory-read signal.
308     assign mem_rstrb = state[EXECUTE_bit] & isLoad | state[FETCH_INSTR_bit];
309
310     // The mask for memory-write.
311     assign mem_wmask = {4{state[EXECUTE_bit] & isStore}} & STORE_wmask;
312
313     // aluWr starts computation (shifts) in the ALU.
314     assign aluWr = state[EXECUTE_bit] & isALU;
315
316     wire jumpToPCplusImm = isJAL | (isBranch & predicate);
317   `ifdef NRV_IS_IO_ADDR
318     wire needToWait = isLoad |
319                       isStore  & `NRV_IS_IO_ADDR(mem_addr) |
320                       isALU & funct3IsShift;
321   `else
322     wire needToWait = isLoad | isStore | isALU & funct3IsShift;
323   `endif
324
325     always @(posedge clk) begin
```

```
326         if(!reset) begin
327            state      <= WAIT_ALU_OR_MEM; // Just waiting for !mem_wbusy
328            PC         <= RESET_ADDR[ADDR_WIDTH-1:0];
329         end else
330
331         // See note [1] at the end of this file.
332         (* parallel_case *)
333         case(1'b1)
334
335           state[WAIT_INSTR_bit]: begin
336              if(!mem_rbusy) begin // may be high when executing from SPI flash
337                 rs1 <= registerFile[mem_rdata[19:15]];
338                 rs2 <= registerFile[mem_rdata[24:20]];
339                 instr <= mem_rdata[31:2]; // Bits 0 and 1 are ignored (see
340                 state <= EXECUTE;         // also the declaration of instr).
341              end
342           end
343
344           state[EXECUTE_bit]: begin
345              PC <= isJALR          ? {aluPlus[ADDR_WIDTH-1:1],1'b0} :
346                    jumpToPCplusImm ? PCplusImm :
347                    PCplus4;
348              state <= needToWait ? WAIT_ALU_OR_MEM : FETCH_INSTR;
349           end
350
351           state[WAIT_ALU_OR_MEM_bit]: begin
352              if(!aluBusy & !mem_rbusy & !mem_wbusy) state <= FETCH_INSTR;
353           end
354
355           default: begin // FETCH_INSTR
356             state <= WAIT_INSTR;
357           end
358
359         endcase
360      end
361
362      /***************************************************************************/
363      // Cycle counter
364      /***************************************************************************/
365
366   `ifdef NRV_COUNTER_WIDTH
367      reg [`NRV_COUNTER_WIDTH-1:0]  cycles;
368   `else
369      reg [31:0]  cycles;
370   `endif
371      always @(posedge clk) cycles <= cycles + 1;
372
373   endmodule
374
375   /*****************************************************************************/
376   // Notes:
377   //
378   // [1] About the "reverse case" statement, also used in Claire Wolf's picorv32:
379   // It is just a cleaner way of writing a series of cascaded if() statements,
380   // To understand it, think about the case statement *in general* as follows:
381   // case (expr)
382   //       val_1: statement_1
383   //       val_2: statement_2
384   //   ... val_n: statement_n
385   // endcase
386   // The first statement_i such that expr == val_i is executed.
387   // Now if expr is 1'b1:
388   // case (1'b1)
389   //       cond_1: statement_1
390   //       cond_2: statement_2
391   //   ... cond_n: statement_n
392   // endcase
393   // It is *exactly the same thing*, the first statement_i such that
394   // expr == cond_i is executed (that is, such that 1'b1 == cond_i,
395   // in other words, such that cond_i is true)
396   // More on this:
397   //     https://stackoverflow.com/questions/15418636/case-statement-in-verilog
398   //
399   // [2] state uses 1-hot encoding (at any time, state has only one bit set to 1).
400   // It uses a larger number of bits (one bit per state), but often results in
401   // a both more compact (fewer LUTs) and faster state machine.
```

GPIOs mit dem Raspberry Pi und Odroid N2

Bernd Paysan

Die Raspberry Pis (und Konkurrenten wie Odroid) verstehen sich als „Bastelcomputer“, auch wenn sie Luxusprozessoren für ein Workstation&Server–OS (Linux) enthalten, inzwischen ja auch mit 64 Bit. Trotzdem gibt es da einen 40–poligen Pfostenstecker, die GPIOs, die man tatsächlich für dieses oder jenes Gebastel nutzen kann. Dieser Artikel verrät, wie das mit der aktuellen Entwicklerversion von Gforth (bald als Version 1.0 released) möglich wird.

Da ich hier keinen Raspberry Pi habe, habe ich das alles erst mal auf meinen Odroiden (insbesondere dem N2+) zum Laufen gebracht, und dann auf den Raspberry Pi 4 portiert und remote ausprobiert.

Raspberry Pi als Bastel–Computer

Was braucht so ein richtiger Bastel–Computer? Richtig: eine Möglichkeit, noch weitere Basteleien anzustöpseln, siehe Artikel TheStack in diesem Heft. Das gilt nicht nur für Arduinos und Forth–Platinen, sondern auch für den Raspberry Pi, der auch eine vierzigpolige Pfostenleiste zum Basteln eingeführt hat, die etwas unaufgeräumt noch kreuz und quer Versorgungen in unterschiedlicher Spannung und Masse enthält.

Was bei den Arduinos “Shields” heißt, heißt bei den Raspberries “HATs”[1]. Es gibt auch Prototypen–Boards zum Basteln eigener HATs, und letztlich braucht man eigentlich nur einen 40–poligen Pfostenstecker und eine Rasterplatine, dann kann man selber loslegen. Man kann das Ganze auch fliegend verdrahten.

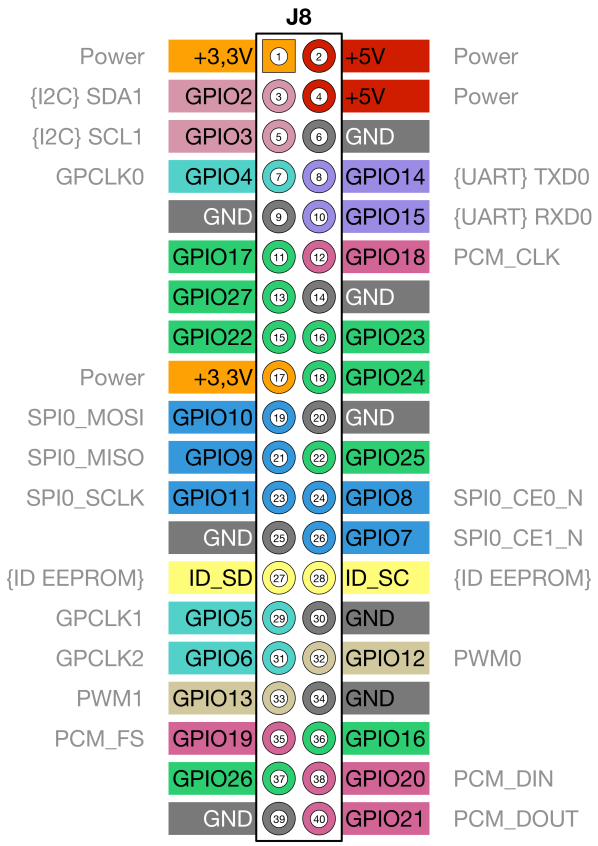


Abbildung 1: GPIO–Leiste des Raspberry Pi [1]

[1] Hardware Attached on Top

Das gibt es dann auch von den Clones in Varianten, wobei man sich dabei so weit wie möglich an das Original vom Raspberry Pi gehalten hat. Was sich dabei verschiebt, sind die GPIO–Nummern, denn die Funktionalitäten wie SPI oder I²C sollen sich möglichst nicht verschieben. Trotzdem haben andere SoCs natürlich andere Schnittstellen, und entsprechend leicht anders sieht das alles dann aus.

Abbildung 2: GPIO–Leiste des Odroid N2+ [2]

GPIOs, aber komplex

Vor ca. 40 Jahren, als CPUs noch mit NMOS–Technologie implementiert wurden, waren 8 GPIOs einfach ein Byte im Adressraum, auf das man schreiben und von dem man lesen konnte. Getrieben wurden eh nur die 0en, und die Pullups zogen auf 1. Mehr gab es nicht, für I²C reicht das aber völlig aus (was man dann als Bitbang–Implementierung realisierte, und schon froh war, wenn man die wahnsinnig schnellen 100 kHz überhaupt erreichte).

Das ist heute natürlich anders. CPUs sind in CMOS implementiert, die IOs können nur noch maximal 3,3 V, und dafür braucht man eine Reihe Register, um die GPIOs anzusprechen. Dabei haben Odroid und Raspberry Pi sogar recht grundsätzlich verschiedene Lösungen gefunden. Mir gefällt die Lösung vom Odroid deutlich besser.

Die GPIOs werden wie vor 40 Jahren immer noch über „memory mapped IO“ angesprochen, und man kann sich das unter Linux auch einfach über das Device `/dev/gpiomem` im Usermode in den Speicher mappen, wenn man Zugriffsrechte auf diese Datei hat. Insofern ist das alles Peek und Poke, bzw. natürlich `@` und `!` in Forth, wie vor 40 Jahren auch schon. Nur, dass man auf einem 64–Bit–Forth `L@` und `L!` nehmen muss, weil die Register „nur“ 32 Bit breit sind. Hier drei Dateien, die man in `/etc/udev/rules.d` hineintut, damit man die passenden Zugriffsrechte bekommt. Den eigenen Nutzer muss man dann „nur noch“ in die neu angelegten Gruppen `gpio`, `spi` und `i2c` packen. Beim Pi ist das alles schon passend aufgesetzt, beim Odroid N2+ musste ich folgende Dateien noch anlegen:

`99-odroid-wiringpi-aml.rules`:

```
1   SUBSYSTEM=="aml-gpiomem", GROUP="gpio", MODE="0660"
```

`99-spidev.rules`:

```
1   KERNEL=="spidev*", RUN="/bin/sh -c 'chgrp -R
     spi /dev/spidev* && chmod -R g+rw /dev/spidev*'"
```

`99-i2c.rules`:

