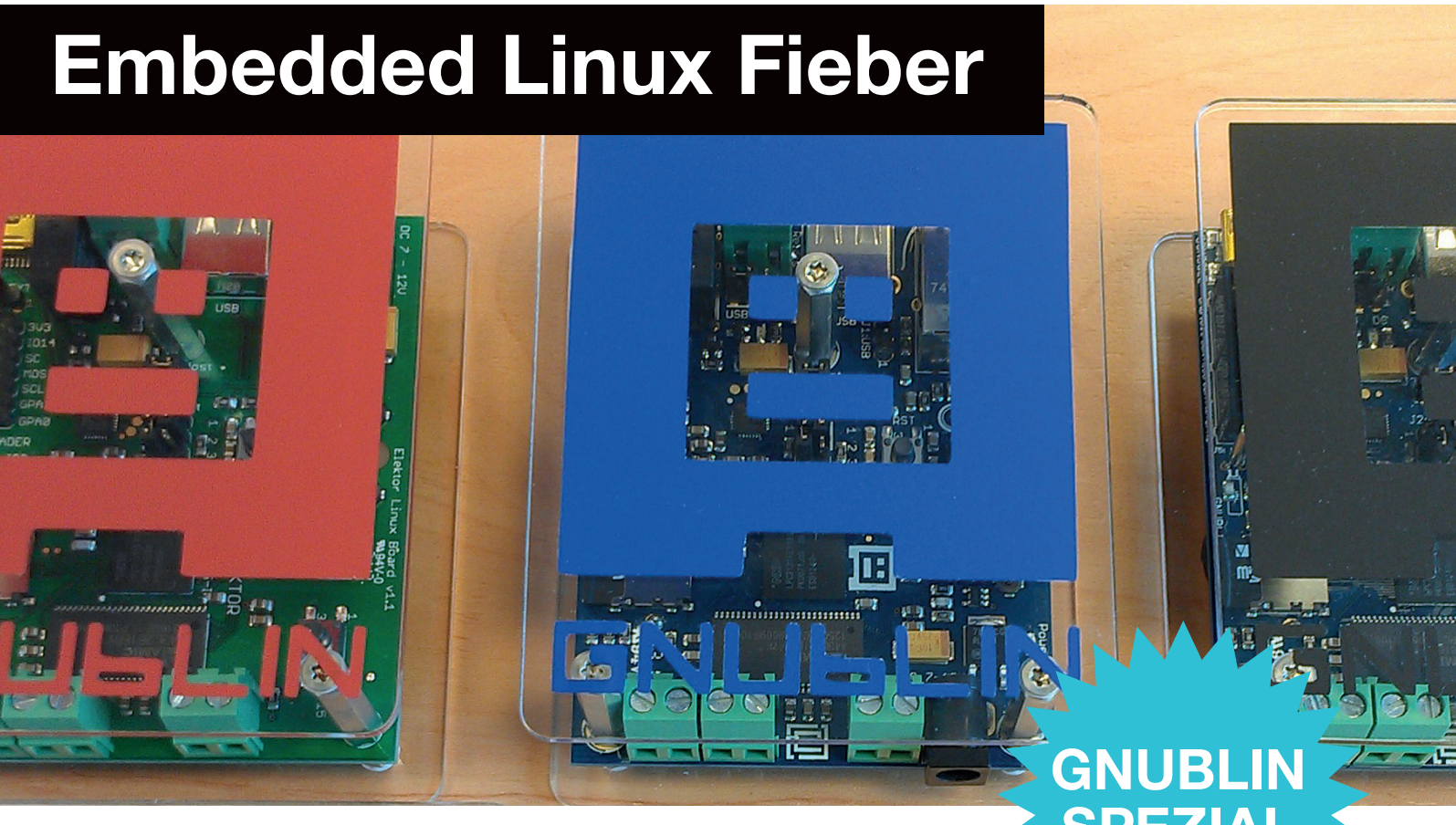
embedded projects JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Eine Open-Source Zeitschrift zum Mitmachen!

Embedded Linux Fieber



GNUBLIN SPEZIAL

[PROJECTS]

- irCube - Infrarot per GPIO
- 3D-Drucker mit Gnublin
- NFC mit Gnublin
- C++ API für Gnublin mit Phyton
- Interrupt mehrer GPIOs
- Jägermeister-Uhr
- Gnublin für die Automatisierung
- Gnublin-Case

Einleitung

Ausgabe 04/2013

Embedded Projects Journal - Ausgabe No. 19

Einleitung

Hallo liebe Leser,

die Idee zu einem Gnublin Sonderheft kam uns, nachdem im vergangenen Sommersemester wieder einige interessante Projekte im Fach „Embedded Linux“ an der Hochschule Augsburg zustande kamen. Ein paar dieser Projekte könnt Ihr Euch in diesem Heft genauer anschauen (neben ein paar „externen“ Artikeln).

Dass wir mal so weit kommen würden, war nicht von Anfang an klar. Ich kann mich noch gut an meinen Wunsch erinnern, mal ein „eigenes“ Board zu haben, das preiswert (maximal 50 Euro), stromsparend und mit einer vernünftigen - also nicht unnötig hohen - Rechenpower für eine breite Palette von nicht-grafischen Anwendungen ausgestattet sein sollte. Daneben sollte das Ganze auch noch relativ leicht selbst herstellbar sein, vielleicht sogar mit nur einer zweilagigen Platine. Im Sommer vor zwei Jahren war der erste Prototyp dann da, im Herbst lief endlich das erste Linux drauf (ELDK).

Kurz darauf, im Februar 2012, haben wir das Board auf der Embedded World in Nürnberg eher nebenbei auf dem Stand des OSADL (www.osadl.org) vorgestellt. Zufällig wurde zur gleichen Zeit auf dem Nachbarstand eines grossen Distributors ein anderes Board mit vollem Marketingeinsatz vorgestellt - Ihr wisst schon, das mit der Himbeere im Logo - das viel mehr kann und dabei sogar noch deutlich preisgünstiger als unseres war. Seit diesem Zeitpunkt hat Embedded Linux auf preiswerten „Community-Boards“ einen extremen Schub erfahren. Als Gnublin-Bastler kann einen schon mal das Gefühl beschleichen, von der Entwicklung komplett überrollt zu werden, zumal bei den grossen Mainstream-Boards auch eine riesige Gemeinschaft an Entwicklern und Anwendern dahinter steht. Man zähle nur mal die 2013 geschriebenen Himbeer-Bücher, bei gut über 20 habe ich aufgehört zu zählen.

Das Schöne am Gnublin-Projekt ist dabei, dass es eine unaufgeregte Nische an Hardware-Eigenschaften belegt, in der ich gerne noch weitere ähnliche Projekte ansiedeln würde. Wichtig sind uns die geringe Leistungsaufnahme von höchstens einem halben Watt, auch wichtig ist der minimalistische Hardware-Entwurf, der sogar ohne festes Netzwerk-Interface auskommt und nach dem Prinzip funktioniert: was Du sonst noch brauchst, steckst Du über USB an. Überhaupt nicht wichtig ist für uns die relativ niedrige Takfrequenz (180 MHz), mit denen der Prozessor läuft und auch die 32 MByte Speicher reichen auch locker für speicherintensive Interpretersprachen wie Python (ich geb's ja zu - Node.js braucht mehr, um vernünftig zu laufen).

So, nach diesem kurzen Ausflug kann ich nur noch die Lektüre der Artikel empfehlen. Vielleicht gibt es auch noch andere Projekte da draussen, die irgendwas mit Gnublin gemacht haben und von denen man bisher noch nichts gehört hat. Ich freue mich immer über Rückmeldungen an <Hubert.Hoegl@hs-augsburg.de>.

Viel Freude beim Lesen,

Hubert Högl

Vielen Dank an Hubert Högl für diese netten einleitenden Worte zu unserem GNUBLIN-Projekt!

Benedikt Sauter

und das embedded projects Team!

[1] http://shop.embedded-projects.net

→shop.embedded-projects.net

HARDWARE FOR YOUR PROJECTS – ONLINESHOP

Design your GNUBLIN

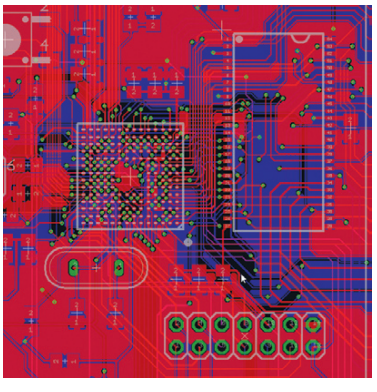
Sie suchen ein Board, das fast so wie GNUBLIN ist und brauchen davon nur eine kleine Menge pro Jahr? Wir passen Ihnen auf kurzem Weg die Schaltung an und ergänzen diese nach Ihrem Wunsch. Dank unserer internen Bestückung können wir Ihnen bereits ab kleinen Mengen ähnliche Stückpreise wie bei den GNUBLIN Boards in diesem Shop liefern.

→ **www.gnublin.org/designer**

Die einfachste Möglichkeit an ein eigenes embedded GNU/Linux Board zu kommen!

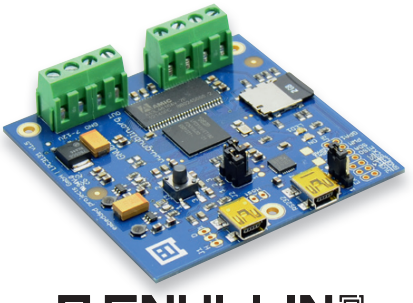
Einfach und genial ist das Produkt „Design your GNUBLIN“. Für alle, die kleine Serien von Platinen mit embedded GNU/Linux benötigen. Basierend auf der Schaltung der Boards der GNUBLIN Familie können wir einfach und schnell kundenspezifische Lösungen erstellen. Innerhalb kürzester Zeit können wir Ihnen die Boards von 1 bis ca. 1000 Stück liefern.

NEU: online GNUBLIN-Designer mit Kalkulator



Wir zeichnen den Schaltplan und das Layout ...

... bestücken mit einem Automaten in Augsburg ...



... und nehmen die Schaltung für Sie in Betrieb.

Holzbachstraße 4, D-86152 Augsburg
Tel +49 (0) 821 279599-0
Fax +49 (0) 821 279599-20
shop@embedded-projects.net

embedded projects GmbH

HARDWARE FOR PROJECTS

EURO CIRCUITS

Die europäische Referenz für PCB Prototypen und Kleinserien

Kostensenkung durch Online-Pooling

- Keine Einrichtkosten
- Keine Mindestbestellwerte - ab der 1. Platine
- Sofort-Bestellung Online - ohne Vorkasse

Zeitgewinn durch neue Online-Daten-Dienste

PCB Visualizer Sofort Design-Rule-Check

PCB Checker sofortige Anzeige von DRC Fehlern im Layout

PCB Configurator kalkulieren und prüfen von Layout-Parametern, Anzeige von Preis-Optionen

PCB proto

spezieller Prototypen-Service für Entwickler, preiswert und schnell

- 1, 2 oder 5 LP in 2, 3, 5 oder 7 Arbeitstagen
- 2 und 4 Lagen
- DRC-geprüft, prof. Ausführung, inkl. 2 x Lötstopplack, 1 x Bestückungsdruck, 150µm Technologie

STANDARD pool

die größte Auswahl an Eurocircuits Pooling Optionen

- FR-4, für bleifreies Löten geeignet, bis 16 Lagen
- Layout-Technologie bis 90µm
- ab 2 AT

RF pool

alle Pooling-Vorzüge mit Hochfrequenz-Material

- Material: Isola I-TERA und Rogers 4000 Serie
- 2-4 Lagen, bis 100µm Technologie
- ab 3 AT

IMS pool

Aluminiumkern-Leiterplatten für hohe Wärmeableitung (z.B. LED-Anwendung)

- Leiterplatten mit einlagig isoliertem Metallsubstrat
- weisser/schwarzer Lötstopplack/Bestückungsdruck oder umgekehrt
- ab 3 AT

www.eurocircuits.de

PCB Visualizer PCB Checker PCB Configurator

HOLEN SIE SICH DIE NEUESTE VERSION!

Neue Funktionen der Version 6.4:

- Simulation Ihres EAGLE Schaltplans in LTspice IV
- Anzeige und Suchfunktion für Attribute im ADD- und RE-PLACE-Dialog
- Import von Designdaten aus P-CAD, Altium und Protel über das Zwischenformat ACCEL ASCII
- Verbesserte Benutzerführung und Voreinstellungen (Tool-tipps, Shortcuts)

www.cadsoft.de

25 Jahre CadSoft
1988 - 2013

irCube - Infrarot per GPIO

Tobias Theile <Tobias.Theile@hs-augsburg.de>

Alexander Koch <Alexander.Koch@hs-augsburg.de>

Die Motivation besteht darin, das Gnublin Mikrocontrollerboard mit embedded Linux so zu programmieren, dass eine Infrarotsenderplatine damit angesteuert werden kann, um Infrarotcodes einer Fernbedienung zu synthetisieren. Zusätzlich sollen über ein Webinterface eine grafische Fernbedienungen simuliert werden, um z.B. den heimischen Fernseher oder anderweitige Peripherie bequem steuern zu können. Das Gnublin-Board und die Senderplatine sollen in einem Gehäuse untergebracht werden, um es für den täglichen Einsatz im Wohnzimmer attraktiv zu machen.

Allgemeine Projektbeschreibung irCube

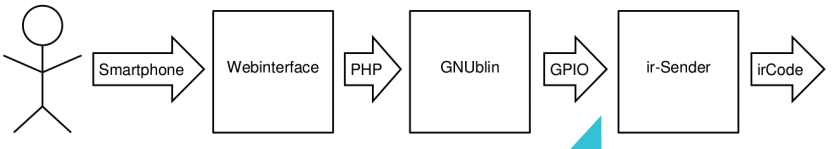
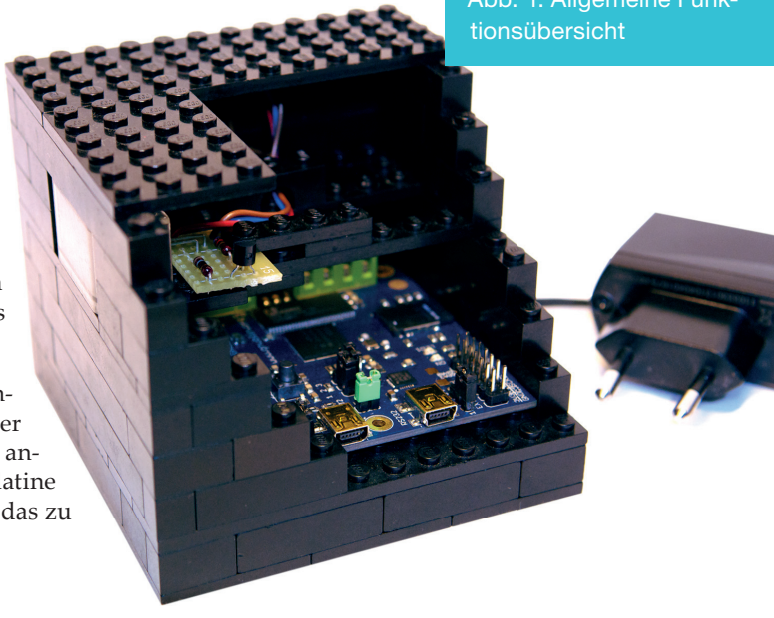


Abb. 1: Allgemeine Funktionsübersicht

Das Projekt trägt den Namen irCube, da sich Gnublin-Board und Infrarotsenderplatine in einem kubischen Gehäuse aus „LEGO“ befinden.

- Mikrocontroller-Board: Gnublin Standard V1.5
- Betriebssystem: Gnublindistribution mit Kernel 2.6.33
- Infrarotsenderplatine: Lochrasterplatine mit Transistorschaltung zum Treiben der Infrarot-LED. (Abb. 4)



Über ein Webinterface sollen eine oder mehrere Fernbedienungen grafisch simuliert werden. Die Webdarstellung soll auf die vorwiegende Benutzung eines Smartphone optimiert sein, kann jedoch mit jedem Browser aufgerufen werden, vorausgesetzt das Gerät verfügt über eine TCP/IP-Verbindung zum irCube.

Via „Tastendruck“ soll ein Befehl vom Webserver aus abgesendet werden. Auf dem Gnublin soll anhand der Aufrufparameter der passende Code aus einer Konfigurationsdatei ausgelesen, anschließend synthetisiert und via GPIO an die Infrarotsenderplatine übergeben werden, welche die Infrarot-LED treiben und somit das zu bedienende Gerät steuern soll (Abb. 1).

NEC-Kodierung und NEC-Protokoll

Warum das NEC-Protokoll? Ein großer Teil aller heimischen Elektronikgeräte mit einer Infrarotfernbedienung verwenden das NEC-Protokoll (Abb. 3). So auch in unserem eigenen Fall. TV, SAT Receiver, AV-Receiver, Subwoofer etc.

Der NEC-Code wird durch die Pulsdauermodulation (PDM) beschrieben. Wie in Abbildung 2 zu sehen ist, bestehen die Merkmale der PDM aus einer konstanten Pulslänge und zwei unterschiedlichen Pausenlängen.

Die logische „1“ besteht aus einem 560µs anstehenden High-Pegel und einem darauf folgenden 1690µs langen Low-Pegel. Die logische „0“ besteht aus einem 560µs anstehenden High-Pegel und einem darauf folgenden 560µs langen Low-Pegel. Jeder High-Pegel muss mit einer Trägerfrequenz von 38kHz ± 2kHz getaktet sein, was auch die Darstellung der High-Pegel in Abb. 2 erahnen lässt. Anhand der hohen, gleichmäßig getakteten Trägerfrequenz können äußere Störeinflüsse, durch unterschiedliche Lichtquellen, beseitigt werden.

Wie in Abb.3 zu erkennen ist, besteht das NEC-Protokoll aus 34-Bit, eine genauere Beschreibung des Protokolls liefern Tabelle 1 und 2.

Frame	34 Bits (1 Start-Bit + 32 Daten-Bits + 1 Stop-Bit)
Start-Bit	9000µs Puls, 4500µs Pause
Daten-Bits	8 Adress-Bits + 8 invertierte Adress-Bits + 8 Kommando-Bits + 8 invertierte Kommando-Bits
Stop-Bit	560µs Puls
Logische „0“	560µs Puls, 560µs Pause
Logische „1“	560µs Puls, 1690µs Pause

Tabelle 1: Das NEC-Protokoll

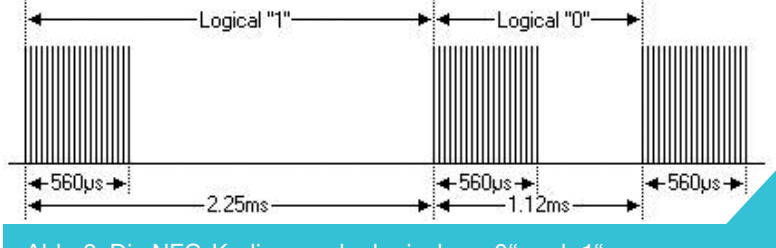


Abb. 2: Die NEC-Kodierung der logischen „0“ und „1“

Die Realisierung einer Infrarotsenderplatine sollte keine große Herausforderung darstellen. Wer löten kann und weiß wo er sich die benötigten Bauteile zu besorgen hat, kann innerhalb kürzester Zeit die Treiberstufe für eine Infrarot-LED realisieren.

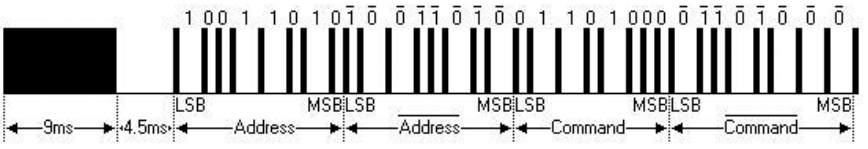
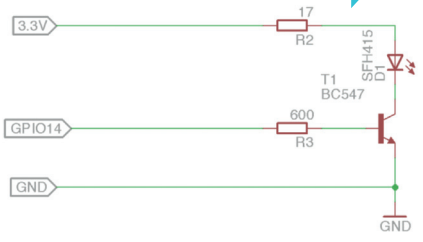


Abb. 3: Das NEC-Protokoll

Abb. 4: Infrarot Treiberstufe



Bits	Beschreibung
Start-Bit	Diese Abfolge markiert den Beginn des Infrarotcodes.
8 Adress-Bits	Beschreiben eindeutig das zu steuernde Gerät.
8 Adress-Bits invertiert	Dienen zur Fehlerüberprüfung der 8 Adress-Bits.
8 Kommando-Bits	Beschreiben den Knopfdruck der Fernbedienung.
8 Kommando-Bits invertiert	Dienen zur Fehlerüberprüfung der 8 Kommando-Bits.
Stop-Bit	Ein einziger High-Pegel mit 560µs markiert das Ende.

Tabelle 2: Nähere Beschreibung der Start-, Stop-, Daten-Bits

Vorbereitung und Grundlagentests

Um dem irCube „WLAN-Fähigkeit“ zu verleihen wurde der USB-WLAN-Stick *ASUS WL-167g v3* verwendet. Auf eine detaillierte Beschreibung der Inbetriebnahme wird an dieser Stelle verzichtet, da dies ausreichend im Gnublin-Wiki [1] beschrieben ist.

Anhand von Grundlagentests, geschrieben in C, soll herausgefunden werden ob es prinzipiell möglich ist eine 38kHz Trägerfrequenz, stabil und zeitkonstant über mind. 560µs zu generieren.

Im nächsten Schritt muss sichergestellt werden, dass dies auch für 9ms der Fall ist, da das Start-Bit 9ms High-Pegel benötigt, welcher mit der Trägerfrequenz getaktet sein muss.

Des Weiteren müssen Tests an den GPIO-Ports durchgeführt werden, um abschätzen zu können, ob eine Taktung der GPIO-Ports mit einem Intervall von 13µs On und 13µs Off stabil und zeitkonstant möglich ist.

Sämtliche Versuche wurden mit einem digitalen Oszilloskop vom Typ „Atten Instruments – ADS- Series 1000“ aufgezeichnet und überprüft.

Unter Linux kann über sysfs direkt auf den GPIO-Port über die Konsole zugegriffen werden und via simplen Dateiänderungen der GPIO-Port gesetzt werden. Siehe hierzu den Artikel im Gnublin-Wiki „GPIO per Konsole“[2].

```
void IrfreqGen()
{
   double erg = 0;
   double sek = 0;
   double frq = 0.0;
   int long msek = 0;

   MakeGpio( Gpio11 );
   DirectionGpio( Gpio11, GpioOut );
   SetValueGpio( Gpio11, GpioClear );
   printf( „Bitte Frequenz in Hz eingeben:\n“ );
   scanf( „%lf“, &frq );
   erg = ( 1 / frq );
   sek = ( erg * 1000000 );
   msek = ((( int long )sec) / 2);
   for ( ; ; ) {
      SetValueGpio( Gpio11, GpioSet );
      usleep(msek);
      SetValueGpio( Gpio11, GpioClear );
      usleep(msek);
   }
}
```

Listing 1: Funktion zum Testen des GPIO aus dem Userspace

Da dies für den irCube von Anfang an nicht in Frage kam, wurde das gleiche Verfahren in C angewandt.

Die ersten Versuche wurden mit einer „gpio.h“ API gemacht, welche von einem User im Gnublin-Forum stammt. Diese beinhaltet lediglich das Öffnen eines Filestreams auf den gewünschten GPIO-Port und ermöglicht das Setzen des Ausgangswertes und der Richtung (IN / OUT) über sysfs.

Um abschätzen zu können, ob es mit dieser Methode aus dem Userspace möglich ist einen GPIO-Port in der benötigten Frequenz anzusteuern, wurde eine kleine Simulationsroutine geschrieben. Wie in Listing 1 zu sehen ist, wird die „gpio.h“ API verwendet um den GPIO-Port 11 anzusteuern.

Der GPIO-Port wird auf „High“ gesetzt und danach der Programmablauf für die umgerechnete Frequenz in Mikrosekunden durch die Funktion *usleep()* schlafen gelegt. Danach wird der GPIO- Port auf „Low“ gesetzt und dieselbe Zeit geschlafen. Anhand des Testcodes kann mit einer eingegebenen Frequenz der GPIO-Port aus dem Userspace angesteuert werden. Allerdings sind hierbei sehr schnell die Grenzen erreicht.

Schon bei einer Frequenz von 1kHz (Erwartungshaltung: Periode mit 1ms) kann das System nicht genauer als ca. 7ms auflösen, siehe Abb. 5. Erst bei Frequenzen kleiner 250Hz (4ms) konnten die eingestellten Frequenzen zeitgenau gemessen werden.

Dies liegt zum einen an den sogenannten Jiffis, das Unix-Standardzeitintervall, welches im Kernel verankert ist und bei der aktuellen Gnublindistribution auf 250Hz (4ms) eingestellt ist. Zum anderen sind die zusätzlichen ca. 2ms durch das Öffnen des Filestreams und Setzen der Wert über das Dateisystem gegeben, wobei auch der Scheduler seinen Teil dazu beiträgt, da er jedem Prozess nur eine gewisse Zeitscheibe zur Verfügung stellt, und nach Ablauf dieser einen Kontextwechsel einleitet.

Generell ist möglich den GPIO-Port direkt über MemoryMapping anzusteuern, was das Öffnen eines Filestreams und Beschreiben von Dateien umgeht und somit Zeit sparen müsste. Dies wurde auch getestet und tatsächlich werden die ca. 2ms Latenz durch das Öffnen des Filestreams umgangen, aber dennoch kann nicht genauer als 1 Jiffy (4ms) aufgelöst werden.

Die Problematik gestaltet sich dahingehend, dass es sich als weitaus schwieriger erweist ein Zeitkonstantes Signal mit hoher Auflösung aus dem Userspace zu generieren, als wir anfänglich gedacht hatten. Um eine höhere Auflösung zu erreichen gibt es ein paar Möglichkeiten die im Folgenden getestet wurden.

High-Resolution-Timer oder kurz HR-Timer, sind seit einigen Jahren fester Bestandteil des Mainline-Kernels. Um sie verwenden zu können muss beim Kompilieren gegen die Realtime-Library gelinkt werden. Dazu muss beim gcc-Aufruf der Parameter *-lrt* mit angegeben werden. Somit können *usleep()* und *nanosleep()*genauer als 1 Jiffy auflösen. Allerdings sind auch den HR-Timern Grenzen gesetzt.