```
1   KERNEL=="i2c*", RUN="/bin/sh -c 'chgrp -R
     i2c /dev/i2c* && chmod -R g+rw /dev/i2c*'"
```

Zum Ausprobieren habe ich mir von Reichelt jeweils zwei EEPROMs von Microchip mit 8– und 16–Bit–Adressen sowie SPI und I²C bestellt, eine passende Anzahl 8–Pin–Sockel, ein Rasterboard und Pinleisten. Fliegende Verdrahtung mit Klemmen auf einer Seite waren schon da. Mein Odroid lebt in dem Lowboard unter dem Fernseher.

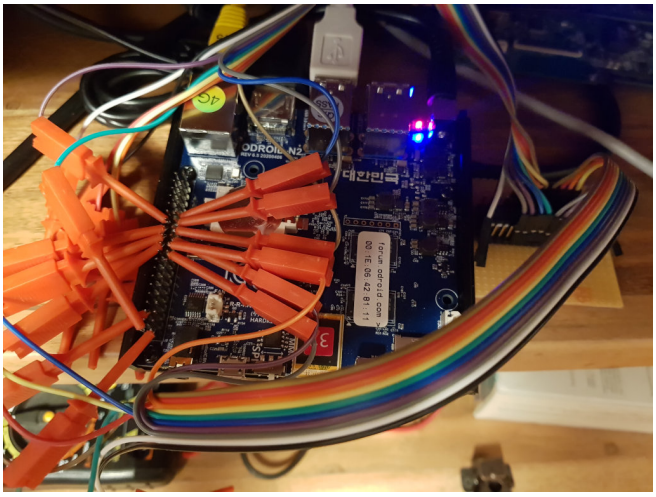


Abbildung 3: Fliegende Verdrahtung

GPIOs auf dem Odroid N2+[3]

Jeder Port (Ports sind 32 Bits breit) hat ein Register für die Richtung (Input/Output, FSEL genannt, 0 ist Output, 1 ist Input). Es gibt ein OUTP–Register zum Schreiben für die Outputs und ein INP–Register zum Lesen der Inputs. Für die Pullups gibt es ein PUEN zum Enablen, und ein PUPD zum Auswählen, ob man Pullup oder Pulldown haben will — und da ist die Richtung auch schön intuitiv, also Pullup bekommt man mit einer 1 und Pulldown mit einer 0. Und dann gibt es noch ein MUX–Register (4 Bits pro Pin) für die Auswahl erweiterter Funktionen (wie SPI oder I²C), dessen Mode 0 der eigentliche GPIO–Mode ist.

GPIOs auf dem Raspberry Pi (BCM 2835 bzw. BCM 2711[4] für den Pi4)

Hier ist Richtung (FSEL) und Auswahl erweiterter Funktionen verwurstet: 0 ist Input, 1 ist Output, 2 und höher sind spezielle Funktionen. Mit dem LEV–Register kann man auch den Input direkt auslesen, aber zum Schreiben des Outputs gibt es ein SET– und ein CLR–Register. Man braucht also kein Read–Modify–Write mit einer Maske, um Portbits gezielt zu setzen oder zu löschen, ein einfacher Write reicht. Dafür kann man nicht mehrere Portbits gleichzeitig auf einen bestimmten Wert setzen, also, einer Nutzung als Parallelport ist das eher im Wege.

Für die Pullups und Pulldowns hat der 2835 eine ganz merkwürdige Logik, der 2711 aber etwas Vernünftigeres. Da der RPi 4 den 2711 drin hat, konnte ich auch nur das ausprobieren: Hier hat jeder Pullup zwei Bits, %00 ist kein Widerstand, %01 ist Pullup und %10 ist Pulldown. Die Werte folgen keiner inneren Logik und die Auswahl Pullup/Pulldown ist beim 2835 auch genau andersrum.

Für den User

Der bastelnde User hätte wahrscheinlich gerne ein etwas abstrakteres Interface, das insbesondere die Umrechnung von Pins auf der Steckerleiste übernimmt. Folgende Wörter für den Endnutzer gibt es:

pin@ (n — bit) liest den Pegel am Pin Nummer *n*.

pin! (bit n —) setzt den Pegel am Pin Nummer *n*.

pinset (n —) setzt den Pegel am Pin Nummer *n* auf 1

pinclr (n —) setzt den Pegel am Pin Nummer *n* auf 0

input–pin (n —) setzt Pin Nummer *n* als Input

output–pin (n —) setzt Pin Nummer *n* als Output

pin–resmode (mode n —) setzt den Pullup/Pulldown von Pin Nummer *n* auf den Modus (0: hochohmig, 1: Pullup, 2: Pulldown)

Alle diese Befehle werden ja meistens mit Konstanten benutzt, und werden dank Constant–Folding im Gforth dann auch entsprechend optimiert.

SPI

Ein sehr einfaches Protokoll, um mit Hardware zu reden, ist SPI, oder „Vierdrahtinterface“. Die vier Drähte sind: Master Out/Slave In (MOSI), Master In/Slave Out (MISO), Clock (SCLK) und Chip Select (CS), wobei man die ersten drei für alle angebundenen SPI–Peripherals gemeinsam nutzen kann, und nur jeweils ein Chip Select braucht. Die Logik ist die eines Schieberegisters, man taktet die Daten aus dem MOSI an der steigenden Flanke der Clock in das Device hinein, und bekommt die Daten vom Device über MISO an der fallenden Flanke in den Master zurück. Alle Treiber sind Push/Pull (CMOS), also schnell; der Master gibt den Takt vor. Es gibt keine besondere Fehlerbehandlung, es ist nicht multimaster–fähig, und es ist einfach zu implementieren. Wenn man also ein FPGA oder so für echt knifflige Aufgaben an seinen RPi anschließen will, dann ist SPI das Protokoll der Wahl. Mit SPI schafft man auch 20–MHz–Transferraten, ohne dass man irgendwie Probleme bekommt.

Was man auf einem RPi oder Odroid N2 aber nicht macht: das dann per Bitbang selbst zu implementieren. Es gibt dafür eigene Hardware im SoC, und Kernel–Treiber dafür. Die Treiberlogik läuft über Dateien und `ioctl()` zum eigentlichen Abarbeiten. Dabei kann man mehrere Transfer–Blöcke auf einmal abarbeiten lassen, also etwa zuerst ein paar Bytes schreiben (Kommando+Adresse), und dann einige Bytes lesen; auch Lesen und Schreiben gleichzeitig ist über dieses Interface machbar.

Für eine Beispiel–Implementierung habe ich mir SPI–EEPROMs von Microchip herausgesucht, die mit einer Byte– oder 16–Bit–Adresse angesprochen werden können. Neben dem SPI–Interface hat das verwendete EEPROM noch einen Hold– und einen WP(write protect)–Pin, die über normale GPIOs realisiert werden können. Das Kernel–Interface lässt sich mit SWIG problemlos in eine Forth–Datei konvertieren:

`spi.i`

```
1   // this file is in the public domain
2   %module spi
3   %insert("include")
4   %{
5   #include <linux/spi/spidev.h>
6   #ifdef __gnu_linux__
7   #undef stderr
8   extern struct _IO_FILE *stderr;
9   #endif
10  %}
11
12  #define SWIG_FORTH_OPTIONS "no-pre-postfix"
13
14  %include <linux/spi/spidev.h>
```

SPI–Kommandos

Um das EEPROM zu lesen und zu beschreiben, gibt es dann folgende Kommandos:

Generische Kommandos

spi–status@ (— status) Liest den Status

spi–status! (status —) Schreibt den Status

spi–wren (—) Write enable (muss vor jedem Write erfolgen)

spi–wrdi (—) Write disable

spi–wip| (—) Wartet, bis der Write in Progress auf 0 geht

Für EEPROMs mit 8–Bit–Adressen

spi–c@ (addr — c) Liest ein Byte

spi–w@ (addr — w) Liest ein Wort (16 Bits)

spi–l@ (addr — l) Liest ein Long (32 Bits)

spi–x@ (addr — x) Liest ein Extra (64 Bits)

spi–c! (c addr —) Schreibt ein Byte

spi–w! (w addr —) Schreibt ein Wort (16 Bits)

spi–l! (l addr —) Schreibt ein Long (32 Bits)

spi–x! (x addr —) Schreibt ein Extra (64 Bits)

Für EEPROMs mit 16–Bit–Adressen

spiw–c@ (addr — c) Liest ein Byte

spiw–w@ (addr — w) Liest ein Wort (16 Bits)

spiw–l@ (addr — l) Liest ein Long (32 Bits)

spiw–x@ (addr — x) Liest ein Extra (64 Bits)

spiw–c! (c addr —) Schreibt ein Byte

spiw–w! (w addr —) Schreibt ein Wort (16 Bits)

spiw–l! (l addr —) Schreibt ein Long (32 Bits)

spiw–x! (x addr —) Schreibt ein Extra (64 Bits)

I²C

Weniger Drähte (nämlich nur 2), und ein deutlich komplexeres Protokoll hat I²C, oder „Zweidrahtinterface“. Das ist multimasterfähig, dafür aber deutlich langsamer, denn es stammt aus der NMOS–Zeit, als man noch mit Pullups arbeitete. Je nach Stärke des Pullups schafft man entweder 100 kHz oder 400 kHz. Die Devices können mit ACK/NACK auf jedes Byte antworten, der Master erfährt also, ob der Transfer geklappt hat. Das ist natürlich doof, weil es den Device–Bauern jetzt die Möglichkeit eröffnet, einfach nicht immer zu reagieren, und man dann Fehlerbehandlung machen und ggf. mehrmals nachfragen muss — mit Pause.

Das Kernel–Interface sieht grundsätzlich etwa gleich aus, nur dass es kein gleichzeitiges Lesen und Schreiben gibt.

`i2c.i`

```
1   // this file is in the public domain
2   %module i2c
3   %insert("include")
4   %{
5   #include <linux/i2c.h>
6   #include <linux/i2c-dev.h>
7   #ifdef __gnu_linux__
8   #undef stderr
9   extern struct _IO_FILE *stderr;
```

```
10  #endif
11  %}
12
13  #define __user
14  #define SWIG_FORTH_OPTIONS "no-pre-postfix"
15
16  %include <linux/i2c.h>
17  %include <linux/i2c-dev.h>
```

I²C–Kommandos

Um das EEPROM zu lesen und zu beschreiben, gibt es dann folgende Kommandos:

Generische Kommandos

i2c–addr (addr —) Setzt die Device–Adresse für alle nachfolgenden I²C–Kommandos

Für EEPROMs mit 8–Bit–Adressen

i2c–c@ (addr — c) Liest ein Byte

i2c–w@ (addr — w) Liest ein Wort (16 Bits)

i2c–l@ (addr — l) Liest ein Long (32 Bits)

i2c–x@ (addr — x) Liest ein Extra (64 Bits)

i2c–c! (c addr —) Schreibt ein Byte

i2c–w! (w addr —) Schreibt ein Wort (16 Bits)

i2c–l! (l addr —) Schreibt ein Long (32 Bits)

i2c–x! (x addr —) Schreibt ein Extra (64 Bits)

Für EEPROMs mit 16–Bit–Adressen

i2cw–c@ (addr — c) Liest ein Byte

i2cw–w@ (addr — w) Liest ein Wort (16 Bits)

i2cw–l@ (addr — l) Liest ein Long (32 Bits)

i2cw–x@ (addr — x) Liest ein Extra (64 Bits)

i2cw–c! (c addr —) Schreibt ein Byte

i2cw–w! (w addr —) Schreibt ein Wort (16 Bits)

i2cw–l! (l addr —) Schreibt ein Long (32 Bits)

i2cw–x! (x addr —) Schreibt ein Extra (64 Bits)

Referenzen

[1] Pinbelegung für den Raspberry Pi übersichtlich: https://indibit.de/raspberry-pi-die-gpio-schnittstelle-grundlagenbelegung/

[2] Pinbelegung für den Odroid N2+ übersichtlich: https://wiki.odroid.com/odroid-n2/hardware/expansion_connectors

[3] Datasheet für den Amlogic S922X (Odroid N2+): https://dn.odroid.com/S922X/ODROID-N2/Datasheet/S922X_Public_Datasheet_V0.2.pdf

[4] Datasheet für den Broadcom BCM 2711 (Raspberry Pi 4 B): https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/README.md

Listings

gpios.fs