Um dies zu verifizieren wurde im nächsten Schritt ein kleines C Programm geschrieben, welches mit der Funktion *clock_getres()* die Auflösung von *usleep()* und *nanosleep()* für 500µs misst, und ausgibt. Um zu überprüfen, wie stark die eingestellte Zeit jittert, wurde mit einer simplen while-Schleife die Nanosleepmessung ausgegeben. Wie in Abbildung 6 zu erkennen ist, schaffen die HR-Timer es deutlich unter einem Jiffy aufzulösen, jedoch jittern die Werte extrem.

Da Userspaceprogramme eine wesentlich niedrigere Priorität als Kerneltasks besitzen, wurde ein Scheduler eingebunden, welcher die Priorität auf RT („Realtime“) setzt. Die Hoffnung besteht darin, die Zeit noch genauer auflösen zu können und den Jitter zu minimieren. Der Code aus Listing 2 wurde durch den Code aus Listing 3 erweitert, um dem Userspaceprozess RT-Priorität über einen FIFO-Scheduler zu geben.

Wie Abb. 7 zeigt, verbessert der RT-Scheduler die Auflösung ein wenig. Auch das Jittern wirkt nicht mehr so extrem. Allerdings ist es auch unter diesen Bedingungen nicht möglich 38kHz zu simulieren. Da bei eingestellten 500µs die Werte von ca. 750µs – 1900µs jittern. 500µs konnten unter keinen Umständen aus dem Userspace erreicht werden.

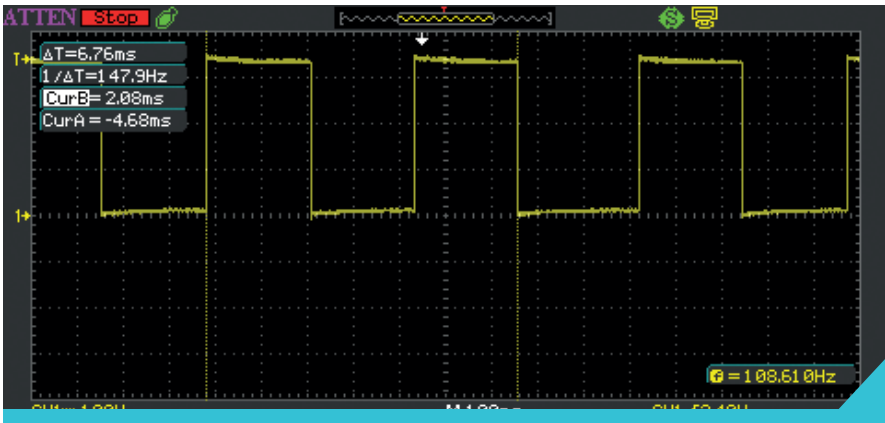


Abb. 5: Messung bei theoretischer Ausgabe von 1 kHz

```
struct timespec ts_res, ts_start, ts_end;
clock_getres( CLOCK_MONOTONIC, &ts_res );
struct timespec ts = { .tv_sec = 0, .tv_nsec = NANO };
clock_gettime( CLOCK_MONOTONIC, &ts_start );
nanosleep( &ts, ( struct timespec * ) NULL );
clock_gettime( CLOCK_MONOTONIC, &ts_end );
printf(„%lu\n“, (ts_end.tv_sec - ts_start.tv_sec) *
       1000000000 + ts_end.tv_nsec - ts_start.tv_nsec);
```

Listing 2: Funktion zum Testen des GPIO mit HR-Timern

```
928667ns
932750ns
997417ns
929750ns
847167ns
1930583ns
1972167ns
935000ns
1862083ns
815269ns
^C
root@gnublin:~/gnublin-projects/test-code/time#
```

Abb. 6: Ausführen der Funktion mit HR-Timern

```
#include <sched.h>
struct sched_param param;
param.sched_priority = 99;
if ( sched_setscheduler(0, SCHED_FIFO, & param) != 0 ) {
   perror( „sched_setscheduler“ );
   exit(EXIT_FAILURE);
}
```

Listing 3: FIFO-Scheduler, Priorität 99 (RT)

```
Messung mit usleep:
836166ns

Messung mit nanosleep:
739250ns
root@gnublin:~/gnublin-projects/test-code/time/src#
```

Abb. 7: Zeitmessung mit RT-Scheduler

Fazit: Es ist auch mit HR-Timern und RT-Scheduler nicht möglich eine zeitkonstante Trägerfrequenz von 38kHz aus dem Userspace zu simulieren.

Zeitmodulation im Kernelspace

Wie aufgezeigt ist es mit den gegebenen Voraussetzungen nicht möglich eine Trägerfrequenz von 38kHz aus dem Userspace zu simulieren. Deshalb war schnell die Entscheidung getroffen einen eigenständigen Treiber im Kernelspace zu schreiben. Da bis zu diesem Zeitpunkt niemand von uns je ein Kernelmodul geschrieben hat, war eine neue Herausforderung geboren.

Das Buch „Linux Device Drivers“ vom O'Reilly Verlag [3] hat uns hierbei sehr geholfen. Das Buch ist frei im Web verfügbar, da es unter der Creative Commons Lizenz veröffentlicht wurde. Nach Einarbeitung in das komplexe Thema der Linux-Treiber-Programmierung konnte das erste Kernelmodul geschrieben werden.

Das Kernelmodul soll beim Einhängen ins System eine Trägerfrequenz mit einer Pulsweite von 26µs erstellen und am GPIO-Port 11 ausgeben. Dieser Test soll zeigen ob es überhaupt möglich ist eine Trägerfrequenz von 38kHz im Gnublin zu simulieren. Beim Laden des Moduls mit dem Befehl *insmod* wird in einer while-Schleife 1000mal die Funktion *pulse_send()* aufgerufen. Der Übergabeparameter (unsigned int length) für diese Funktionen ist 560µs, um den High-Pegel des NEC- Protokolls mit 560µs und einer Trägerfrequenz von 38kHz zu simulieren. Unter Listing 4 ist der Code für die Funktion *pulse_send()* zu sehen.

```
void pulse_send(unsigned int lenght) {
   int flag;
   unsigned long t, act;
   t = PULSE / 2;
   flag = 1;
   act = 0;
   while ( act <= lenght ) {
      gpio_set_value( GPIO_GPIO11, flag );
      udelay(t);
      act += t;
      flag = !flag;
   }
}
```

Listing 4: Kernelmodul zur Simulation der Frequenz

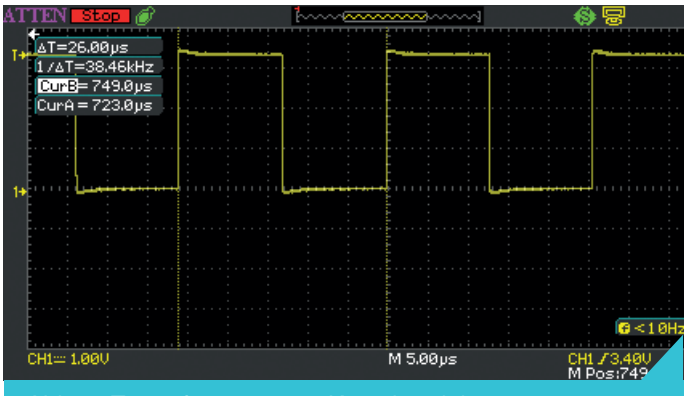


Abb. 8: Trägerfrequenz aus Kernelmodul

Diese Funktion durchläuft eine while-Schleife so lange bis sie den Übergabewert von 560 erreicht hat. In dieser Zeit wird der GPIO-Port alternierend ein und ausgeschaltet.

Abbildung 8 zeigt das erfolgreiche Ergebnis unseres ersten Kernelmoduls. Im Kernelspace ist möglich die Trägerfrequenz von 38kHz zeitkonstant zu erzeugen. Somit muss nun ein Kernelmodul geschrieben werden, welches als Treiber funktioniert und anhand einer Userspaceapplikation mit der passenden Logik bedient wird um das NEC-Protokoll synthetisieren zu können.

irCube Software, Kernel- und Userspace

Nachdem im vorherigen Kapitel bewiesen wurde, dass eine Trägerfrequenz mit 38kHz im Kernelspace synthetisiert werden kann, müssen nun Applikationen und das Kernelmodul geschrieben werden.

Die Zusammenhänge der einzelnen Programmteile, Module und Funktionen werden in Abb. 9 schematisch beschrieben.

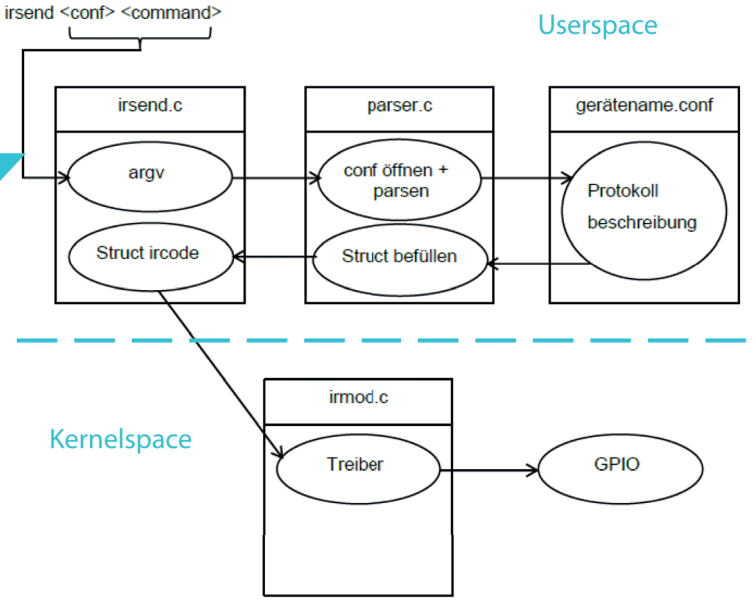


Abb. 9: Schematische Beschreibung der irCube Software

Von der Kommandozeile soll es möglich sein über den Aufruf „irsend <Gerätename> <Kommandoname>“ den vollständigen NEC-Code für einen Tastendruck der Fernbedienung zu synthetisieren. Anhand von unterschiedlichen Konfigurationsdateien sollen unterschiedliche Fernbedienungen beschrieben werden können. Diese sind unter /etc/ircube zu finden und tragen den Namen „Gerätename.conf“,.

In diesen Konfigurationsdateien wird der zu synthetisierende Code der spezifischen Fernbedienung beschrieben. Adress-Bits und Kommando-Bits müssenv als 16 Bit Hexadezimalcode vorhanden sein (8Bit + 8 invertierte Bits).

Die Beschreibung des Start- und Stop-Bits wird über die tatsächliche Puls- und Pausenzeit in Mikrosekunden beschrieben.

Listing 6 zeigt als Beispiel den Inhalt der „sat.conf“ welche meinen Humax-Satreceiver beschreibt.

Die „irsend.c“ [3] ist die Userspace – Kernelspace Schnittstelle. Sie beinhaltet die gesamte Logik um einen Struct mit den benötigten Werten aus der config-Datei aufzubauen und an das Kernelmodul weiterzuleiten.

Ein Teil der „irsend.c“ ist die „parser.c“ und „parser.h“. Der zu füllende Struct für den ircode wird in der „parser.h“ erstellt und somit in „irsend.c“ und „parser.c“ eingebunden.

Der ircstruct ist so konstruiert, dass er die zur Beschreibung des NEC-Protokolls wichtigen Parameter in unsigned Integern speichert, um diese dann als „Paket“ an das Kernelmodul weiter zu reichen.

Aus der „irsend.c“ wird der Parser aufgerufen. Anhand des Konsolenbefehls „irsend sat MUTE“ möchte ich kurz die Vorgehensweise der Userspaceapplikation beschreiben.

Die beiden argv-Parameter „sat“ und „MUTE“ werden beim Parseraufruf übermittelt. Anhand des argv Parameters “sat” wird die “sat.conf“ unter /etc/ircube geöffnet. Diese Datei wird Zeilenweise nach den benötigten Parametern durchsucht und passende Zeilen temporär gepuffert. Bedingt durch den argv-Parameter „MUTE“ wird nur die Zeile mit dem Inhalt „MUTE“ ausgelesen und temporär gepuffert.

Die gepufferten Zeilen werden nach den benötigten Werten geparst und in den Struct geschrieben. Der fertige Struct wird nun aus der „irsend.c“ mit Hilfe von ioctl() dem Kernelspace zugänglich gemacht.

Das Kernelmodul kopiert via ioctl() den Inhalt des erstellten Structs und moduliert den fertigen Infrarotcode.

Der Quellcode des Kernelmoduls ist im Git-Repository[3] einsehbar, und wird in diesem Artikel aus Platzgründen nicht gelistet.

```
sb _ pulse = 9000
sb _ space = 4500
adress = 0x0008
   command:
   MUTE = 0x18E7
   PUP = 0x08F7
   PDOWN = 0xF00F
   BACK = 0x827D
   INFO = 0xC23D
   EXIT = 0x6897
   VDOWN = 0x02FD
   VUP = 0xF807
   ONE = 0xC03F
   TWO = 0x20DF
   THREE = 0xA05F
   FOUR = 0x609F
   FIVE = 0xE01F
   SIX = 0x10EF
   SEVEN = 0x906F
   EIGHT = 0x50AF
   NINE = 0xD02F
   ZERO = 0x30CF
   POWER = 0x00FF
stop = 560
```

Listing 6: sat.conf

Allerdings möchte ich noch kurz auf die Probleme eingehen die uns bei der Codemodulation immer wieder beschäftigt haben und es eine geraume Zeit verhindert haben den SAT-Receiver zu „muten“.

Obwohl der Kerneltreiber für die Modulation der Trägerfrequenz funktioniert, war es nicht möglich eine komplette Infrarotcodeabfolge zu synthetisieren um den SAT-Receiver tatsächlich zu muten.

Dies lag daran, dass der Kernel den Befehl udelay() zwar nicht unterbrechen kann, aber der komplette Code aus unterschiedlichen pulse_send() und space_send() aufrufen besteht, und der Scheduler zwischen den Funktionssprüngen die Möglichkeit hat den auszuführenden Code zu unterbrechen.

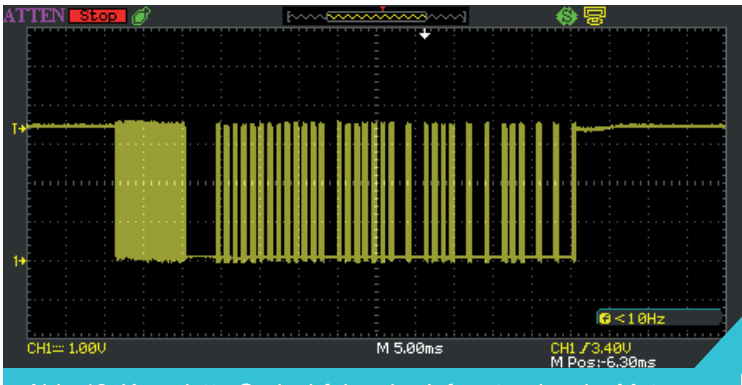


Abb. 10: Komplette Codeabfolge des Infrarotcodes der Mute-Taste für den Humax-Receiver

Durch das Verhalten des Schedulers wurde der Infrarotcode zerhackt und stellenweise Latenzen von mehreren Millisekunden im Code erzeugt, welche den Empfänger daran hinderten den Infrarotcode als Ganzes zu erkennen. Deshalb wurde mit einem Spinlock gearbeitet, um den Kernel samt Scheduler während der Erzeugung des Infrarotcodes zu blockieren, damit die Codeabfolge nicht durch Interrupts zerhackt werden kann.

Des Weiteren wird im Struct das Datenwort (32-Bit) in seine Bestandteile Adress-Bit (16 Bit) und Kommando-Bit (16 Bit) zerlegt. Der Versuch den Hexadezimalcode in 2 for-Schleifen a 16-Bit zu generieren hat nicht funktioniert, da durch die beiden einzelnen for-Schleifen eine Latenz in der Codegenerierung auftritt, welche dazu führt das der Code nicht vom Gerät erkannt wird. Deshalb wird im Kernelmodul aus beiden 16 Bit Werten ein einzelner 32 Bit Wert erstellt, der in einer einzelnen for-Schleife ausgegeben wird.

Auch der Durchlauf der while-Schleife in der pulse_send() hat pro Durchlauf (GPIO ON – OFF) ca. 1µs an zusätzlicher Latenz erzeugt. Wurde das Start-Bit mit 9000µs synthetisiert, so muss die while-Schleife 750-Mal durchlaufen werden. Nach Beendigung war das erzeugte Start-Bit aber nicht 9000µs andauernd, sondern ca. 9750µs. Wie in Listing 5 zu sehen, wird dies nun abgefangen, wobei „lenght“ der Signaldauer entspricht (z.B. 9000µs) und „t“ der halben Periode der Trägerfrequenz (12µs).

```
while ( act <= ( lenght - (lenght / t) ) ) {
   gpio _ set _ value( GPIO _ GPIO11, flag );
   udelay( t );
   act += t;
   flag = !flag;
}
```

Listing 5: Korrektur des systematischen Synthesefehlers

Abb. 11: sat.php

Nun war es endlich möglich einen kompletten Infrarotcode zu synthetisieren, der auch vom Gerät empfangen und interpretiert werden kann. Die Software funktioniert und es können Infrarotcodes mit NEC-Kodierung synthetisiert werden. Abbildung 10 zeigt die komplette Codeabfolge des Infrarotcodes der Mute-Taste für den Humax-Receiver. Mit dem Webserver lighttpd wird im irCube ein eigener Webserver in Betrieb genommen. Via PHP soll der Befehl „irsend <config> <command>“ generiert werden, sobald man auf den passenden grafischen Button drückt.

Alle Grafiken wurden von Tobias Theile in Gimp kreiert. Wird der Webserver über die IP-Adresse des irCube angesprochen, so erscheint die Startseite mit den konfigurierten Geräten. In Abbildung 11 ist meine aktuelle Konfiguration mit der Fernbedienung für den Humax Sat-Reciever dargestellt.

Anhand des PHP-Codes kann jeder passende irsend-Befehl generiert werden. „$_POST[‘one_x‘]“ wird benötigt um den Button in X-Richtung auszuwerten. Sobald der Button Gedrückt wird entsteht hier ein Event welches ausgewertet wird (Listing 7). Anhand des Events wird der irsend-Befehl zusammengebaut und abgesendet.

```
$remote = „sat“;
if(isset($ _ POST[,power _ x‘] )) {
$output = „<pre>“.shell _ exec(“irsend $remote POWER“).“</pre>“; echo
$output;
}
```

Listing 7: HTML/PHP: Beschreibung der SAT-Fernbedienung

Fazit

Es ist erstaunlich wie genau man Zeit im Kernel auflösen kann. Ein 12µs genaues Signal zu generieren erledigt an sich die Funktion udelay(), jedoch ist der Aufwand weit aus größer wenn man ein Signal mit einer Laufzeit von knapp 70ms generieren möchte, welches nicht durch den Scheduler zerstückelt werden darf.

Das Softwarepacket des irCubes ist noch lange nicht vollendet. Es fehlt noch ein Modul welches die Infrartocodes einer Fernbedienung einlesen kann, und diese automatisch in die config-Datei einbindet. Auch sollte die statische HTML-Seite der Fernbedienung durch einen Prozess dynamisch aus der config-Datei erstellt werden. Diese und weitere Features werden wir versuchen im Laufe der Zeit zu erweitern, aus Spaß an der Freude. Denn derartige Projekte erweitern den Horizont und das Verständnis für Betriebssysteme immens.

Literatur und Links

[1] http://wiki.gnublin.org/index.php/WLAN

[2] http://wiki.gnublin.org/index.php/GPIO

[3] http://lwn.net/Kernel/LDD3/

[4] https://bitbucket.org/PeterZosh/ircube

3D-Drucker mit Gnublin

Manuel Liebert <Manuel.Liebert@hs-augsburg.de>

Grundgerüst

Die Grundlage war bereits Vorhanden. Der 3D-Drucker wurde früher direkt über die parallelle Schnittstelle mit dem PC verbunden und steuerte über eine spezielle Software die Schrittmotoren an. Zuerst wurde die alten Steuerplatinen – auf Abb. 1 zu sehen – entfernt.

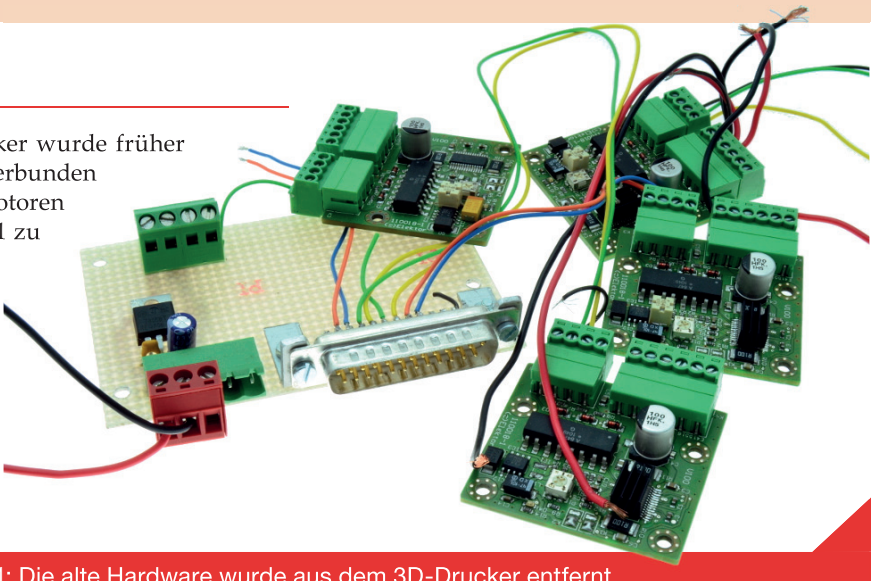


Abb. 1: Die alte Hardware wurde aus dem 3D-Drucker entfernt

Hardware

Als nächstes wurde die neue Hardware montiert und verkabelt (Abb. 2).

- 1x Gnublin-LAN [1]
- 1x Module-Extension (Statusanzeige, Taster und Buzzer) [2]
- 4x Module-Step (Schrittmotor) [3]

Die Hardware der Heizung ist sehr einfach aufgebaut. Im Druckkopf ist ein PT1000 integriert. Über einen Spannungsteiler aus einem Festwiderstand und dem Temperaturabhänigen PT1000 im Druckkopf konnte somit sehr einfach durch den ADC-Eingang am Gnublin der Spannungsabfall über dem PT1000 bestimmt und dadurch die Temperatur berechnet werden.

Ist die Temperatur zu niedrig, wird über das Relais auf dem Gnublin-LAN der Stromkreis der Heizung im Druckkopf geschlossen. Ist die Temperatur wider hoch genug, wird das Relais wieder geöffnet.

Software

Die verwendeten Gnublin Module-Step werden über den I2C Bus angesteuert. Für die komfortable Ansteuerung der Module in der Drucker Software wurde zuerst eine Klasse gnublin_module_step [4] für die Gnublin-API erstellt.

Die Software für den 3D-Drucker baut komplett auf die Gnublin-API auf. Es werden die Objekte der Schrittmotor-Klasse erzeugt.

```
gnublin_module_step Motor_x;
gnublin_module_step Motor_z;
gnublin_module_step Motor_y;
gnublin_module_step Motor_p;
```

Listing 1: Objekte Druckerdentfernt

Auf die dann bei der Initialisierung der Motoren und dem restlichen Programmablauf zugegriffen wird.

```
Motor_x.setVmax(4);
Motor_x.setIrun(15);
Motor_x.setMotorParam();
Motor_x.getFullStatus1(); //before operating
Motor_x.runInit();        //before Operating
```

Listing 2: Init Drucker

Die Temperatur- und Heizungssteuerung wurde in einem extra Programm implementiert. Auch das Auslesen der ADC-Werte wurde duch die API realisiert. Der nachfolgende Code zeigt das einlesen der Spannung und das Berechnen der Temperatur.

```
float get_temperature() {
    int voltage = 0;
    float resistance = 0;
    float temperature = 0;
    if((voltage = adc.getVoltage(1)) == -1) {
      return -1;
    } else {
      resistance=(voltage*330)/(3300-voltage);
      temperature=((resistance/100)-1)*259.74;
    }
    return temperature;
}
```

Listing 3: Berechnung der Temperatur

Der folgend gezeigte Code wird in einer Endlosschleife ausgeführt und überpüft nach jeder Sekunde die Ist-Temperatur und schaltet evtl. die Heizung ein oder aus. Am Display wird der Status der Heizung und die Temperatur ausgegeben. Beim ersten Erreichen der gewünschten Temperatur von 240°C wird ein Piep-Ton ausgegeben, um zu signalisieren, dass jetzt gedruckt werden kann.

```
temperature = get _ temperature();
if((temperature < 900) & (temperature > -100))
{
   sprintf(tempchar,„Temp: %.2f“,temperature);
   display.clear();
   display.controlDisplay(1, 0, 0);
   display.print(tempchar);
   if(temperature < 230) {
     gpio.digitalWrite(18, HIGH);
     heating = 1;
   } else {

     if(temperature > 240) {
        if(initial) {
            initial = 0;
            system(“gnublin-pwm -v 0x400“);
            sleep(1);
            system(“gnublin-pwm -v 0x000“);
        }
        gpio.digitalWrite(18, LOW);
        heating=0;
     }
   }
   if(heating) {
     display.print((char*)“Heizung: ein“, 2);
   } else {
     display.print((char*)“Heizung: aus“, 2);
   }
}
sleep(1);
```