```
1   \ gpios.fs      GPIO access
2   \
3   \ Authors: Bernd Paysan
4   \ Copyright (C) 2021 Free Software Foundation, Inc.
5
6   \ This file is part of Gforth.
7
8   \ Gforth is free software; you can redistribute it
9   \ and/or modify it under the terms of the GNU
10  \ General Public License as published by the Free
11  \ Software Foundation, either version 3 of the
12  \ License, or (at your option) any later version.
13
14  \ This program is distributed in the hope that it
15  \ will be useful, but WITHOUT ANY WARRANTY; without
16  \ even the implied warranty of MERCHANTABILITY or
17  \ FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
18  \ General Public License for more details.
19
20  \ You should have received a copy of the GNU General
21  \ Public License along with this program. If not,
22  \ see http://www.gnu.org/licenses/.
23
24  require unix/libc.fs
25  require unix/mmap.fs
26
27  0 Value gpio-base
28
29  : reg: ( offset -- )
30      Create sfloats ,
31    DOES> @ gpio-base + ;
32
33  \ register access
34
35  : lor! ( l addr -- )
36      >r r@ l@ or r> l! ;
37  : land! ( l addr -- )
38      >r r@ l@ and r> l! ;
39  : lmask! ( l mask addr -- )
40      \G set the bits where @var{mask} is 1
41      \G in @var{addr} by the corresponding
42      \G bits in @var{l}.
43      >r r@ l@ swap mux r> l! ;
44
45  \ mask generation
46
47  1 Constant 1bit ( addr gpio -- addr gpio mask )
48  : 2bit ( addr gpio -- addr' gpio' mask )
49      2* tuck 5 rshift sfloats + swap $1F and 3 ;
50  : 3bit ( addr gpio -- addr' gpio' mask )
51      #10 /mod swap >r sfloats + r> dup 2* + 7 ;
52  : 4bit ( addr gpio -- addr' gpio' mask )
53      2* 2* tuck 5 rshift sfloats + swap $1F and $F ;
54
55  \ device specific stuff
56
57  s" /sys/firmware/devicetree/base/model"
58  ' slurp-file catch
59  [IF] 2drop s" unknown" [THEN] 2Constant model
60  : model?" ( "name"<"> -- flag )
61      model '"' parse search nip nip ;
62
63  model?" ODROID-N2" [IF]
64      model?" ODROID-N2Plus" [IF]
65          : odroid-n2+ ;
66      [ELSE]
```

```
        : odroid-n2 ;
[THEN]
$FF634000 Constant GPIO-Base-map

\ actual offset is $FF634400, i.e. starts at reg $100

\ mux is 0 for IO, 1..F for other functions
\ FSEL bit is 0 for Output, 1 for Input
\ pins have pullups/downs, which can be enabled

$116 reg: N2_GPIOX_FSEL_REG
$117 reg: N2_GPIOX_OUTP_REG
$118 reg: N2_GPIOX_INP_REG

$120 reg: N2_GPIOA_FSEL_REG
$121 reg: N2_GPIOA_OUTP_REG
$122 reg: N2_GPIOA_INP_REG

$13C reg: N2_GPIOX_PUPD_REG
$13F reg: N2_GPIOA_PUPD_REG

$14A reg: N2_GPIOX_PUEN_REG
$14D reg: N2_GPIOA_PUEN_REG

$1B3 reg: N2_GPIOX_MUX_3_REG \ GPIOX[0:7]
$1B4 reg: N2_GPIOX_MUX_4_REG \ GPIOX[8:15]
$1B5 reg: N2_GPIOX_MUX_5_REG \ GPIOX[16:19]

$1BD reg: N2_GPIOA_MUX_D_REG \ GPIOA[0:7]
$1BE reg: N2_GPIOA_MUX_E_REG \ GPIOA[8:15]

$1D2 reg: N2_GPIOX_DS_REG_2A \ GPIOX[0:15]
$1D3 reg: N2_GPIOX_DS_REG_2B \ GPIOX[16:19]
$1D6 reg: N2_GPIOA_DS_REG_5A

Create shift/type
' 1bit , ' 1bit , ' 1bit , ' 1bit ,
' 1bit , ' 2bit , ' 4bit ,

Variable gpio-dummy

: gpio>mask ( gpio type table -- shift mask addr )
    third -1 = IF
        2drop drop 0 0 gpio-dummy gpio-base -
        EXIT  THEN
    swap { s/t }
    s/t 2* cells + over 5 rshift cells + @
    swap $1F and shift/type s/t cells + perform
    over lshift rot ;

-1
1+ dup Constant fsel#
1+ dup Constant outp#
1+ dup Constant inp#
1+ dup Constant pupd#
1+ dup Constant puen#
1+ dup Constant ds#
1+ dup Constant mux#
drop

Create gpio-reg[]
N2_GPIOX_FSEL_REG  , N2_GPIOA_FSEL_REG  ,
N2_GPIOX_OUTP_REG  , N2_GPIOA_OUTP_REG  ,
N2_GPIOX_INP_REG   , N2_GPIOA_INP_REG   ,
N2_GPIOX_PUPD_REG  , N2_GPIOA_PUPD_REG  ,
N2_GPIOX_PUEN_REG  , N2_GPIOA_PUEN_REG  ,
N2_GPIOX_DS_REG_2A , N2_GPIOA_DS_REG_5A ,
N2_GPIOX_MUX_3_REG , N2_GPIOA_MUX_D_REG ,
  DOES> ( pin type -- shift mask addr )
    gpio>mask gpio-base + ;
[: lits# 2 u>= IF  2lits> rot >body
        gpio>mask >3lits ]] gpio-base + [[
    ELSE  does,  THEN ;] optimizes gpio-reg[]

\ pins to GPIO table: X=$000+, A=$020+
    Create pin>gpio ( pin -- gpio )
    -1   , -1   ,
    $011 , -1   ,
    $012 , -1   ,
    $02D , $00C ,
    -1   , $00D ,
    $003 , $010 ,
    $004 , -1   ,
    $007 , $000 ,
    -1   , $001 ,
    $008 , -1   ,
    $009 , $002 ,
    $00B , $00A ,
    -1   , $024 ,
    $02E , $02F ,
    $00E , -1   ,
    $00F , $02C ,
    $005 , -1   ,
    $006 , $013 ,
    -1   , -1   ,
    -1   , -1   ,
    DOES> swap 1- #39 umin cells + @ ;
      [: lits# 1 u>= IF  >body
          lits> 1- #39 umin cells + @ >lits
        ELSE  does,  THEN ;] optimizes pin>gpio
[THEN]
model?" ODROID-C2" [IF]
    : odroid-c2 ;
    $C8834000 Constant GPIO-Base-map

    $118 reg: C2_GPIOX_FSEL_REG
    $119 reg: C2_GPIOX_OUTP_REG
    $11A reg: C2_GPIOX_INP_REG
    $13E reg: C2_GPIOX_PUPD_REG
    $14C reg: C2_GPIOX_PUEN_REG

    $10F reg: C2_GPIOY_FSEL_REG
    $110 reg: C2_GPIOY_OUTP_REG
    $111 reg: C2_GPIOY_INP_REG
    $13B reg: C2_GPIOY_PUPD_REG
    $149 reg: C2_GPIOY_PUEN_REG

\    $10C reg: C2_GPIODV_FSEL_REG
\    $10D reg: C2_GPIODV_OUTP_REG
\    $10E reg: C2_GPIODV_INP_REG
\    $148 reg: C2_GPIODV_PUPD_REG
\    $13A reg: C2_GPIODV_PUEN_REG

    $12C reg: C2_MUX_REG_0
    $12D reg: C2_MUX_REG_1
    $12E reg: C2_MUX_REG_2
    $12F reg: C2_MUX_REG_3
    $130 reg: C2_MUX_REG_4
    $131 reg: C2_MUX_REG_5
    $133 reg: C2_MUX_REG_7
    $134 reg: C2_MUX_REG_8

    Create shift/type
    ' 1bit , ' 1bit , ' 1bit , ' 1bit ,
    ' 1bit , ' 2bit , ' 4bit ,

    Variable gpio-dummy

    : gpio>mask ( gpio type table -- shift mask addr )
        third -1 = IF
            2drop drop 0 0 gpio-dummy gpio-base -
            EXIT  THEN
        swap { s/t }
        s/t 2* cells + over 5 rshift cells + @
        swap $1F and shift/type s/t cells + perform
        over lshift rot ;

    -1
    1+ dup Constant fsel#
    1+ dup Constant outp#
    1+ dup Constant inp#
```

```
      1+ dup Constant pupd#
      1+ dup Constant puen#
      1+ dup Constant mux#
      drop

      Create gpio-reg[]
      C2_GPIOX_FSEL_REG  , C2_GPIOY_FSEL_REG  ,
      C2_GPIOX_OUTP_REG  , C2_GPIOY_OUTP_REG  ,
      C2_GPIOX_INP_REG   , C2_GPIOY_INP_REG   ,
      C2_GPIOX_PUPD_REG  , C2_GPIOY_PUPD_REG  ,
      C2_GPIOX_PUEN_REG  , C2_GPIOY_PUEN_REG  ,
      C2_MUX_REG_0       , C2_MUX_REG_4       ,
        DOES> ( gpio type -- shift mask addr )
          gpio>mask gpio-base + ;
      [: lits# 2 u>= IF  2lits> rot >body gpio>mask
              >3lits ]] gpio-base + [[
          ELSE  does,  THEN ;] optimizes gpio-reg[]

      \ pins to GPIO table: X=$000+, Y=$020+
      Create pin>gpio ( pin -- gpio )
      -1   , -1   ,
      -1   , -1   ,
      -1   , -1   ,
      $015 , -1   ,
      -1   , -1   ,
      $013 , $00A ,
      $00B , -1   ,
      $009 , $008 ,
      -1   , $005 ,
      $007 , -1   ,
      $004 , $003 ,
      $002 , $001 ,
      -1   , $02E ,
      -1   , -1   ,
      $000 , -1   ,
      $028 , $02D ,
      $006 , -1   ,
      $023 , $027 ,
      -1   , -1   ,
      -1   , -1   ,
      DOES> swap 1- #39 umin cells + @ ;
        [: lits# 1 u>= IF  >body
            lits> 1- #39 umin cells + @ >lits
          ELSE  does,  THEN ;] optimizes pin>gpio
  [THEN]
  model?" Raspberry Pi" [IF]
      $00200000 Constant GPIO-Base-map

      \ fsel are 3 bits per function,
      \ 000 is input, 001 is output

      $000 reg: RPI_GPFSEL0
      $001 reg: RPI_GPFSEL1
      $002 reg: RPI_GPFSEL2
      $003 reg: RPI_GPFSEL3
      $004 reg: RPI_GPFSEL4
      $005 reg: RPI_GPFSEL5

      $007 reg: RPI_GPSET0
      $008 reg: RPI_GPSET1

      $00A reg: RPI_GPCLR0
      $00B reg: RPI_GPCLR1

      $00D reg: RPI_GPLEV0
      $00E reg: RPI_GPLEV1

      $010 reg: RPI_GPEDS0
      $011 reg: RPI_GPEDS1

      $013 reg: RPI_GPREN0
      $014 reg: RPI_GPREN1

      $016 reg: RPI_GPFEN0
      $017 reg: RPI_GPFEN1

      $019 reg: RPI_GPHEN0
      $01A reg: RPI_GPHEN1

      $01C reg: RPI_GPLEN0
      $01D reg: RPI_GPLEN1

      $01F reg: RPI_GPAREN0
      $020 reg: RPI_GPAREN1

      $022 reg: RPI_GPAFEN0
      $023 reg: RPI_GPAFEN1

      \ BCM2835 variant
      $025 reg: RPI_GPPUD
      $026 reg: RPI_GPPUDCLK0
      $027 reg: RPI_GPPUDCLK1

      \ BCM2711 variant
      $039 reg: RPI_GPIO_PUP_PDN_CNTRL_REG0
      $03A reg: RPI_GPIO_PUP_PDN_CNTRL_REG1
      $03B reg: RPI_GPIO_PUP_PDN_CNTRL_REG2
      $03C reg: RPI_GPIO_PUP_PDN_CNTRL_REG3

      -1
      1+ dup Constant fsel#
      1+ dup Constant set#
      1+ dup Constant clr#
      1+ dup Constant inp#
      model?" Raspberry Pi 4 Model B" [IF]
          1+ dup Constant pudmode#
          drop