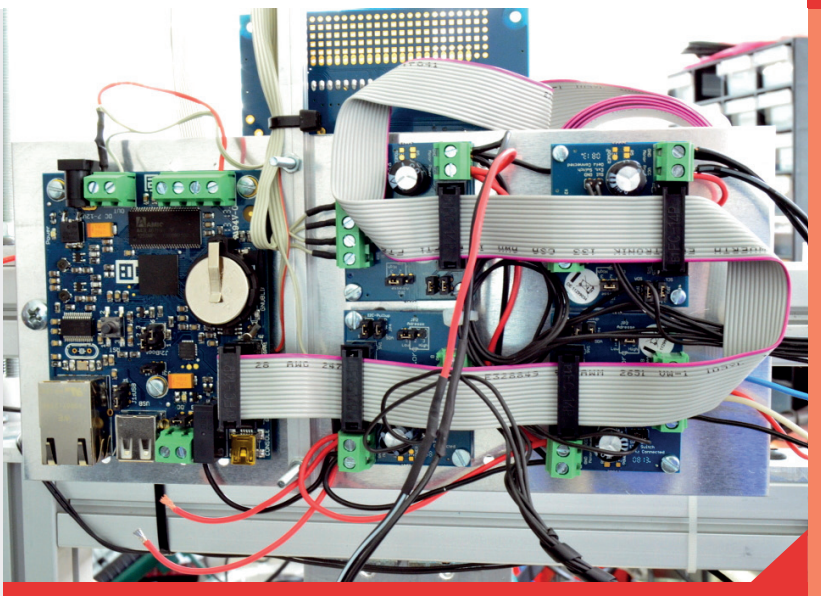


Abb. 2: Die Hardware montiert und verkabelt

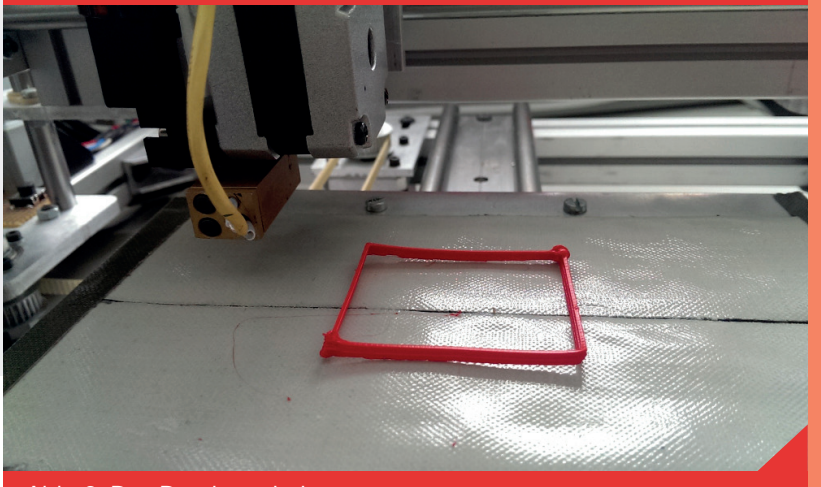


Abb. 3: Das Druckergebnis

Ausblick

Die Software für die Initialisierung inkl. Einer Initialisierungsfahrt jeder Achse des Druckers und anschließendem Anfahren eines bestimmten Punktes über der Druckfläche funktioniert einwandfrei. Leider wurde jedoch der 3D-Drucker aus Zeitmangel nicht fertiggestellt. Momentan ist nur ein einfaches Programm vorhanden, welches kleine Türme drucken kann (siehe Abb. 3). Ein nächstes Projekt in Zukunft könnte sein, einen G-Code Interpreter für das Gnublin zu implementieren, wodurch dann 3D-Formen aus einer G-Code Datei gedruckt werden können.

Bachelorarbeit

Aktuell wird im Rahmen einer Bachelorarbeit ein fertiges Softwarepaket für den Drucker geschnürt. Per Konsole oder Browser kann man G-Code Dateien direkt über Linux auf Gnublin ausdrucken.

Literatur

[1] http://wiki.gnublin.org/index.php/GNUBLIN-LAN

[2] http://wiki.gnublin.org/index.php/GNUBLIN_Module-Extension

[3] http://wiki.gnublin.org/index.php/GNUBLIN_Module-Step

[4] http://gnublin.org/gnublin-api/doc/html_de/classgnublin__module__step.html

NFC mit Gnublin

Christopher Proske <christopher.proske@hs-augsburg.de>

Ziel war es, dass das System einen NFC-Tag einlesen und mit der verknüpften Information einen GPIO-Port steuern. Die Logdaten und die Verwaltung der ID's sollen über ein Webinterface erfolgen. Grundlage hierfür war das Gnublin Board, ein Ethernet zu USB-Adapter und ein NFC-Stick wie es in Abbildung 1 zu sehen ist. Ebenfalls verwendet wurden NFC-Tags, ein USB zu miniUSB Adapter und ein USB-Hub. Mit den Komponenten soll die LED des Boards über den NFC-Stick getoggelt werden. Zum schreiben der ID's wird eine Weboberfläche mit CGI verwendet. Ebenso für das lesen der ID's und der Logdaten.

Hardware

Das Gnublin Board ist ein sehr kompaktes und leistungsstarkes Entwicklungsboard der Firma embedded projects GmbH. Es ist mit einem ARM9 mit 180 MHz und 32 MB RAM ausgestattet. Ebenfalls ist ein USB zu seriell Adapter mit einem miniUSB Anschluss verbaut. Als Hauptspeicher wird eine MicroSDHC Karte verwendet. Als weitere Schnittstellen sind GPIO, I2C, SPI, UART und eine miniUSB Device oder Host Schnittstelle welche auch OTG verwendet werden kann verbaut.

Abb. 2: Der Ethernet-to-USB Adapter

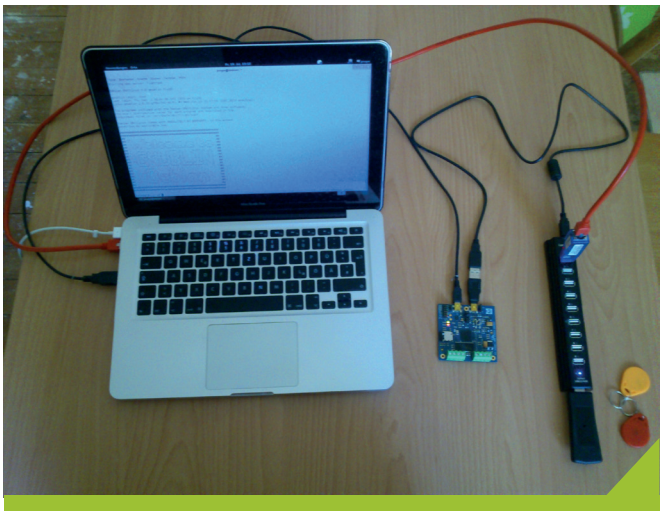


Abb. 1: Aufbau des kompletten Systems

Als Ethernet-Schnittstelle wurde ein einfacher und preiswerter No-Name Adapter verwendet (Abb. 2). Der Adapter hat einen USB 1.1 Eingang und einen RJ-45 Ethernet Ausgang. Der Stick beherrscht den IEEE802.3 Ethernet & Fast Ethernet mit 10/100 MBit/s. Im Stick ist der dm9601 Chipsatz verbaut.

Abb. 3: Der NFC-Stick

Der NFC-Stick (Abb. 3) kann als Reader und Writer für NFC Tags verwendet werden. Er unterstützt den NFC P2P Modus zum Datenaustausch zwischen Geräten. Der Stick wird von der Firma Sensor ID hergestellt und ist mit dem NXP PN533 Chipsatz ausgerüstet und deckt damit die volle Funkionsbandbreite des NFC Standrads ab. Der Stick wird vollständiG von der LibNFC unterstützt.

Durchführung

Zur Durchführung des Projekts muss der USB-Hub mit dem Board verbunden werden, hierfür ist ein Adapter von miniUSB auf UBS nötig. Am Hub werden der Ethernet-Stick und der NFC-Stick eingesteckt. Um den Ethernet-Stick mit dem dm9601 Chipsatz in Betrieb zu nehmen muss das Kernelmodul angepasst werden. Für den Betrieb des NFC-Stick muss die LibNFC installiert werden. Als Betriebssystem wird eine Debian Distribution mit einem 2.6.33 Linux-Kernel verwendet. Als Bootloader wird der Apex Bootloader verwendet.

Bereitstellen der Entwicklungsumgebung

Um Software für das Gnublin Board auf dem Host-Rechner kompilieren zu können muss eine Toolchain installiert werden. Hierfür habe ich die von ELDK verwendet. Als ersten Schritt müssen die nötigen Daten heruntergeladen und endpackt werden. Dazu wechselt man in einen beliebigen Ordner und führt folgende Befehle aus:

```
wget http://gnublin.org/downloads/eldk-eglibc-i686-arm-toolchain-qte-5.2.1.tar.bz2

sudo tar xjf eldk-eglibc-u686-arm-toolchain-qte-5.2.1.tar.bz2 -C /
```

Anschließend wird im Ordner /opt/eldk-5.2.1 ein Skript erstellt.

```
cd /opt/eldk-5.2.1

sudo vim set.sh
```

Statt vim kann auch jeder andere Texteditor verwendet werden. Die Datei set.sh muss folgenden Inhalt haben:

```
P1=/opt/eldk-5.2.1/armv5te/sysroots/i686-eldk-linux/usr/bin/armv5te-linux-gnueabi/
P2=/opt/eldk-5.2.1/armv5te/sysroots/i686-eldk-linux/bin/armv5te-linux-gnueabi/
export ARCH=arm
export CROSS _ COMPILE=arm-linux-gnueabi-
export PATH=$P1:$P2:$PAT
```

Listing 1: Inhalt der der /opt/eldk-5.2.1/set.sh

Das Skript muss abschließend noch gesourced werden:

```
. /opt/eldk-5.2.1/set.sh
```

Installation des dm9601

Da der Hersteller des dm9601 Chips Veränderungen bezüglich der USB-Lizenzen vorgenommen hat, kann es beim Betrieb zu Problemen kommen. Diese können während des Betriebs behoben werden. Um dies zu bewerkstelligen muss zunächst die ID des verwendeten Sticks bekannt sein. Die ID kann mit dem dmesg Kommando heraus gefunden werden, welches nach dem ersten Einstecken des Sticks erfolgen muss. Hierbei ist noch die Option |tail -20 empfehlenswert.

```
dmesg |tail -20

usb2-2: New USB device found,
<strong>idVendor</strong>=XXXX,<strong>
idProduct</strong>=XXXX
```

Die wichtigen Informationen sind in den XXXX bei idVendor und idProduct enthalten. Sobald diese Informationen verfügbar sind kann ein für das Gnublin Board passender Kernel heruntergeladen werden. Hierfür steht ein git Repository bereit.

```
git clone https://github.com/embeddedprojects/
gnublin-lpc3131-2.6.33.git
```

```
cd gnublin-lpc3131-2.6.33/linux-2.6.33-lpc313x
```

In diesem Repository ist bereits eine angepasste Makestruktur vorhanden. Mit dieser können der Kernel und die passenden Module schnell und einfach erzeugt werden.

```
make gnublin _ defconfig
make menuconfig
```

In der menuconfig muss das dm9601 Modul eingebunden werden, da es in der Standardkonfiguration nicht enthalten ist. Wenn dies passiert ist muss die dm9601.c Datei um die oben ermittelten Daten erweitert werden. Die Datei befindet sich im Ordner ../driver/net/usb und muss mit einem Texteditor um folgende Passage erweitert werden.

```
{
USB _ DEVICE(<strong>0xXXXX</
strong>,<strong>0xXXXX</strong>),
.driver _ info = (unsigned long)&amp;dm9601 _
info,
}
```

Hierbei müssen die XXXX durch die ermittelten Zeichen ersetzt werden. Danach können der Kernel und die Module kompiliert werden.

```
make zImage
```

```
make modules
```

```
make modules _ install
```

Wenn das erfolgreich abgeschlossen ist kann der neue Kernel und die Module auf die SD-Karte übertragen werden. Anschließend muss das System gebootet werden und der Stick kann angesteckt werden. Um diesen nun in Betrieb zu nehmen muss nur das Modul eingebunden werden.

```
modprobe dm9601
```

Nun kann die Ethernetschnittstelle konfiguriert werden. Für die Zuweisung der IP mit DHCP wird der dhclient benötigt.

```
dhclient eth0
```

Eine manuelle IP kann mit ifconifg zugewiesen werden.

```
ifconfig eth0 192.168.0.1 netmask 255.255.255.0 up
```

Installation der LibNFC

Um den NFC-Stick ansprechen zu können muss die LibNFC installiert werden. Hierfür müssen zwei weitere Librarys Installiert werden. LibUSB und LibPSCS. Dies kann mit apt-get erfolgen.

```
apt-get install libusb-dev libpcsclite-dev
```

Ebenfalls wird die pkg-config benötigt. Diese kann auch mit apt-get installiert werden.

```
apt-get install pkg-config
```

Nun kann die LibNFC installiert werden. Diese steht zum Download bereit.

```
wget http://libnfc.googlecode.com/files/libnfc-
1.7.0-rc7.tar.gz
```

```
tar -xvzf libnfc-1.7.0-rc7.tar.gz
```

```
cd libnfc-1.7.0-rc7
```

In diesem Ordner ist alles was für die Installation benötigt wird. Die Installation erfolgt über ./configure und make.

```
./configure
```

```
make
```

```
make install
```

Abschließend muss noch der Pfad zur Lib dem System übergeben werden.

```
sh -c „echo /usr/local/lib > /etc/ld.so.
conf.d/usr-local-lib.conf“
```

Die Installation kann einfach getestet werden, indem man eines der mit installierten Programme verwendet.

```
nfc-list
```

Umsetzung des CGI Servers

Für die Benutzeroberfläche (Abb. 4) wurden ein Textfeld und drei Buttons implementiert. Das Textfeld dient zur Eingabe der ID‘s und die Buttons sind jeweils zum senden der ID, zum betrachten der eingetragenen ID‘s und zum betrachten der Logdatei. Für das Schreiben der ID‘s wurde die Methode pos verwendet und zum lesen jeweils die Methode get.