          Create shift/type
          ' 3bit , ' 1bit , ' 1bit , ' 1bit , ' 2bit ,

          : rpi-4 ;
      [ELSE]
          1+ dup Constant puclk#
          drop

          Create shift/type
          ' 3bit , ' 1bit , ' 1bit , ' 1bit , ' 1bit ,

          : rpi ; \ model 0-3
      [THEN]

      Variable gpio-dummy

      : gpio>mask ( gpio type table -- shift mask addr )
          third -1 = IF
              2drop drop 0 0 gpio-dummy gpio-base -
              EXIT  THEN
          swap { s/t }
          s/t cells + over 5 rshift cells + @
          swap $1F and shift/type s/t cells + perform
          over lshift rot ;

      Create gpio-reg[]
      RPI_GPFSEL0    ,
      RPI_GPSET0     ,
      RPI_GPCLR0     ,
      RPI_GPLEV0     ,
      model?" Raspberry Pi 4 Model B" [IF]
          RPI_GPIO_PUP_PDN_CNTRL_REG0 ,
      [ELSE]
          RPI_GPPUDCLK0 ,
      [THEN]
        DOES> ( gpio type -- shift mask addr )
          gpio>mask gpio-base + ;
      [: lits# 2 u>= IF  2lits> rot >body gpio>mask
              >3lits ]] gpio-base + [[
          ELSE  does,  THEN ;] optimizes gpio-reg[]

      \ pins to GPIO table:
      Create pin>gpio ( pin -- gpio )
```

```
      -1   , -1   ,
      $002 , -1   ,
      $003 , -1   ,
      $004 , $00E ,
      -1   , $00F ,
      $011 , $012 ,
      $01B , -1   ,
      $016 , $017 ,
      -1   , $018 ,
      $00A , -1   ,
      $009 , $019 ,
      $00B , $008 ,
      -1   , $007 ,
      $000 , $001 ,
      $005 , -1   ,
      $006 , $00C ,
      $00D , -1   ,
      $013 , $010 ,
      $01A , $014 ,
      -1   , $015 ,
      DOES> swap 1- #39 umin cells + @ ;
        [: lits# 1 u>= IF  >body
            lits> 1- #39 umin cells + @ >lits
          ELSE  does,  THEN ;] optimizes pin>gpio
  [THEN]

  \ optimizing helpers

  : lmask!, ( lits:n mode -- ) >r
      lits> pin>gpio r> gpio-reg[]
      rot >lits ]] lshift [[
      gpio-base - >2lits ]] gpio-base + lmask! [[ ;
  : l@, ( lits:n mode -- )  >r
      lits> pin>gpio r> gpio-reg[]
      gpio-base - >2lits ]] gpio-base + l@ and [[
      >lits ]] rshift [[ ;
  : l!, ( lits:n mode -- )  >r
      lits> pin>gpio r> gpio-reg[] rot drop
      gpio-base - >2lits ]] gpio-base + l! [[ ;
  [IFDEF] fsel#
      : fsel! ( val n -- ) pin>gpio fsel#
          gpio-reg[] 2>r lshift 2r> lmask! ;
      opt: lits# 1 u>= IF  drop  fsel# lmask!,
          ELSE  :,  THEN ;
      : fsel@ ( n -- val ) pin>gpio fsel#
          gpio-reg[] l@ and swap rshift ;
      opt: lits# 1 u>= IF  drop  fsel# l@,
          ELSE  :,  THEN ;
  [THEN]
  : pin@ ( pin -- val )
      \G get input level of pin
      pin>gpio inp# gpio-reg[] l@ and swap rshift ;
  opt: lits# 1 u>= IF  drop inp# l@,  ELSE  :,  THEN ;
  [IFDEF] set#
      : pinset ( n -- ) pin>gpio set#
          gpio-reg[] l! drop ;
      opt: lits# 1 u>= IF  drop set# l!,
          ELSE  :,  THEN ;
  [THEN]
  [IFDEF] clr#
      : pinclr ( n -- ) pin>gpio clr#
          gpio-reg[] l! drop ;
      opt: lits# 1 u>= IF  drop clr# l!,
          ELSE  :,  THEN ;
  [THEN]
  : pin! ( val pin -- )
      \G set output level of pin to @var{val}
      [IFDEF] outp#
          pin>gpio outp# gpio-reg[] 2>r
          lshift 2r> lmask! ;
      opt: lits# 1 u>= IF  drop outp# lmask!,
          ELSE  :,  THEN
      [ELSE] [IFDEF] pinset
              swap 1 and IF  pinset  ELSE  pinclr  THEN ;
          opt: lits# 2 u>= IF  drop
                  lits> lits> swap >lits
                  1 and IF  ]] pinset [[
                  ELSE  ]] pinclr [[  THEN
              ELSE
                  lits# 1 u>= IF  drop
                      lits> >r
                      ]] 1 and IF [[ r@ >lits
                          ]] pinset  ELSE [[
                          r> >lits  ]] pinclr  THEN [[
                  ELSE
                      :,
                  THEN
              THEN
      [THEN] [THEN] ;
  [IFUNDEF] pinset
      : pinset ( pin -- )
          \G set pin to high
          1 swap pin! ;
      opt: drop ]] 1 swap pin! [[ ;
  [THEN]
  [IFUNDEF] pinclr
      : pinclr ( pin -- )
          \G set pin to low
          0 swap pin! ;
      opt: drop ]] 0 swap pin! [[ ;
  [THEN]
  [IFDEF] mux#
      : mux! ( val pin -- ) pin>gpio mux#
          gpio-reg[] 2>r lshift 2r> lmask! ;
      opt: lits# 1 u>= IF  drop  mux# lmask!,
          ELSE  :,  THEN ;
      : mux@ ( pin -- val ) pin>gpio mux#
          gpio-reg[] l@ and swap rshift ;
      opt: lits# 1 u>= IF  drop  mux# l@,
          ELSE  :,  THEN ;
  [THEN]
  : input-pin ( pin -- )
      \G set pin mode to input
      [IFDEF] mux!
          0 over mux!  1 swap fsel! ;
      opt: drop lits# 1 u>= IF
              lits> dup
              ]] 0 Literal mux!  1 Literal fsel! [[
          ELSE  ]] 0 over mux!  1 swap fsel! [[  THEN
      [ELSE]
          0 swap fsel! ;
      opt: drop ]] 0 swap fsel! [[
      [THEN] ;
  : output-pin ( pin -- )
      \G set pin mode to output
      [IFDEF] mux!
          0 over mux!  0 swap fsel! ;
      opt: drop lits# 1 u>= IF
              lits> dup
              ]] 0 Literal mux!  0 Literal fsel! [[
          ELSE  ]] 0 over mux!  0 swap fsel! [[  THEN
      [ELSE]
          1 swap fsel! ;
      opt: drop ]] 1 swap fsel! [[
      [THEN] ;
  [IFDEF] puen#
      : puen! ( val pin -- ) pin>gpio puen#
          gpio-reg[] 2>r lshift 2r> lmask! ;
      opt: lits# 1 u>= IF  drop  puen# lmask!,
          ELSE  :,  THEN ;
      : puen@ ( pin -- val ) pin>gpio puen#
          gpio-reg[] l@ and swap rshift ;
      opt: lits# 1 u>= IF  drop  puen# l@,
          ELSE  :,  THEN ;
  [THEN]
  [IFDEF] pupd#
      : pupd! ( val pin -- ) pin>gpio pupd#
          gpio-reg[] 2>r lshift 2r> lmask! ;
      opt: lits# 1 u>= IF  drop  pupd# lmask!,
          ELSE  :,  THEN ;
      : pupd@ ( pin -- val ) pin>gpio pupd#
```

```
520         gpio-reg[] 1@ and swap rshift ;
521     opt: lits# 1 u>= IF  drop  pupd# 1@,
522         ELSE  :,  THEN ;
523 [THEN]
524 [IFDEF] puclk#
525     : puclk! ( val pin -- ) pin>gpio puclk#
526         gpio-reg[] 2>r lshift 2r> lmask! ;
527     opt: lits# 1 u>= IF  drop  puclk# lmask!,
528         ELSE  :,  THEN ;
529     : 150cyc  25 0 DO  LOOP ;
530     \ assumes 5 cycles per iteration
531     : 0<>1 ( pin -- n' )
532         dup 1 rshift swap 1 lshift or 3 and ;
533 [THEN]
534 : pin-resmode ( mode pin -- )
535     \G set pullup/down mode of pin @var{n} to:
536     \G @enumerate 0
537     \G @item none
538     \G @item pullup
539     \G @item pulldown
540     \G @end enumerate
541     [IFDEF] pupd#
542         >r case
543             0 of  0 r> puen!  endof
544             1 of  1 r@ puen!  1 r> pupd!  endof
545             2 of  1 r@ puen!  0 r> pupd!  endof
546             rdrop  endcase ;
547     opt: lits# 2 u>= IF  drop
548             2lits> >r  case
549                 0 of  0 r> >2lits ]] puen! [[  endof
550                 1 of  1 r@ >2lits ]] puen! [[
551                     1 r> >2lits ]] pupd! [[  endof
552                 2 of  1 r@ >2lits ]] puen! [[
553                     0 r> >2lits ]] pupd! [[  endof
554                 rdrop  endcase
555         ELSE  :,  THEN
556     [THEN]
557     [IFDEF] puclk#
558         >r 0<>1 RPI_GPPUD 1!
559         150cyc  1 r@ puclk!  150cyc
560         0 RPI_GPPUD 1!  0 r> puclk! ;
561     opt: lits# 1 u>= IF  drop lits> >r
562             ]] 0<>1 RPI_GPPUD 1!  150cyc  1 [[
563             r@ >lits ]] puclk!  150cyc  0 [[
564             r> >lits ]] puclk!  0 RPI_GPPUD 1! [[
565         ELSE  :,  THEN
566     [THEN]
567 [IFDEF] pudmode#
568     pin>gpio pudmode#
569     gpio-reg[] 2>r lshift 2r> lmask! ;
570     opt: lits# 1 u>= IF  drop  pudmode# lmask!,
571         ELSE  :,  THEN
572 [THEN]
573 ;
574 [IFDEF] pudmode#
575     : pin-resmode@ ( pin -- val )
576         pin>gpio pudmode#
577         gpio-reg[] 1@ and swap rshift ;
578     opt: lits# 1 u>= IF  drop  pudmode# 1@,
579         ELSE  :,  THEN ;
580 [THEN]
581
582 : map-gpio ( -- )
583     s" /dev/gpiomem" r/w open-file throw
584     dup >r fileno >r
585     0 $1000 PROT_READ PROT_WRITE or MAP_SHARED
586     r> GPIO-Base-map mmap
587     r> close-file throw dup 0= ?ior to gpio-base ;
588
589 map-gpio
590
591 : pin-show { mode -- }
592     41 1 DO
593         I pin>gpio dup -1 = IF drop ." -"
594         ELSE  mode gpio-reg[] 1@ and swap rshift 0 .r
595         THEN
596     LOOP ;
597 : .pin#s ( -- ) cr
598     ." pin " 41 1 DO  I 10 / 0 .r  LOOP cr
599     ." pin " 41 1 DO  I 10 mod 0 .r  LOOP ;
600 : .pins ( -- ) .pin#s cr
601     [IFDEF] fsel# ." fsel" fsel# pin-show cr [THEN]
602     [IFDEF] inp#  ." inp " inp#  pin-show cr [THEN]
603     [IFDEF] outp# ." outp" outp# pin-show cr [THEN]
604     [IFDEF] puen# ." puen" puen# pin-show cr [THEN]
605     [IFDEF] pupd# ." pupd" pupd# pin-show cr [THEN]
606     [IFDEF] pudmode# ." pud " pudmode# pin-show cr [THEN]
607     [IFDEF] mux#  ." mux " mux#  pin-show cr [THEN] ;
608 : inps@ ( -- u ) 0 41 1 DO  I pin@ I lshift or  LOOP ;
609
610 : pin-connect? ( pin -- )
611     dup pin>gpio -1 = IF  ." -/-"  drop  EXIT  THEN
612     >r r@ output-pin
613     r@ pinclr  inps@ invert
614     r@ pinset  inps@ and
615     dup 2/ #40 ['] .r 2 base-execute space
616     41 1 DO
617         dup 1 I lshift and  IF  I .  THEN
618     LOOP  drop
619     r> input-pin ;
620
621 : .pin-matrix ( -- )
622     41 1 DO cr I pin-connect? LOOP cr ;
```

spi.fs