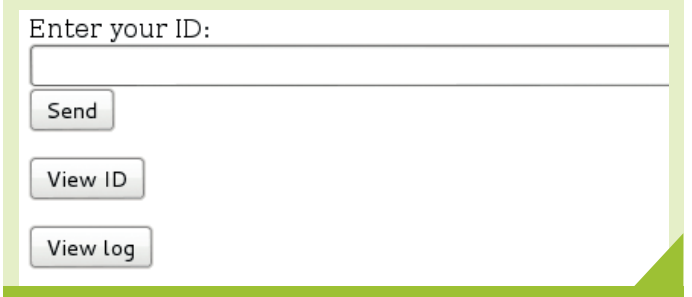


Abb. 4: Benutzeroberfläche des CGI Servers

Für die drei beschriebenen Funktionen mussten drei Anwendungen implementiert werden. Dies wurde mit der Programmiersprache C durchgeführt. Für die put Anwendung musste eine Funktion für die Umrechnung der eingegebenen Zeichen verwendet werden. Dieser ist in der Funktion unencode() enthalten [1]. Der main Teil umfasste das Lesen der eingegebenen Daten und das Öffnen des angegebenen Files und das anschließende Schreiben der gelesenen Daten. Die put Anwendungen sind einfacher strukturiert und umfassen lediglich das Öffnen und Ausgeben der Datei (Abb. 5). Die Anwendungen müssen mit gcc mit der Dateiendung .cgi kompiliert werden und in den Ordner /var/www/cgi-bin kopiert werden. Abschließend muss die put Anwendung noch root-Rechte erhalten.

```
chmod u+s print.cgi
```

```
Log ID: e4 49 0c c7
Log Datum: Thu Jan  1 00:05:21 1970
Log ID: e4 49 0c c7
Log Datum: Thu Jan  1 00:05:24 1970
Log ID: e4 49 0c c7
Log Datum: Thu Jan  1 00:05:27 1970
Log ID: e4 49 0c c7
Log Datum: Thu Jan  1 00:05:30 1970
```

Abb. 5: Anwendung zum Lesen der Logdatei

Programmierung des C Code

Um die gewünschte Funktionalität der Steuerung der LED mit Authentifizierung über NFC zu realisieren müssen primär zwei Funktionen implementiert werden. Zum einen die Steuerung der LED und zum anderen das Auslesen des NFC Tags und das abgleichen mit einer Liste. Ebenfalls wurde eine Log Funktion integriert. Die Initialisierung der NFC Hardware wird in der main() Funktion übernommen. Die Ausgaben mit printf() wurden für den Betrieb im Hintergrund auskommentiert. Für das Logging wurde eine Textdatei angelegt und das aktuelle Datum und die ID des gelesenen Tag's in die Datei geschrieben. Um die gelesene ID zu vergleichen muss zunächst eine Datei mit den eingetragenen ID's geöffnet werden. Diese Datei wird byteweise eingelesen und mit der NFC-ID verglichen. Wenn alle Bytes übereinstimmen wird eine 1 zurückgegeben. Das Toggeln der LED ist mit C vergleichsweise komplex. Hierfür muss ein Pointer auf den Memory-Ordner generiert werden. Dieser Bereich wird anschließend in den Arbeitsspeicher gemapt, wo die einzelnen Register ausgelesen und mit Bitoperationen manipuliert werden können. Zu Ausgabe der ID des gelesenen Tag's wurde zusätzlich noch eine Funktion implementiert. Diese durchläuft das gelesene Array und gibt es zeichenweise aus.

Um das gegebene Programm mit gcc zu kompilieren muss zunächst ermittelt werden wo die nfc.h Datei liegt. Auf dem von mir verwendeten System befand sich sich im Ordner /include/nfc/. Dieser Pfad muss dem Linker übergeben werden. Ebenfalls muss die libnfc verwendet werden.

```
gcc -o nfc foo.c -I /include
-lnfc
```

Mit der aufgeführten Konfiguration muss im Ordner /etc der Ordner nfc angelegt werden in welchem auch die Anwendung nfc liegen sollte. Die CGI-Anwendungen müssen im Ordner/var/www/cgi-bin liegen. Ergänzend können im Ordner /etc/nfc die Dateien log.txt und id.txt angelegt werden. Dieses kann auch während des Betriebs von den Anwendungen übernommen werden. In die Datei id.txt könnten von hand ID's eingetragen werden, dieses sollte aber von der CGI Anwendung durchgeführt werden.

Alle Dateien zum Projekt sind hier [1] zu finden.

Das Startskript

Um das Board mit der richtigen Konfiguration zu starten musste ein Startskript (Listing 2) verwendet werden. Dieses wurde in die Runlevel integriert.

Das Skript umfasst das Starten der NFC-Anwendung, das Laden des dm9601 Moduls und das konfigurieren der IP-Adresse. Die IP-Adresse kann statisch oder per DHCP-Server vergeben werden. Um nun das Skript zu nutzen muss es ausführbar gemacht und verlinkt werden.

```
update-rc.d name _ of _ skript defaults
```

Danach wird es beim booten verwendet.

```
#! /bin/sh

echo „Starting autstartskript"
./etc/nfc/nfc &
echo „NFC started"
modprobe dm9601
echo „dm9601 loaded"

#for use of static IP
ifconfig eth0 192.168.0.2 netmask 255.255.255.0 up
echo „IP set to 192.168.0.2"

#for use of DHCP
#dhclient eth0
#ip addr show eth0
```

Listing 2: Das Startskript

Projektstatus und Ausblick

Im aktuellen Status kann das System einen NFC-Tag einlesen und mit der verknüpften Information einen GPIO-Port steuern. Die Logdaten und die Verwaltung der ID's erfolgt über ein Webinterface. Das Board bootet komplett selbstständig. Für zukünftige Erweiterungen könnte die Schreibfunktion des Sticks verwendet werden. Ebenso kann der Stick einen Tag emulieren was Möglichkeiten im Bereich der IT-Security eröffnet. Eine weitere interessante Anwendung wäre die Datenübertragung über P2P zu anderen NFC Geräten wie einem Handy.

Links

[1] https://elk.informatik.fh-augsburg.de/dav/elinux-13-berichte/Proske/Bericht/

C++ API für Gnublin mit Python Wrapper

Simon Kunzmann <Simon.Kunzmann@hs-augsburg.de>

Ziel war es eine API zu programmieren um Einsteigern wie auch Fortgeschrittenen ein einfaches Interface für die einfache Verwendung der verfügbaren Schnittstellen bereitzustellen. Im Laufe der Entwicklung entsand unter anderem auch ein größeres Makefile um die API, gnublin-tools und die Python Module komfortabel erstellen zu können. Die API ist momentan nicht nur auf dem Gnublin sondern auch auf dem Raspberry Pi lauffähig. Den aktuellen Stand der API kann man auf Github [1] einsehen.

I2C

Eine meiner Aufgaben war es eine API für den I2C Bus zu entwicklen. Mit der I2C API ist es sehr einfach auf die I2C Schnittstelle zu zugreifen wie man im folgenden Listing 1 sehen kann.

Die I2C API stellt folgende Methoden bereit:

- setAddress
- getAddress
- setDevicefile
- send
- receive
- getErrorMessage
- fail

```
#include „gnublin.h“
int main()
{
   gnublin _ i2c i2c;
   i2c.setAddress(0x42);
   unsigned char buffer[8];
   unsigned char RxBuf[8];
   buffer[0]=0x22;
   i2c.send(buffer,5);
   i2c.receive(RxBuf, 3);
   //sende 2 byte aus buffer an Register- Adresse 0x12
   i2c.send(0x12, buffer, 2);
   // Lese 3 bytes aus Adresse 0x23, speichere sie in RxBuf
   i2c.receive(0x23, RxBuf, 3);
}
```

Listing 1: Beispiel i2c Ansteuerung

Funktion

Die I2C API öffnet das i2c Devicefile mit dem Parameter O_RDRW für den Schreib/Lesezugriff. Das Devicefile auf welches zugegriffen wird, kann über die API auch geändert werden. Nach dem erfolgreichen Öffnen der Gerätedatei wird auf den Slave mit der angegebenen Slave-Adresse zugegriffen. Schlägt dies fehl, wird eine Fehlermeldung generiert. Siehe nachfolgendes Listing 2. Nun kann man mit read() bzw. write() auf den Dateideskriptor zugreifen und somit von einem Slave lesen, bzw an ihn schreiben.

```
int gnublin _ i2c::open _ fd() {
   error _ flag = false;
   if (fd) {
     close _ fd();
     fd = 0;
   }
   if (slave _ address == -1)
     return errorMsg(“ERROR slave address is not set\n“);
   if ((fd = open(devicefile.c _ str(), O _ RDWR)) < 0)
     return errorMsg(“ERROR opening: „ + devicefile + „\n“);
   if (ioctl(fd, I2C _ SLAVE, slave _ address) < 0)
     return errorMsg(“ERROR address: „ +
numberToString(slave _ address) + „\ n“); return 0;
}
```

Listing 2: i2c öffnen der Gerätedatei

Beim Errorhandling haben wir uns darauf geeinigt, es möglichst simpel zu halten. Die API soll ja auch für Anfänger verständlich und nachvollziehbar sein. Deswegen wurde auf ein Try-Catch Konstrukt verzichtet. Jede Schnittstellenklasse (gpio, i2c, spi, etc.) hat eine fail und eine getErrorMessage Methode. Erstere gibt immer das ErrorFlag zurück, ist dieses “true“ ist ein Fehler aufgetreten. GetErrorMessage gibt die letzte Fehlermeldung als String zurück, den man sich dann ausgeben lassen kann.

Im Laufe der Entwicklung der API ist ein recht großes Makefile entstanden. Dieses Makefile fügt nach einem einfachen Aufruf von make alle Sourcecodes zusammen in eine “gnublin.h“ und “gnublin.cpp“ Datei. Es werden außerdem noch ein shared object erstellt und die gnublin-tools kompiliert (rekursiver Aufruf von Make). Ein make install bewirkt, dass die gnublin-tools kompiliert werden, falls das noch nicht passiert ist, und kopiert die binaries nach /usr/local/bin/. Die Tools sind dann einfach per Kommandozeile aufrufbar.

Wird make mit dem Parameter python-module aufgerufen, wird mit Hilfe von swig2.0 aus der C++ API ein Python Modul gebaut, was die API dann per Python zugänglich macht.

Mit make release werden die Gnublin-Tools zu einem Debian Packet gebaut, dass später dann ganz einfach per „dpkg -i“ installiert werden kann. Außerdem wird auch das Python Modul erstellt.

```
make clean
```

löscht alle nicht benötigten Dateien und Objekte.

Das Makefile includiert die Datei „API-config.mk“ (Listing 3). Diese Datei ist für den Benutzer gedacht, um hier Einstellungen vornehmen zu können (z.b. Compiler einstellen, für welches Board soll compiliert werden etc.).

```
#Crosscompiler for Gnublin
CXX := arm-linux-gnueabi-g++
#Crosscompiler for Raspberry Pi:
#CXX := arm-linux-gnueabihf-g++
#Compiler for onboard compilation:
#CXX := g++

#Compilerflags:
CXXFLAGS = -Wall

#Architecture for gnublin:
Architecture = armel
#Architecture for raspberryPi and beaglebone black:
#Architecture = armhf

#Define which Board you want:
BOARD := GNUBLIN
#BOARD := RASPBERRY_PI
#BOARD := BEAGLEBONE_BLACK
#DO NOT EDIT BEYOND THIS LINE!
```

Listing 3: Inhalt der Datei API-config.mk

In den Zeilen 1-6 werden die Compiler aufgeführt. Der jeweils benötigte wird einfach einkommentiert, alle anderen auskommentiert. Wer möchte kann die Compilerflags in Zeile 9 noch erweitern. In Zeile 11 bis 14 legen sie die Architektur fest, was für den Bau des Debian Pakets von bedeutung ist. Am Ende der Datei kann man noch einstellen, welches Board man verwendet. Momentan werden von unserer API das Gnublin-Board, das RaspberryPi und das Beaglebone Black unterstützt. Auch hier gilt: Das benötigte einkommentieren und den rest auskomentieren.

Empfangen von Daten über die Funkschnittstelle

Die API ist auch auf dem RaspberryPi lauffähigt, wie oben schon erwähnt. Das ganze kann man auch im Wiki Artikel Tutorial API RaspberryPi [3] nachlesen. Das RaspberryPi muss gestartet sein und eine Verbindung in das Internet haben. Mit git kann man jetzt das Repository klonen.

```
pi@raspberrypi $ git clone https://github.
com/embeddedprojects/gnublin-api.git
```

Wechselt in das Verzeichnis:

```
pi@raspberrypi $ cd gnublin-api
```

Bevor man die API übersetzen kann, muss man angeben, das die API auf einem RaspbberryPi laufen soll. Dazu ändert man in der Datei API-config.mk ab, wie in 1.3.2 beschrieben. Nun kann man die API übersetzen und die Programme und Beispiele installieren:

```
pi@raspberrypi $ make && sudo make install
```

Damit die API auch funktioniert, müssen folgende Treiber geladen werden:

```
pi@raspberrypi $ sudo modprobe spi-bcm2708
```

```
pi@raspberrypi $ sudo modprobe i2c-bcm2708
```

```
pi@raspberrypi $ sudo modprobe i2c-dev
```

Diese befinden sich in der aktuellen RaspberryPi Distribution. Optional kann man die Treiber fest in die Datei /etc/modules eintragen, dann werden die Treiber Module automatisch beim booten geladen.

```
spi-bcm2708
```

```
i2c-bcm2708
```

```
i2c-dev
```

Hat man jetzt bereits eine Erweiterungsplatine angeschlossen, kann man sofort die kleinen Tools testen.