```
1   \ spi.fs        SPI access to Microchip EEPROMs
2   \
3   \ Authors: Bernd Paysan
4   \ Copyright (C) 2021 Free Software Foundation, Inc.
5
6   \ This file is part of Gforth.
7
8   \ Gforth is free software; you can redistribute it
9   \ and/or modify it under the terms of the GNU
10  \ General Public License as published by the Free
11  \ Software Foundation, either version 3 of the
12  \ License, or (at your option) any later version.
13
14  \ This program is distributed in the hope that it
15  \ will be useful, but WITHOUT ANY WARRANTY; without
16  \ even the implied warranty of MERCHANTABILITY or
17  \ FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
18  \ General Public License for more details.
19
20  \ You should have received a copy of the GNU General
21  \ Public License along with this program. If not,
22  \ see http://www.gnu.org/licenses/.
23
24  require ./gpios.fs
25  require unix/spi.fs
26
27  : _IOC ( dir type nr size -- constant )
28      >r swap 8 lshift or r> $10 lshift or
29      swap $1E lshift or ;
30  : SPI_IOC_MESSAGE ( n -- constant )
31      >r 1 SPI_IOC_MAGIC 0 r> spi_ioc_transfer * _IOC ;
32  : SPI_IOC_WR_MAX_SPEED_HZ ( -- constant )
33      1 SPI_IOC_MAGIC 4 4 _IOC ;
34  : SPI_IOC_RD_MAX_SPEED_HZ ( -- constant )
35      2 SPI_IOC_MAGIC 4 4 _IOC ;
36
37  s" /dev/spidev0.0" r/w open-file throw Value spi-fd
38
39  [DEFINED] odroid-n2+ [DEFINED] odroid-n2 or [IF]
40      : mux-spi ( -- )
41          #4 #19 mux!  #4 #21 mux!  #4 #23 mux! ;
42  [THEN]
43  [DEFINED] rpi-4 [DEFINED] rpi or [IF]
44      : mux-spi ( -- )
45          #2 #19 fsel!  #2 #21 fsel!  #2 #23 fsel! ;
```

```
[THEN]

#22 Constant wp-pin   \ write protect pin
#24 Constant cs-pin   \ chip select pin
#26 Constant hold-pin \ hold pin
: gpio-spi ( -- )
    hold-pin output-pin
    cs-pin output-pin
    wp-pin output-pin
    hold-pin pinset
    cs-pin pinset
    wp-pin pinset ;
: spioctl ( n buf -- )
    cs-pin pinclr
    >r >r spi-fd fileno r> SPI_IOC_MESSAGE
    r> ioctl cs-pin pinset ?ior ;

[IFUNDEF] alloz
: alloz ( n -- )
    here swap dup allot erase ;
[THEN]

#3000000 constant spi-hz

1 Constant MC-WRSR
2 Constant MC-WRITE
3 Constant MC-READ
4 Constant MC-WRDI
5 Constant MC-RDSR
6 Constant MC-WREN

$20 buffer: pagebuf
Create readbuf  MC-READ  c, 0 w,
Create writebuf MC-WRITE c, 0 w,

Create spi-rd-msgs
spi_ioc_transfer 2* alloz

spi-rd-msgs spi_ioc_transfer + Constant spi-rd-msg2

spi-hz spi-rd-msgs spi_ioc_transfer-speed_hz l!
readbuf spi-rd-msgs spi_ioc_transfer-tx_buf !
2 spi-rd-msgs spi_ioc_transfer-len l!
spi-hz spi-rd-msg2 spi_ioc_transfer-speed_hz l!
pagebuf spi-rd-msg2 spi_ioc_transfer-rx_buf !
8 spi-rd-msgs spi_ioc_transfer-bits_per_word c!
8 spi-rd-msg2 spi_ioc_transfer-bits_per_word c!

: spi-readrest ( len readlen -- )
    spi-rd-msgs spi_ioc_transfer-len l!
    spi-rd-msg2 spi_ioc_transfer-len l!
    2 spi-rd-msgs spioctl ;
: spi-readb ( addr len -- )
    swap readbuf 1+ c!     2 spi-readrest ;
: spi-readw ( addr len -- )
    swap readbuf 1+ be-w!  3 spi-readrest ;
: spi-c@ ( addr -- byte )
    1 spi-readb  pagebuf c@ ;
: spi-w@ ( addr -- word )
    2 spi-readb  pagebuf w@ ;
: spi-l@ ( addr -- long )
    4 spi-readb  pagebuf l@ ;
: spi-x@ ( addr -- extra )
    8 spi-readb  pagebuf x@ ;
: spiw-c@ ( addr -- byte )
    1 spi-readw  pagebuf c@ ;
: spiw-w@ ( addr -- word )
    2 spi-readw  pagebuf w@ ;
: spiw-l@ ( addr -- long )
    4 spi-readw  pagebuf l@ ;
: spiw-x@ ( addr -- extra )
    8 spi-readw  pagebuf x@ ;

Create stbuf MC-RDSR c, 0 c,

Create spi-st-msgs
spi_ioc_transfer 2* alloz

spi-st-msgs spi_ioc_transfer + Constant spi-st-msg2

spi-hz spi-st-msgs spi_ioc_transfer-speed_hz l!
stbuf spi-st-msgs spi_ioc_transfer-tx_buf !
1 spi-st-msgs spi_ioc_transfer-len l!
spi-hz spi-st-msg2 spi_ioc_transfer-speed_hz l!
stbuf 1+ spi-st-msg2 spi_ioc_transfer-rx_buf !
1 spi-st-msg2 spi_ioc_transfer-len l!
8 spi-st-msgs spi_ioc_transfer-bits_per_word c!
8 spi-st-msg2 spi_ioc_transfer-bits_per_word c!

: spi-status@ ( -- status )
    MC-RDSR stbuf c!
    2 spi-st-msgs spioctl
    stbuf 1+ c@ ;
: spi-status! ( status -- )
    MC-WRSR stbuf c!  stbuf 1+ c!
    2 spi-st-msgs spi_ioc_transfer-len l!
    1 spi-st-msgs spioctl
    1 spi-st-msgs spi_ioc_transfer-len l! ;

: spi-wren ( -- )
    MC-WREN stbuf c!
    1 spi-st-msgs spioctl ;
: spi-wrdi ( -- )
    MC-WRDI stbuf c!
    1 spi-st-msgs spioctl ;

: spi-wip| ( -- )
    BEGIN  spi-status@ 1 and 0=  UNTIL ;

Create spi-wr-msgs
spi_ioc_transfer 2* alloz

spi-wr-msgs spi_ioc_transfer + Constant spi-wr-msg2

spi-hz spi-wr-msgs spi_ioc_transfer-speed_hz l!
8 spi-wr-msgs spi_ioc_transfer-bits_per_word c!
writebuf spi-wr-msgs spi_ioc_transfer-tx_buf !
spi-hz spi-wr-msg2 spi_ioc_transfer-speed_hz l!
8 spi-wr-msg2 spi_ioc_transfer-bits_per_word c!
pagebuf spi-wr-msg2 spi_ioc_transfer-tx_buf !

: spi-writerest ( len writelen -- )
    spi-wr-msgs spi_ioc_transfer-len l!
    spi-wr-msg2 spi_ioc_transfer-len l!
    2 spi-wr-msgs spioctl spi-wip| ;
: spi-writeb ( addr len -- )
    swap writebuf 1+ c!     2 spi-writerest ;
: spi-writew ( addr len -- )
    swap writebuf 1+ be-w!  3 spi-writerest ;
: spi-c! ( byte addr -- )
    swap pagebuf c!  1 spi-writeb ;
: spi-w! ( word addr -- )
    swap pagebuf w!  2 spi-writeb ;
: spi-l! ( long addr -- )
    swap pagebuf l!  4 spi-writeb ;
: spi-x! ( extra addr -- )
    swap pagebuf x!  8 spi-writeb ;
: spiw-c! ( byte addr -- )
    swap pagebuf c!  1 spi-writew ;
: spiw-w! ( word addr -- )
    swap pagebuf w!  2 spi-writew ;
: spiw-l! ( long addr -- )
    swap pagebuf l!  4 spi-writew ;
: spiw-x! ( extra addr -- )
    swap pagebuf x!  8 spi-writew ;
```

i2c.fs

```
\ i2c.fs        I²C access to Microchip EEPROMs
\
\ Authors: Bernd Paysan
```

```
\ Copyright (C) 2021 Free Software Foundation, Inc.

\ This file is part of Gforth.

\ Gforth is free software; you can redistribute it
\ and/or modify it under the terms of the GNU
\ General Public License as published by the Free
\ Software Foundation, either version 3 of the
\ License, or (at your option) any later version.

\ This program is distributed in the hope that it
\ will be useful, but WITHOUT ANY WARRANTY; without
\ even the implied warranty of MERCHANTABILITY or
\ FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
\ General Public License for more details.

\ You should have received a copy of the GNU General
\ Public License along with this program. If not,
\ see http://www.gnu.org/licenses/.

require ./gpios.fs
require unix/i2c.fs

s" /dev/i2c-1" r/w open-file throw Value i2c-fd

[DEFINED] odroid-n2+ [DEFINED] odroid-n2 or [IF]
    : mux-i2c-0 ( -- )
        #1 #3 mux! #1 #5 mux! ;
    : mux-i2c-1 ( -- )
        #2 #27 mux! #2 #28 mux! ;
[THEN]
[DEFINED] rpi-4 [DEFINED] rpi or [IF]
    : mux-i2c-0 ( -- )
        #2 #3 fsel!  #2 #5 fsel! ;
    : mix-i2c-1 ( -- )
        #2 #27 fsel!  #2 #28 fsel! ;
[THEN]

#29 Constant wren-pin
: wren ( -- ) \ set write enable
    wren-pin output-pin  wren-pin pinset ;
: i2ctl ( msgs n -- )
    { | msgbuf[ i2c_rdwr_ioctl_data ] }
    msgbuf[ i2c_rdwr_ioctl_data-nmsgs l!
    msgbuf[ i2c_rdwr_ioctl_data-msgs !
    i2c-fd fileno I2C_RDWR msgbuf[ ioctl ?ior ;

[IFUNDEF] alloz
    : alloz ( n -- )
        here swap dup allot erase ;
[THEN]

$12 buffer: i2c-writebuf
\ 1 or 2 bytes command, rest write buffer
$10 buffer: i2c-readbuf
i2c_msg buffer: i2c-writemsg
i2c-writebuf i2c-writemsg i2c_msg-buf !

i2c_msg 2* buffer: i2c-readmsgs
i2c-readmsgs i2c_msg + Constant i2c-readmsg2

i2c-writebuf i2c-readmsgs i2c_msg-buf !
i2c-readbuf i2c-readmsg2 i2c_msg-buf !
I2C_M_RD i2c-readmsg2 i2c_msg-flags w!