API in Python verfügbar machen

Zunächst habe ich versucht mit SIP die C++ API zu wrappen. Die Dokumentation war allerdings nicht sehr verständlich und es gab Probleme mit den C++ strings. Deshalb habe ich mich dann nach einer Alternative umgesehen und bin auf swig gestoßen. Um die API in Python zu überführen habe ich schlussendlich swig [4] verwendet. Man braucht dafür nur eine kleine „config“Datei. Diese Datei heißt in unserem Falle gnublin.i und sieht wie folgt aus (Listing 4).

Mit Zeile 1 wird festgelegt wie das Python Modul später heißen wird. Zeile 2 includiert eine interface Datei, damit man die C++ Strings verwenden kann. Zeile 6 ist nötig, da in der API Board-spezifischer Code vorhanden ist, der ohne das Define falsch kompiliert wird.

Hat man nun die gnublin.i Datei sind nur noch folgende Arbeitsschritte (Listing 5) nötig.

```
%module gnublin
%include „std_string.i“
%{
#include „gnublin.h“
%}
#define BOARD GNUBLIN
%include „gnublin.h“
```

Listing 4: swig-Datei

```
swig2.0 -c++ -python gnublin.i
arm-linux-gnueabi-g++ -DBOARD=GNUBLIN -fpic -I python2.6/ -c
gnublin _ wrap. cxx
arm-linux-gnueabi-g++ -DBOARD=GNUBLIN -fpic -c gnublin.cpp
arm-linux-gnueabi-g++ --shared -o _ gnublin.so gnublin _
wrap.o gnublin.o
```

Listing 5: Kommandos für swig

Zeile 1 erzeugt die gnublin_warp.cxx, die in Zeile 2 dann kompiliert wird. Wichtig ist auch, dass man in Zeile 2 die richtige Python Version angibt, unter der das Modul später laufen soll. Zum schluss wvird dann noch eine shared library erstellt. Wichtig ist hier der _ vor gnublin.so. Es befinden sich nun 2 Dateien in dem Ordner, einmal eine gnublin.py und die _gnublin.so. Diese beiden Dateien übertragen wir dann auf das Gnublin.

Auf dem Gnublin wechselt man nun in den Ordner, wo die beiden Dateien liegen und startet die Python Konsole mit dem Aufruf:

```
python
```

als nächstes importieren wir das Python-Modul:

```
>>> import gnublin
```

wollen wir nun als Beispiel unsere GPIOs in python ansteuern, müssen wir zunächst ein GPIO Objekt erstellen:

```
>>> gpio = gnublin.gnublin _
gpio()
```

Ist das Objekt erstellt, kann man wie aus der C++ API gewohnt auf die Methoden der Klasse zugreifen:

Beispiel LED an/ausschalten:

```
>>> gpio.pinMode(3,'out')
1
>>> gpio.digitalWrite(3,1)
1
```

Dieses Programm lässt die LED auf dem Gnublin blinken. Dazu legen wir eine Datei namens ledblink.py an und schreiben den Inhalt von Listing 6 in die Datei.

```
import gnublin
import time
gpio = gnublin.gnublin _
gpio()
gpio.pinMode(3,'out')
while True:
gpio.digitalWrite(3,1)
time.sleep(0.5)
gpio.digitalWrite(3,0)
time.sleep(0.5)
```

Listing 6: ledblink.py

Das Programm wird wie folgt ausgeführt:

```
root@gnublin: # python LED-
blink.py
```

Wollen wir das Programm beenden, drücken wir STRG + C

Links und Literatur

[1] https://github.com/embeddedprojects/gnublin-api

[2] http://www.nxp.com/documents/data_sheet/LM75A.pdf

[3] http://wiki.gnublin.org/index.php/Tutorial_API_RaspberryPi

[4] http://www.swig.org/

Interrupt mehrerer GPIOs

Manuel Liebert <Manuel.Liebert@hs-augsburg.de>

Die Gnublin-API bietet unter anderem einen komfortablen Zugriff auf die auf dem Board befindlichen General Purpose Input/Outputs (GPIOs). Dieses Interface bringt aber noch keine Unterstützung von Interrupts mit sich, deshalb wurde der Versuch gestartet, mehrere GPIOs per Interrupt im Userspace zu erfassen.

Implementierung

Um das andauernde Abfragen der GPIOs zu vermeiden wurde die Funktion Poll benutzt. Dadurch kann zum einen der Pegelwechsel des GPIO ohne größere Verzögerungen verarbeitet werden und zum anderen wird wertvolle Rechenzeit auf dem System gespart.

Im Programm Code werden zuerst die GPIOs als Eingang exportiert, dies wurde mit Hilfe der API erledigt. Als nächstes werden in den edge Dateien im GPIO Sysfs der gewünste Trigger Typ gesetzt. Die möglichen Typen sind steigend(rising), fallend(falling) oder beides(both). Nachdem die Filestreams geöffnet und das Event gesetzt wurden muss ein Lesezugriff erfolgen. Danach kann die Poll Funktion aufgerufen werden und die Eventbehandlung ausgeführt werden.

In dem gerade gezeigten Quellcode Ausschnitt (Listing 1) ist ein Mutex lock zu sehen. Dieser ist nötig, weil in einem 2. Thread des Programms die Anzahl der erfassten Pegelwechsel der einzelnen GPIOs zeitgleich verarbeitet und am Terminal ausgegeben werden.Definiert wird die Mutex Variable in der ersten Zeile. Erstellt und aufgerufen werden die beiden Threads im Listing 2.

Der komplette Quellcode ist im Gnublin-Wiki [1] zu finden.

```
pthread_mutex_t region_mutex = PTHREAD_MUTEX_INITIALIZER;
bytes_read = read(fds[1].fd, buf, 1);
bytes_read = read(fds[2].fd, buf, 1);
ret = poll(fds, 3, -1);
if(ret > 0) {  // An event on one of the fds has occurred.
   for(i = 0; i < 3; i++) {
     if(fds[i].revents & POLLPRI) {
            //lock rad
            pthread_mutex_lock(&region_mutex);
            rad[i]++;
            //unlock rad
            pthread_mutex_unlock(&region_mutex);
     }
   }
}
```

Listing 1: Mutex definieren, Poll starten und Eventbehandlung

```
pthread_t interrupt_thread, printer_thread;
//create threads
pthread_create(&interrupt_thread, NULL, interrupt, NULL);
pthread_create(&printer_thread, NULL, printer, NULL); //wait
until the threads are terminated....
pthread_join(interrupt_thread, NULL);
pthread_join(printer_thread, NULL);
```

Listing 2: Threads

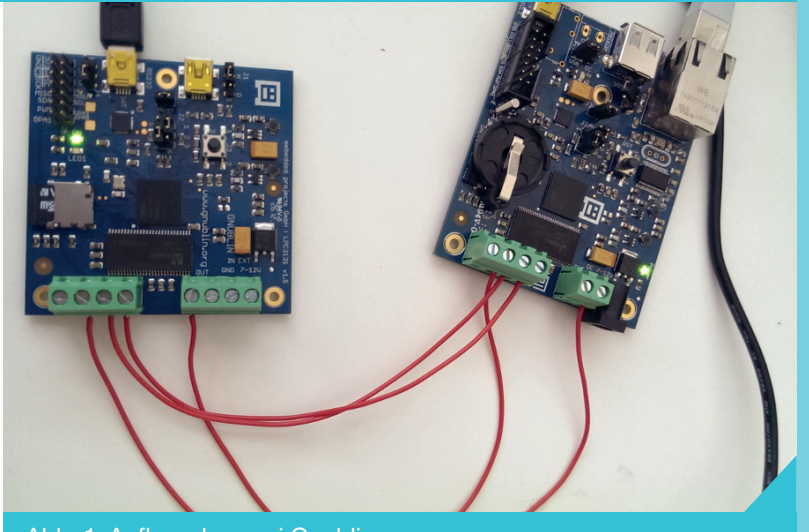


Abb. 1: Aufbau der zwei Gnublins

Test mit 2 Gnublins

Ein zweites Gnublin wurde als einfacher Frequenzgenerator verwendet und über 4 Leitungen (GPIOs und GND) mit dem Mess-Gnublin verbunden (Abb. 1). Bei dem Testsignal erzeugenden Gerät wurde mithilfe der Gnublin-API mittels digitalWrite() an den drei Ausgängen die Rechteckspannung erzeugt, welche in Abbildung 2 zu sehen ist. Die Frequenz lag bei jedem GPIO bei ca. 100Hz. Die Flankenwechsel wurden zuverlässig aufgenommen und ausgegeben. Die Geschwindigkeit von 100Hz ist zwar nicht all zu hoch, sollte aber für einige Anwendungen ausreichen. Um schnellere Pegelwechsel zu erreichen könnte man zum Beispiel mit direktem Zugriff auf den Speicher arbeiten (mem/mmap).

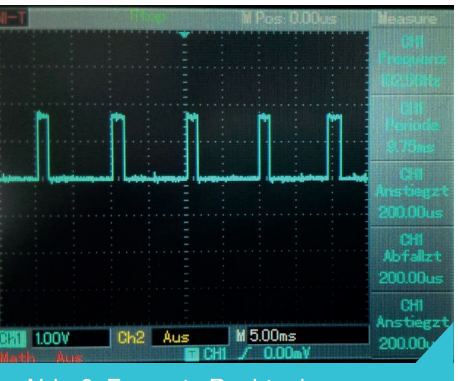


Abb. 2: Erzeugte Rechteckspannung

Literatur

[1] Gnublin-Wiki: http://goo.gl/tdf933

Lesen Sie die neue Elektor ein Jahr lang in der ultimativen GOLD-Mitgliedschaft und profitieren Sie von allen Premium-Vorteilen!

elektor GOLD membership

Die Elektor-GOLD-Jahresmitgliedschaft bietet Ihnen folgende Leistungen/Vorteile:

- Sie erhalten **10 Elektor-Hefte** (8 Einzelhefte + 2 Doppelausgaben Januar/Februar und Juli/August) pünktlich und zuverlässig frei Haus.
- **Extra:** Jedes Heft steht Ihnen außerdem als PDF zum sofortigen Download unter www.elektor-magazine.de (für PC/Notebook) oder via App (für Tablet) bereit.
- **Neu & Exklusiv:** Sie erhalten alle 2 Wochen per E-Mail ein neues Extra-Schaltungsprojekt (frisch aus dem Elektor-Labor).
- **Neu & Exklusiv:** Wir gewähren Ihnen bei jeder Online-Bestellung 10% Rabatt auf alle unsere Webshop-Produkte – dauerhaft!
- **Neu & Exklusiv:** Der Online-Zugang zum neuen Community-Bereich www.elektor-labs.com bietet Ihnen zusätzliche Bauprojekte und Schaltungsideen.
- **Extra:** Die neue Elektor-Jahrgangs-DVD (Wert: 27,50 €) ist bereits im Mitgliedsbeitrag inbegriffen. Diese DVD schicken wir Ihnen sofort nach Erscheinen automatisch zu.
- **Extra:** Top-Wunschprämie (im Wert von 30 €) gibts als Dankeschön GRATIS obendrauf!

UMWELTSCHONEND – GÜNSTIG – GREEN

Möchten Sie Elektor lieber im elektronischen Format beziehen? Dann ist die neue GREEN-Mitgliedschaft ideal für Sie!
Die GREEN-Mitgliedschaft bietet (abgesehen von den 10 Printausgaben) alle Leistungen und Vorteile der GOLD-Mitgliedschaft.

elektor GREEN membership

Jetzt Mitglied werden unter www.elektor.de/mitglied!

„Jägermeister-Uhr“

Ansteuerung einer LED-Matrix als Display mit Webinterface

Peter Fink <Peter.Fink@hs-augsburg.de>

Ziel dieses Projekt war es, einen Atmega48 vom Gnublin aus anzusteuern, welcher die Ausgabe auf einer LED-Matrix regelt. Dabei war das Display von Anfang an als Teilprojekt eines größeren privaten Gesamtprojekts geplant, wobei als Projektarbeit in der Vorlesung „Embedded Linux“ eben nur das Display und die Software behandelt wurden. Das Gesamtprojekt stellt eine „Jägermeister-Uhr“ dar, die aus 12 beleuchteten Jägermeisterflaschen, sowie einem Logo mit einstellbarer RGB-Beleuchtung und eben dem Display besteht. Die Uhr wiegt etwa 25kg und misst 70cm im Durchmesser.

In diesem Kapitel geht es um den Hardwareaufbau, das Übertragungsprotokoll und die Software auf dem Atmega48. Das Display - wie es in Abbildung 1 zu sehen ist - besteht aus 7x29 quadratischen, 0,5mm großen, gelben LEDs, die einem Frame aus 5 Platten á 2mm Edelstahlblech stecken. Auf der Rückseite sind die LEDs auf einer Lochrasterplatine elektrisch gemäß Schaltplan verbunden. Die Bleche waren bei Projektbeginn bereits vorhanden und wurden zuerst mit einem CAD-Programm gezeichnet und anschließend von einem Bekannten in einem metallverarbeitenden Betrieb gelasert bzw. gestanzt. Die LEDs wurden für das Multiplexen so verschaltet, dass jede Zeile eine gemeinsame Anode hat und jede Spalte eine gemeinsame Kathode. Daraus entsteht dann die Matrix, die über entsprechende Zeilen- und Spaltentreiber so gemultiplext werden kann, dass es für das menschliche Auge so aussieht, als würden alle LEDs gleichzeitig leuchten können. Tatsächlich wird beim Multiplexen aber jeweils nur eine Zeile angezeigt und so schnell zwischen den Zeilen gewechselt und umgeschaltet, das ein statisches Bild entsteht. Das hat den Vorteil, dass es die Ansteuerungselektronik um ein vielfaches vereinfacht, 29 LED-Treiber sind weniger als (7 x 29) 203 Treiber, wenn man jede LED einzeln ansteuern würde. Der Nachteil des Multiplexens besteht in der höheren Komplexität der Ansteuerungssoftware, als auch in der zusätzlichen Elektronik für die Zeilenumschaltung, die man aber gern in Kauf nimmt um die Vorzüge genießen zu können.