: i2c-addr ( addr -- )
    \G specify device address
    dup i2c-writemsg i2c_msg-addr w!
    dup i2c-readmsgs i2c_msg-addr w!
    i2c-readmsg2 i2c_msg-addr w! ;

: i2c-readb ( cmd len -- )
    swap i2c-writebuf c!
    1 i2c-readmsgs i2c_msg-len w!
    i2c-readmsg2 i2c_msg-len w!
    i2c-readmsgs 2 i2ctl ;
: i2c-readw ( cmd len -- )
    swap i2c-writebuf be-w!
    2 i2c-readmsgs i2c_msg-len w!
    i2c-readmsg2 i2c_msg-len w!
    i2c-readmsgs 2 i2ctl ;

: i2c-c@ ( cmd -- byte )
    1 i2c-readb  i2c-readbuf c@ ;
: i2c-w@ ( cmd -- word )
    2 i2c-readb  i2c-readbuf w@ ;
: i2c-l@ ( cmd -- long )
    4 i2c-readb  i2c-readbuf l@ ;
: i2c-x@ ( cmd -- extra )
    8 i2c-readb  i2c-readbuf x@ ;
: i2cw-c@ ( cmd -- byte )
    1 i2c-readw  i2c-readbuf c@ ;
: i2cw-w@ ( cmd -- word )
    2 i2c-readw  i2c-readbuf w@ ;
: i2cw-l@ ( cmd -- long )
    4 i2c-readw  i2c-readbuf l@ ;
: i2cw-x@ ( cmd -- extra )
    8 i2c-readw  i2c-readbuf x@ ;

: i2c-wip| ( -- )
    BEGIN  0 ['] i2c-c@ catch  WHILE  drop  REPEAT  drop ;

: i2c-writeb ( cmd len -- )
    swap i2c-writebuf c!
    1+ i2c-writemsg i2c_msg-len w!
    i2c-writemsg 1 i2ctl  i2c-wip| ;
: i2c-writew ( cmd len -- )
    swap i2c-writebuf be-w!
    2 + i2c-writemsg i2c_msg-len w!
    i2c-writemsg 1 i2ctl  i2c-wip| ;

: i2c-c! ( byte cmd -- )
    swap i2c-writebuf 1+ c!  1 i2c-writeb ;
: i2c-w! ( word cmd -- )
    swap i2c-writebuf 1+ w!  2 i2c-writeb ;
: i2c-l! ( long cmd -- )
    swap i2c-writebuf 1+ l!  4 i2c-writeb ;
: i2c-x! ( extra cmd -- )
    swap i2c-writebuf 1+ x!  8 i2c-writeb ;
: i2cw-c! ( byte cmd -- )
    swap i2c-writebuf 2 + c!  1 i2c-writew ;
: i2cw-w! ( word cmd -- )
    swap i2c-writebuf 2 + w!  2 i2c-writew ;
: i2cw-l! ( long cmd -- )
    swap i2c-writebuf 2 + l!  4 i2c-writew ;
: i2cw-x! ( extra cmd -- )
    swap i2c-writebuf 2 + x!  8 i2c-writew ;
```

Interview mit Wolf Wejgaard, dem Entwickler von Holon

Wolf Wejgaard und Ulrich Hoffmann

Holon ist ein bemerkenswertes Programmiersystem. Quellcode organisiert Holon strukturiert in einer Datenbank. So kann man auch bei großen Projekten einen guten Überblick behalten. Wolf Wejgaard hat Holon in den 1980ern entwickelt, damals noch unter DOS. Heute läuft Holon auf modernen Computern. Ulrich Hoffmann hatte im Frühjahr am Rande einer Video–Konferenz die Gelegenheit, mit Wolf Wejgaard über Holon, seine Ideen und Hintergründe zu sprechen.

Hallo Wolf, ich freue mich, dass wir dieses Interview führen können. Ich möchte Dir gerne ein paar Fragen zu Holon, HolonForth, seinen Eigenschaften und auch zu seiner Geschichte stellen. Ja? Fangen wir also gerne an, wenn Du bereit bist.

uho: Wolf, wenn Du in einem Satz beschreiben solltest, was Holon ist, was würdest Du sagen?

ww: Holon bezeichnet etwas, das gleichzeitig ein Ganzes ist und Teil eines höheren Ganzen.

uho: So wie Programme andere Programme enthalten können und auch wiederum selbst in größeren Programmen enthalten sein können, ja? Du hast diese Ideen mit HolonForth auf Programmentwicklungs–Systeme angewendet. Was hat Dich auf die Holon–Idee gebracht, Quellcode in einer Datenbank zu speichern? Wann war das ungefähr?

ww: Als „BYTE Magazine“ die zwei Ausgaben zu Forth [2] und Smalltalk [3] publizierte (1980/81), hatte ich mein erstes Forth–Projekt als Freelancer abgeschlossen und dabei Forth gelernt. Worte bilden war relativ einfach, normale Arbeit. Worte verwalten dagegen nicht so leicht. Weder Screens noch Files überzeugten. Mit Smalltalk kam der Wunsch, Forth in einer Browser–Hierarchie zu behandeln. Doch Forth ist nicht objektorientiert — was ist die natürliche Struktur von Forth Worten — gibt es eine?

1988 hatte ich die Lösung: Forth besteht aus Worten, warum nicht auch so behandeln? Ich versuchte ein Screens–Projekt mit je einem Wort pro Screen, erweiterte den Screen–Editor um die Verwaltung der Screens, jede mit dem Namen ihres Wortes bezeichnet, konnte also Worte leicht herumschieben und gruppieren. Das war der Beginn der Datenbank.

uho: Und das Ganze hast Du dann Holon bzw. HolonForth genannt. Woher kommt der Name Holon?

ww: Der universelle Schriftsteller Arthur Koestler beschäftigte sich u. a. mit der Frage, wie die Natur ihre Komplexität im Griff hat, und sah, dass die lebendige Welt aus Teilen besteht, die gleichzeitig ein Ganzes sind [4]. Er nannte die Teile *Holons*. Angefangen bei den Elementarteilchen die ganze Skala hinauf: Atome, Moleküle, Zellen, Organe, Körper, Familien, Gesellschaften. Das Prinzip durchzieht die Welt, inklusive Forth–Worte. Der Name Holon ist zusammengesetzt aus den griechischen Worten „holos“ für Ganzes und der Endsilbe „–on“ für Teilchen (Elektron . . .).

uho: Gibt es eine Haupt–Eigenschaft der Holon–Systeme? Wenn ja, welche ist das?

ww: Holon verwaltet den Source–Code in einer Datenbank und bearbeitet ihn in einem Browser als CMS. Und ersetzt den klassischen File–Editor. Die Datenbank hält stets aktuelle Sourcefiles als Pipes für den Compiler. Der Editor bildet das Dictionary, der Compiler setzt den Code ein beim Laden.

Im Holon–System ist damit das Dictionary dauernd sichtbar. Unabhängig davon, ob die Worte geladen sind oder nicht. Top–Down–Design, Bottom–Up–Entwicklung im Holon–Browser.

uho: Wie hast Du die erste Version von Holon entwickelt?

ww: Das Projekt mit Screen–Editor und einem Wort pro Screen ging besser als jedes Projekt zuvor.

Ich konnte endlich Leo Brodie's Idee verwirklichen, Programme als ein Buch zu schreiben (Thinking Forth [5]). Es brauchte nur noch zwei Teile:

1. Einen Browser, der das Programm als eine Struktur von Chaptern, Sections und Units zeigt. Im Gegensatz zu Brodie, der den Screen als Unit sah, war nun das Wort der elementare Teil des Programms. (Brodie hat Recht mit einem Wort pro Screen.)
2. Ein erweitertes Dictionary mit Headern auch für Sections und Chaptern. Also eine Datenbank.

Ich entwickelte Holon in HS–Forth (James Callahan, Harvard Softworks [6]) wegen seiner Module für Grafik und DB. Diese brachten genügend Anregung für eine eigene Lösung. Von F–PC übernahm ich die Aufteilung des verfügbaren DOS–Speichers in CODE 64–kB–CPU und LIST für Colon–Worte.

uho: Die DOS–Versionen von Holon haben dann weite Verbreitung gefunden. Welche Versionen gab es da und was zeichnet sie aus?

ww: Holon86 für DOS–IDE für DOS–Rechner, heute Embedded PC104. Erfüllt meine Wünsche für ein Entwicklungssystem: **Klarheit und Sofortheit**. Klares Programm, direkte Änderungen im laufendem Code.

Holon11 für 68HC11 — als Shareware verbreitet — in Fachschulen verwendet wegen dem direkten Kontakt zum Code. Bspw. Debuggen schrittweise durch den Code mit

Anzeige der CPU–Register bzw. Returnstack — voll ausgebautes System mit Application–Notes.

Holon68k 32–Bit, Thomas Beierlein schrieb den Assembler. Martin Kreier baute darauf ein komplettes Betriebssystem.

HolonJ mit Forth zu JVM–Compiler — Motiv: Holon auf den Desktop bringen, überall. Java–Bibliotheken waren mühsam. Die HolonJ–Beispiele funktionieren noch in Win XP.

HolonIX entwickelt für Delta–T/DESY — letztlich nicht zum Einsatz gekommen.

Auch Varianten für 6502, 8080 u. a. m. in Kurzform.

Außerdem **HolonX** als universeller Source–Browser mit Wort–Zustands–Angaben = change control. Basierend auf Diskussionen mit Howerd Oakford.

uho: Oh — das sind ja eine Menge Systeme. Folgen sie alle der Holon–Idee? Wie hast Du dabei den Überblick behalten? Wenn sich in einem System Verbesserungen ergeben haben, wie haben die anderen davon profitiert?

ww: Alle Systeme sind von Host86 ausgegangen und hatten dadurch die bewährten Grundfunktionen. Jeweils wurden der Zielassembler und Targetkontakt angepasst. Vielleicht auch mehr — ist lange her.

uho: Neuere Versionen von Holon laufen auch unter Windows, Linux und MacOS. Wie unterscheiden Sie sich von den DOS–Versionen?

ww: Die aktuellen GUI–Versionen sind in Tcl/Tk programmiert, statt in x86 und sind, wie auch ihre Apps, unverändert auf jedem Desktop lauffähig (32– und 64–Bit).

uho: Die Web–Seite, auf der Du über Holon berichtest, heißt `holonforth.com` [1]. In welchem Verhältnis stehen Holon und Forth?

ww: Das Konzept Holon beschreibt das Wesen von Forth: die intrinsische Hierarchie von Worten. Holon ist universell, Forth ist auf Software bezogen.

uho: Kann ich Holon also nur verwenden, um damit Forth–Anwendungen zu machen?

ww: Nein, Holon–IDEs sind universell, weil sie Worte verwalten und Software aus Worten besteht. Das ist auf viele andere Sprachen und Projekte anwendbar. Ich habe Holon verwendet mit HTML, CSS, Javascript, PHP und vor allem Tcl/Tk.

Worte im Sinne von Units = Definitionen, die einen Namen als Kennzeichnung (Identifier) haben, wie Forth–Worte.

HolonH hat einen Webcode–Compiler eingebaut, für HTML und CSS. Ich verwalte damit die `holonforth.com` Website mit inzwischen ca. 150 Seiten.

Außerdem programmierte ich für ein befreundetes Schmuckatelier einen Webshop, in dem man seinen eigenen Schmuck entwerfen kann aus einer Reihe von Ringtypen und Steinen in div. Variationen. Das war damals, 2013, neu (`https://egocollection.ch`). Die Buchstruktur von Holon bewährte sich bestens, konventionell mit Textfile–Editor hätte ich es nicht geschafft.

uho: Du sagst, Quellcode muss in einer Datenbank gespeichert werden. Welche Vorteile bringt das mit sich?

ww: Holon bietet Content–Management, Übersicht, intrinsisch Hypertext, beliebige Struktur, Browser, vor allem als Buch, unabhängig vom Compiler.

Forth muss das Dictionary erst kompilieren, um diese zentrale Liste verfügbar zu haben und bekommt es daher nur für Worte, die schon kompilierbar sind.

Weitere Features: selektives Laden von Worten, Top–Down– und Bottom–Up–Entwicklung, Platz für Revisions–Notizen, Generierung von reinen Sourcefiles ohne Kommentare.

Quellcode ist eine Sammlung von Definitionen und anderen benannten Teilen. Warum speichern wir im normalen Leben unsere Sammlungen in Datenbanken und nicht in Textfiles?

uho: Wie strukturierst Du den Quellcode in der Datenbank?

ww: Wie ein Buch. Kapitel sind Files. Sections fassen Gedanken zusammen. Quellcode ist Wissen. Bücher haben sich bewährt, um Wissen zu fassen und zu vermitteln.

uho: Du hast das Holon–Prinzip „ein Wort, eine Einheit (Unit)“ erwähnt. Warum sollte man nicht mehrere Funktionen/Worte in einer Unit definieren?

ww: Für Holon ist jede Programmeinheit mit einem identifizierenden Namen eine Unit.

Wenn wir einzelne Units verwalten und jeden Namen als Header führen, haben wir mit der Suchfunktion automatisch Hypertext; brauchen also keine Verweise einzubauen. Ein globaler Namensraum hilft auch.

Namen können beliebig lang sein. Das ist einfacher, als Wortlisten zu verwalten und zum Finden eines Wortes dann den Namen der Wortliste und den Namen des Wortes eingeben zu müssen. Bzw. immer zu wissen, in welcher Wortliste wir gerade sind.

Der Hyperlink ist in Holon ein Suchen nach einem Namen im Dictionary. Das ist eindeutig, weil Holon eindeutige Namen verwendet in einem universellen Namensraum. Also keine separaten Wortlisten. Beim Klick auf ein Wort erkennt der Editor den Text und sucht den Namen.

⇒ Es gibt nichts zu pflegen ...

Nach meiner Erfahrung gibt es nur eine Gruppe von Namen, die wiederverwendet werden müssen: Messages auf Methoden einer Klasse bzw. Objekttyp in Holon.

Holon lässt dazu das jeweilige Objekt die zu der gegebenen Message passende Methode finden. Bzw. jeder Objekttyp hat seine private Zuordnung (Tabelle) von Messages zu Methoden.

uho: Wenn also der Quellcode der Anwendung in Holon verwaltet wird, wie entsteht daraus dann die ausführbare Anwendung?

ww: Chapter, deren Name einen Filetyp angehängt haben, bilden ein File mit diesem Namen und schreiben den Code der Units des Chapters auf das File. Nach jeder Änderung von Code wird das File aufgefrischt. Files sind also jederzeit im Einklang mit dem Holon–Browser. Bereit als Pipes für den Compiler.

Die Quell–Files werden in jeder Session neu generiert bzw. überschrieben. Es genügt daher, für ein Projekt die Datenbank (`.hdb`) zu speichern und zu archivieren.

uho: Welche Rolle spielt die interaktive Programmentwicklung in Holon?

ww: Beim Programmieren habe ich zwei Punkte schätzengelernt: Klarheit und Sofortheit (Clarity and Immediacy).

Das Buch macht lesbar und klar. Code austauschen macht immediate. Mit Holon86 in DOS–Zeiten, als Reloaden Minuten brauchte, hatte ich einen Edit/Compile/Test–Zyklus von Sekunden.

uho: Das heißt, man kann in Holon das Ziel–System auch interaktiv programmieren? Wie findet da die Kommunikation zwischen dem Host Holon und der Anwendung, dem Ziel–System, statt?

ww: Wenn Host und Target im selben Rechner liegen, wird ein Hilfs–File `holon.mon` verwendet. Host und Target haben je einen Monitor, der das File überwacht, es liest, wenn es sich ändert und Resultate hineinschreibt.

Wenn Host und Target getrennt sind, funktioniert der Austausch über einen Kanal. Seriell bei HolonDOS–Versionen, per Socket implementiert in HolonT.

uho: Gibt es auch einen Rückkanal vom Ziel–System zurück zu Holon?

ww: `holon.mon` geht beide Wege. Der Host schreibt und wartet, ob/bis sich das File ändert.

uho: Der Holon–Quellcode–Browser erinnert an Smalltalk. Hast Du daher Deine Inspiration? Hast Du auch mit Smalltalk gearbeitet?

ww: Smalltalk war die große Inspiration. Nach den BYTE–Ausgaben 1980/81 mit Forth und Smalltalk träumte ich von Forth in einem solchen Browser.

uho: Gibt es neben dem Holon–Quellcode–Browser auch noch andere Möglichkeiten, den Quellcode anzusehen, und warum sollte man sie nutzen?

ww: Holon hat einen *Linear View*, der das Programm fortlaufend von Anfang bis Ende zeigt und mit dem Browser synchron läuft. Nützlich, um Worte mit ihrer Umgebung zu betrachten.

Die Chapter–Files können als gewohntes Format nützlich sein, je im gewohnten File–Editor. Chapter–File und Datenbank entsprechen sich immer.

uho: Wie kann ich schon vorhandenen Quellcode mit Holon bearbeiten?

ww: Holon kann vorhandenen (legacy) Quelltext importieren mit Markierung als `.hml` Files. Holon verwendet das Format auch zum Austausch zwischen Holon–Projekten. Der Markup ist in HolonCode beschrieben und wohl am besten an einem exportierten File zu verstehen.

uho: Was ist TclForth und in welcher Beziehung steht es zu Holon?

ww: TclForth begann als ein Forth–zu–Tcl–Crosscompiler in HolonT. Es machte dann Sinn, die Systeme zu trennen und TclForth allgemein verfügbar zu halten. Forth und Tcl passen gut zusammen.

uho: Du hast viele Holon–Versionen, insbesondere die aktuelle Version, HolonCode, auf GitHub (`https://github.com/wejgaard/HolonCode`) unter der GPL bereitgestellt. Vielen Dank dafür. Wie sind deine zukünftigen Pläne mit Holon?

ww: Aktuell ein Projekt mit eForth und Dr. Ting's Overview. Passt bestens in Holon, nutzt den ganzen Browser mit Unit–, Section– und Chapter–Kommentaren.

uho: Wo kann ich noch mehr über Holon erfahren?

ww: Auf der Holon–Web–Seite `www.holonforth.com` [1] und in meinen GitHub–Repositories [9]. Über die Entwicklung von Holon habe ich auch schon in der „Vierten Dimension“ geschrieben [7][8].

uho: Gibt es etwas, was Du unseren Lesern mit auf den Weg geben möchtest?

ww: Keep it simple =

„Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.“

Antoine de Saint–Exupéry, Airman's Odyssey

Vielen Dank, Wolf, für dieses interessante Gespräch. Ich glaube, es lohnt sich, dieses außergewöhnliche System genauer anzusehen.

Referenzen

[1] HolonForth, `https://www.holonforth.com`

[2] Byte–Magazin, Forth–Ausgabe, `https://archive.org/details/byte-magazine-1980-08`

[3] Byte–Magazin, Smalltalk–Ausgabe, `https://archive.org/details/byte-magazine-1981-08`

[4] „The Ghost in the Machine“, Koestler, A. Hutchinson. (1979).

[5] „Thinking Forth“, Leo Brodie, `http://thinking-forth.sourceforge.net/`

[6] Brian Fox Erweitert HSForth für DOS `https://github.com/bfox9900/HsF2012`

[7] „Die Entwicklung von HolonForth — Teil 1“, Wolf Wejgaard, in Vierte Dimension 3+4/1997,

[8] „Die Entwicklung von HolonForth — Teil 2“, Wolf Wejgaard, in Vierte Dimension 2/1998,

[9] `https://github.com/wejgaard`

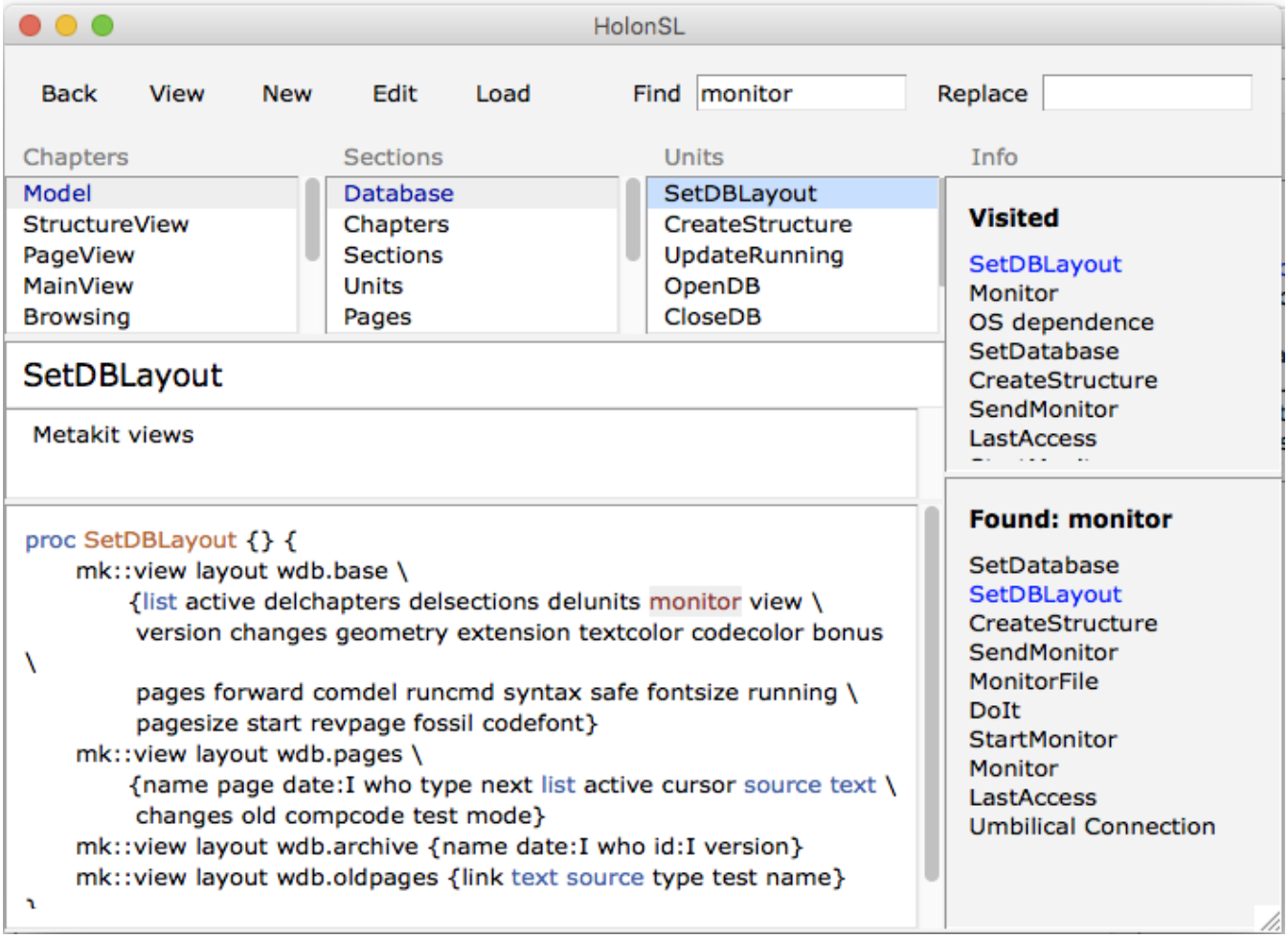


Abbildung 1: Aktuelle Holon–Versionen laufen als Tcl/Tk–Apps in Windows, macOS und Linux (2021).

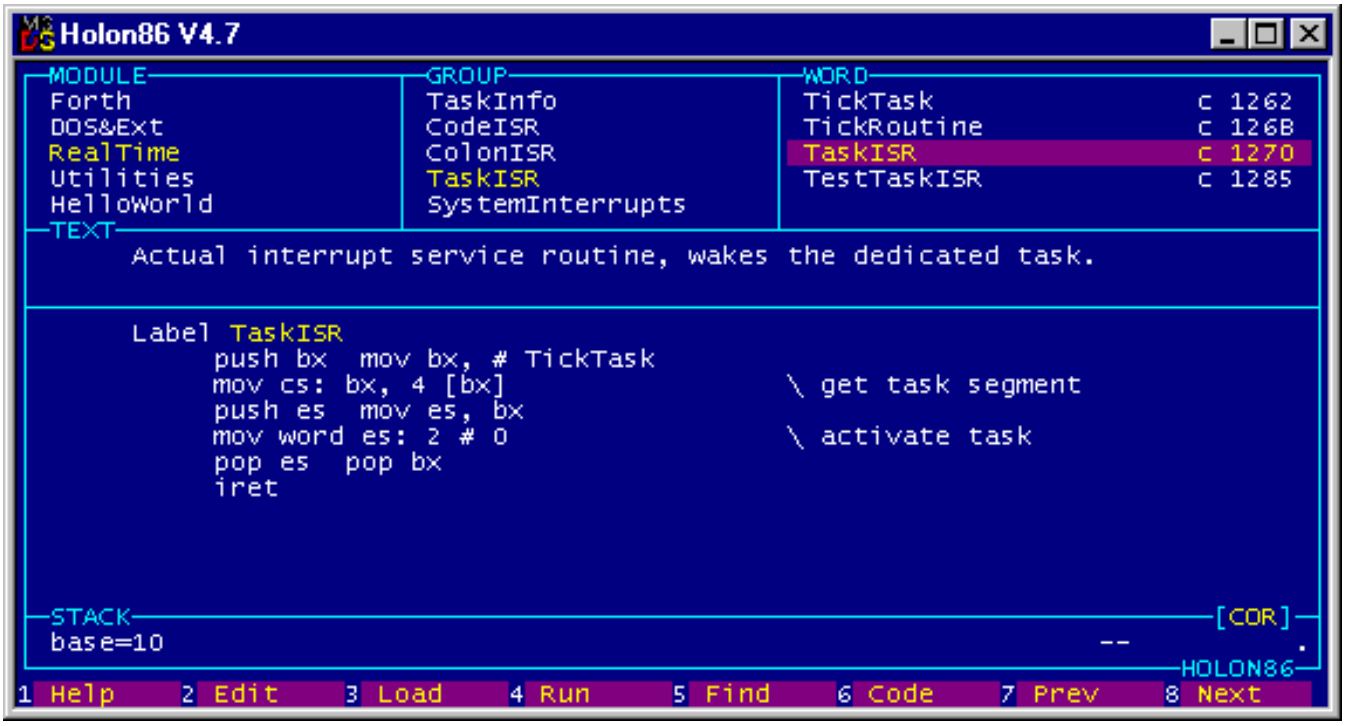


Abbildung 2: Holon86 ist eine text–basierte MS–DOS–App (1998).

Forth–Gruppen regional

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

München **Bernd Paysan**
Tel.: (0 89) – 41 15 46 53
`bernd@net2o.de`
Treffen: Jeden 4. Donnerstag im Monat um 19:00 in der Pizzeria La Capannina, Weitlstr. 142, 80995 München (Feldmochinger Anger).

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

Ruhrgebiet **Carsten Strotmann**
`ruhrpott-forth@strotmann.de`
Treffen alle 1–2 Monate im Unperfekthaus Essen
`http://unperfekthaus.de`.
Termine unter: `https://www.meetup.com/Essen-Forth-Meetup/`

Dienste der Forth–Gesellschaft

Nextcloud `https://cloud.forth-ev.de`

Github `https://github.com/forth-ev`

Twitch `https://www.twitch.tv/4ther`

μP–Controller Verleih **Carsten Strotmann**
`microcontrollerverleih@forth-ev.de`
`mcv@forth-ev.de`

Spezielle Fachgebiete

Forth-Hardware in VHDL microcore (uCore)	**Klaus Schleisiek** Tel.: **(0 58 46) – 98 04 00 8**$_p$ kschleisiek@freenet.de
KI, Object Oriented Forth, Sicherheitskritische Systeme	**Ulrich Hoffmann** Tel.: **(0 41 03) – 80 48 41** `uho@forth-ev.de`
Forth–Vertrieb volksFORTH ultraFORTH RTX / FG / Super8 KK–FORTH	Ingenieurbüro **Klaus Kohl–Schöpe** Tel.: **(0 82 66)–36 09 862**$_p$

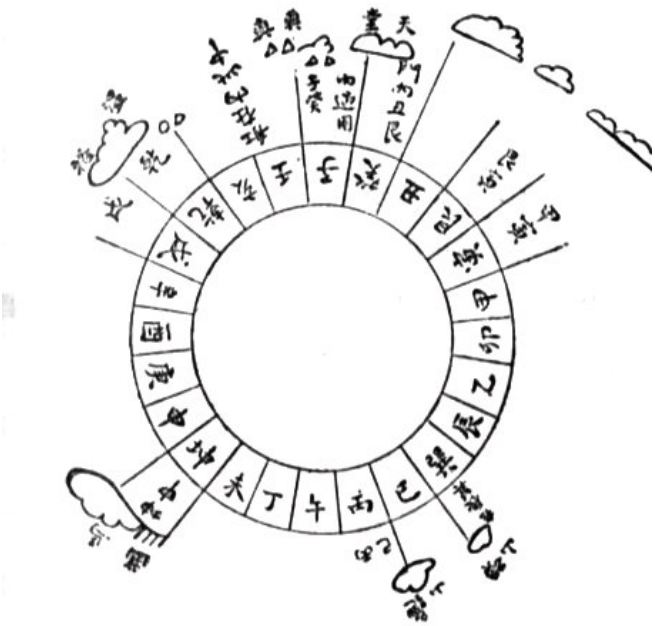
Termine

Donnerstags ab 20:00 Uhr
Forth–Chat net2o `forth@bernd` mit dem Key `keysearch kQusJ`, voller Key:
`kQusJzA;7*?t=uy@X}1GWr!+0qqp_Cn176t4(dQ*`

Montags ab 20:30 Uhr
Forth lernen
Videotreffen (nicht nur) für Forthanfänger
Info und Teilnahmelink: E–Mail an `wost@ewost.de`

Jeder 2. Samstag im Monat
ZOOM–Treffen der Forth2020 Facebook–Gruppe
Infos zur Teilnahme: `www.forth2020.org`

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



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

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

Tagungen 2021

Mitteilung des Direktoriums

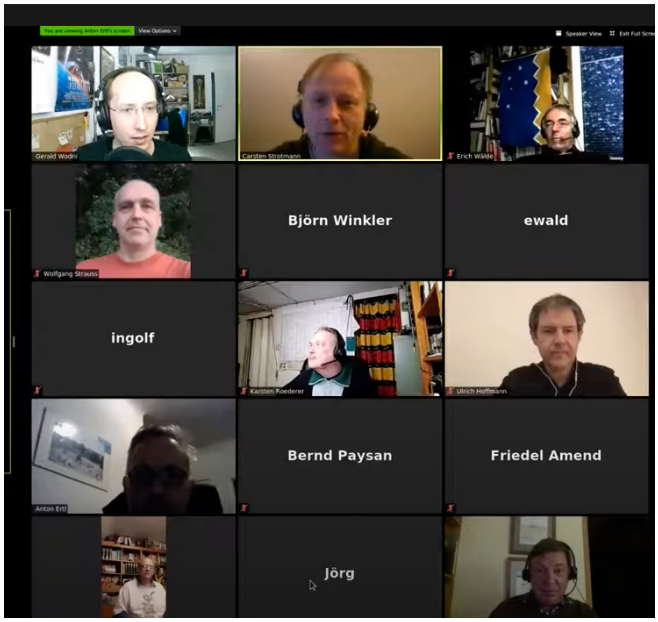
Im letzten Heft hatten wir unsere Mitglieder dazu aufgerufen, ihre Meinung mitzuteilen, in welcher Form nun das Jahrestreffen gewünscht wird. Zur EuroForth hingegen gab es keine Umfrage, da hatte sich der Veranstalter schon entschieden.

Jahrestagung der Forthgesellschaft mit Mitgliederversammlung — online

Liebe Mitglieder, vielen Dank für Eure Teilnahme an der Abstimmung über die Tagung und MV 2021. Das Ergebnis ist eindeutig, die meisten Mitglieder, welche an der Abstimmung teilgenommen haben, wünschen sich eine Online–Tagung mit Mitgliederversammlung. Hier die Auszählung:

- Klassische „In Person"–Tagung mit MV (wenn zulässig): 4 (+7 vielleicht)
- Kombinierte Online– und „In Person"–Tagung mit MV (wenn zulässig): 2 (+7 vielleicht)
- Tagung mit MV nur online via Videokonferenz: 10 (+1 vielleicht)
- Weder Tagung noch MV in 2021, wir warten auf 2022: 2 (+0 vielleicht)

Das Direktorium wird nun diese Online–Tagung mit MV vorbereiten. Um die Tagung nicht zu sehr in zeitlicher Nähe der EuroForth (s. unten) durchzuführen, wird die Tagung und MV 2021 am Wochenende **6. – 7. November 2021** stattfinden. Das Direktorium macht sich an die Arbeit, insbesondere, wie wir Abstimmungen und Wahlen rechtssicher abhalten können.



Das Direktorium wird neben Vorträgen auch Räume für Diskussionen und lockere Gespräche vorbereiten. Wenn Ihr Ideen und Vorschläge habt, aber auch Anträge zur Tagesordnung der Mitgliederversammlung, dann sendet diese bitte ab sofort und so früh wie möglich an `direktorium@forth-ev.de` oder per Brief an die Büroadresse der Forth–Gesellschaft.

Im nächsten Heft, voraussichtlich Ende September oder Anfang Oktober diesen Jahres, wird es dann die Einladung und weitere Informationen zur Tagung und zur MV geben.

Wir freuen uns auf rege Rückmeldungen.

Euer Direktorium

Ullrich Hoffmann, Bernd Paysan, Carsten Strotmann

EuroForth — Roma or Online

The 37th EuroForth conference takes place in Rome/Italy from 10. to 12. September 2021. If travel is still an issue due to COVID–19, the conference will be online like last year on the same date.

Please see the official call for papers[1] for instructions on how to submit papers.

The conference will be preceded by the Forth Standards Meeting which starts on September, 2nd.

Both, meeting and conference, will be hosted in the *SHG Hotel Portamaggiore*.

The hotel is within walking distance to Rome's main train station "Roma Termini".

The registration will be announced in July on the EuroForth website: `https://euro.theforth.net/2021`

[1] `http://www.euroforth.org/ef21/cfp.html`