Hardware-Entwicklung und Software des Atmega48

Als Spaltentreiber wurden zwei CAT4016 von ON Semiconductor gewählt, die jeweils 16 Konstantstromquellen und ein internes Schieberegister bereitstellen, in das die Daten seriell übertragen werden. Die Stromquellen können pro LED einen Strom von maximal 100mA liefern und sind über einen externen Widerstand regelbar; hier wurde ein Poti verbaut, um die Stromstärke und somit die Helligkeit der LEDs stufenlos jederzeit anpassen zu können. Da die Treiberbausteine auch einen seriellen Ausgang besitzen, können beide Chips direkt kaskadiert werden und von der Software wie ein einzelnes Schieberegister angesteuert werden. Für die Zeilentreiber wurden aufgrund der schnellen Schaltzeiten Mosfettransistoren verwendet. Die Zeilentreiber schalten gegen Masse ab, somit mussten P-Mosfets benutzt werden, die im positiven Schaltungszweig schalten.

Der Plan war eine Platine zu layouten und zu ätzen, die sowohl die Zeilen- als auch die Spaltentreiber enthält. Das Display funktionierte mit den verwendeten Transisoren leider nicht und bereitete einige Probleme. Von dieser Platine wird nun nur noch der Teil der Zeilenansteuerung verwendet und es wurde eine neue Platine erstellt, die die sieben neuen IRF9622 P-Mosfets beherrbergt anstatt der zuerst verwendeten SMD Mosfets. Für den Mikrocontroller wurde extra für die Entwicklung eine separate Platine gefertigt, auf der nur der Atmega48, ein Quarz und ein Spannungsregler sind.

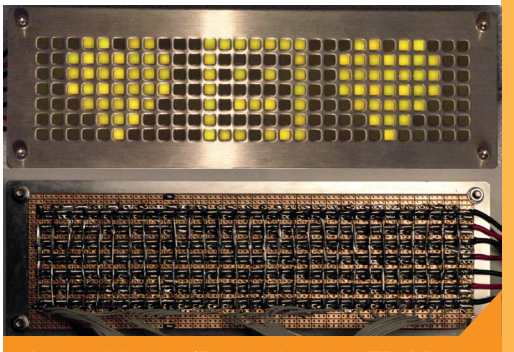


Abb. 1: Vorder-/Rückseite der LED-Matrix

Alle anderen Pins wurden auf Header herausgeführt.

Bilder der gerenderten Platinen sind in den Abbildungen 2 bis 4 zu sehen. Die Schaltpläne, das Layout und die 3D-Ansichten wurden mit dem kostenlosen Programm Diptrace erstellt (leider nur für Windows erhältlich). Als Test für die SPI Kommunikation wurde zuerst jeweils ein Atmega48 als SPI-Master und als SPI-Slave programmiert, die beide über UART mit dem PC verbunden waren. Der Master empfängt Daten über die

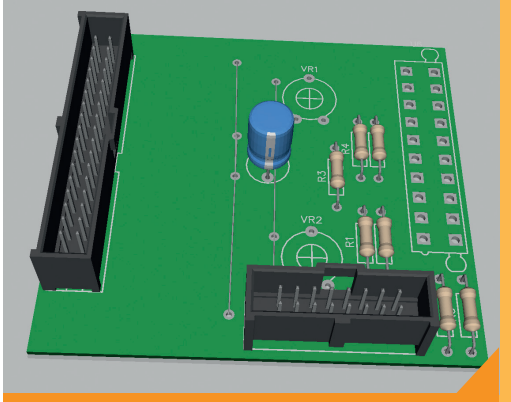


Abb. 2: Gerenderte Platine mit den CAT4016

serielle Schnittstelle vom PC und sendet sie weiter über SPI an den Slave, der wiederum die empfangenen Daten auf der zweiten Konsole ausgibt. Nach erfolgreichem Abschluss dieses Tests konnte eine Stufe weiter gegangen werden, nämlich Kommunikation GNUBLIN(Master) -> Atmega(Slave). Die Programmierung des Slaves wurde nicht verändert und auf dem Gnublin wurde zuerst das Kernelmodul für die SPI-Schnittstelle geladen:

```
modprobe spidev
```

Das Testprogramm für SPI wurde etwas abgeändert und anschließend gestartet:

```
./spidev_test -D /dev/
spidev0.11 -s 500000 -d 10 -b 8
```

Der Test wurde auch erfolgreich abgeschlossen und der Atmega produzierte folgende Ausgabe:

```
Hello!
recv: 0x00
recv: 0x83
```

Das Protokoll der SPI Datenübertragung wurde mit der Anforderung entworfen, möglichst kompakt zu sein ohne die Flexibilität zu verlieren oder spätere Erweiterungen zu erschweren. Eine Datenübertragung besteht somit aus:

- Kommandobyte
- Datenbytes mit Paritybit

Kommandobyte: Für das Kommandobyte ist das MSB reserviert, das in keinem andern Byte der Übertragung gesetzt sein darf. Diese einfache Konvention erleichtert die Interpretation der empfangenen Bytes und es dient als Startbyte. Wird ein gültiges Startbyte empfangen, ohne dass der vorherige Puffer vollständig mit Daten gefüllt wurde, darf dieser verworfen werden. Dies dient dazu fehlerhafte Frames einfach zu ignorieren, könnte aber im Extremfall zu einer ruckelnden Darstellung führen, wenn die Verbindung schlecht ist. Verschiedene Kommandos könnten sein: Übetragung eines vollständigen Bildes; Abschalten des gesamten Displays (mute/blank); Aktualisieren einer einzelnen Zeile/Spalte; etc. Im Moment ist nur die Übertragung eines vollständigen Bildes (C_FULL = OxCO) implementiert. Abhängig vom Kommandobyte ist die Anzahl der folgenden Datenbytes.
Datenbyte: Ein Datenbyte darf nie das MSB gesetzt haben und muss eine gerade Parity besitzen; dazu wird das Bit 6 verwendet. Anschließend sind noch 6 Bit für tatsächliche Daten vorhanden. Da das Display später noch auf „Graustufen“ erweitert werden soll, bzw. das Protokoll dies nicht erschweren soll sind hier pro LED 3 Bit, also 8 Graustufen vorgesehen. Das bedeutet, dass pro Byte der Zustand von zwei LEDs übertragen werden kann. In der aktuellen Implementierung wird jeder Wert größer als 0 als „ein“ angesehen.

Ein vollständiges Bild besteht also aus einem Kommandobyte, 102 Datenbytes für

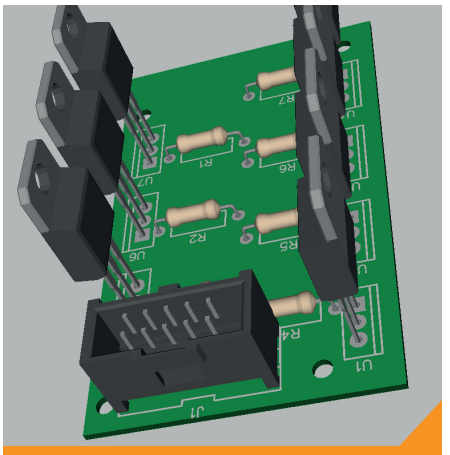


Abb. 3: Separate Platine der Zeilentreiber

7 x 29 = 203 LEDs und zwei reservierten Bytes, die für eine Erweiterung mit 12 beleuchteten Flaschen zur visualisierung der Uhrzeit vorgesehen sind.

Die Software auf dem Displaycontroller, dem Atmega48, besteht aus der Initialisierung aller Hard- und Software und einer leeren while(1)-Schleife (Listing 1).

```
int main(void) {
    init_display();
    init_spi();
    init_buffer();
    init_timer();
    //~ Interrupts erlauben
    sei();
    //~ Warte auf Interrupts
    while (1) { }
    return 0; }
```

Listing 1: Software des Displaycontrollers

Timinganalyse der Software auf dem Atmega48

Es war von vornherein klar, dass ein Mikrocontroller verwendet werden muss, der hauptsächlich schnell und billig ist. Der billigste Atmega von Atmel, den man laut Datenblatt bis 20MHz betreiben kann ist der Atmega48. Es wurde von Anfang an auf diesem entwickelt, da er in der Bastlerkiste bereits vorhanden war und weil er den Kriterien entspricht und auch vom Speicher ausreichend erschien. Erste Tests wurden auf dem Steckbrett mit dem internen 8MHz Oszillator gemacht. Schnell wurde aber klar, dass die Erhöhung auf die maximale Frequenz (was nur 2,5 mal so viele Rechenoperationen pro Sekunde sind im Vergleich zu vorher!) unumgänglich ist.

Um ein möglichst flimmerfreies Bild auf dem Display zu erzeugen muss die Multiplexfrequenz möglichst hoch sein; deshalb wurde durch praktische Versuche eine Timerfrequenz für den 8-bit Timer0 gesucht und eingestellt. Der Timer wird im Modus CTC mit einem Prescaler von 64 und einem Endwert von 255 betrieben (CTC = „Clear Timer on Compare“, sprich bei Erreichen des eingestellten Wertes löst der Timer einen Interrupt aus und startet automatisch von vorn). Zeilenfrequenz: 20*10^6Hz / (64*256) = 1220,7Hz. Bei 7 Zeilen entspricht das einer Bildwiederholfrequenz von 174,4Hz. vEs musste eine so hohe Frequenz gewählt werden, weil hier die LEDs immer nur einen Bruchteil einer Sekunde tatsächlich leuchten und ein kontinuierliches Leuchten ohne Flimmern und Flackern erwünscht ist.

Da der Timer nicht der einzige Interrupt auf dem Mikrocontroller ist, sondern auch die Daten, die über SPI empfangen werden und im Anschluss verarbeitet werden müssen, kann es passieren, dass sich die Interrupts und deren Verarbeitungszeit sich gegenseitig beeinflussen und stören. Das liegt auch maßgeblich

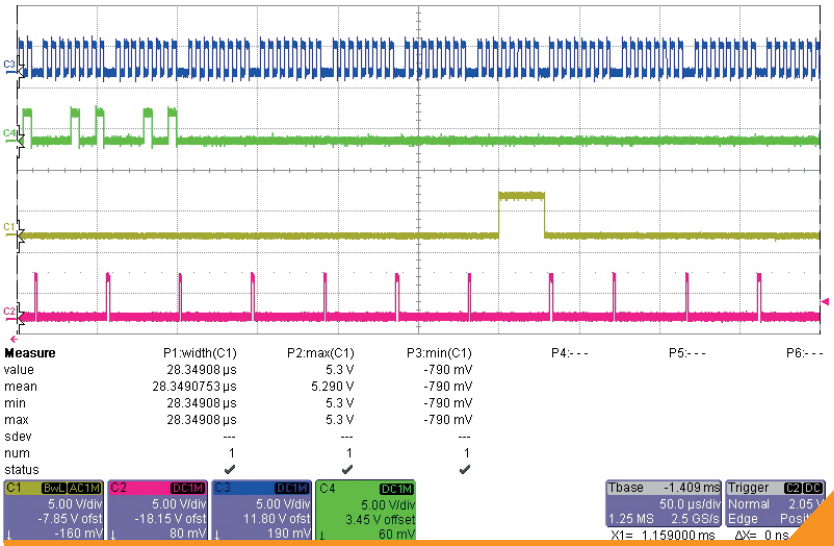


Abb. 4: Messungen zur Interruptkontrolle und SPI-Übertragungsgeschwindigkeit

daran, dass Atmegas keine Nested Interrupts oder Priorisierte Interrupts kennen. Dies bedeutet, dass der Interrupt bearbeitet wird, der zuerst aufgetreten ist und dass er nicht von einem anderen Interrup unterbrochen werden kann. Um dies sicherzustellen und um die endgültige SPI-Übertragungsgeschwindigkeit herauszufinden wurden einige Messungen angestellt (Abb. 4). Für die Messungen gilt folgende Legende:

- Kanal 1 (braun): Timer-ISR
- Kanal 2 (rot): SPI-ISR (pro empfangenem Byte ein Interrupt)
- Kanal 3 (blau): SPI-Clock (SCK)
- Kanal 4 (grün): SPI-Daten (MOSI)

Bei ersten Tests mit einer SPI-Geschwindigkeit von 0,5 Mhz war die Byteübertragung so schnell, dass bei laufender Timer-ISR zwei SPI-Interrupts auftreten konnten, aber keine ISR aufgerufen wurde um das erste übertragene Byte zu sichern. Dies bedeutete einen Datenverlust, was so nicht akzeptiert werden konnte. Die Bildgenerierung ist gegenüber dem Empfang der Daten priorisiert, also muss die Übertragungsrate so verändert werden, dass im Worst-Case, also bei quasi gleichzeitigem Auftreten der Interrupts (Timer zuerst), die Byte-Übertragungszeit länger dauert als die Timer-ISR. Als möglichst hohe, sichere Übertragungsgeschwindigkeit wurde 200 kHz ausgewählt (siehe Tabelle 1).

Byteübertragung	~ 44 µs
Timer ISR	< 33 µs
SPI ISR	< 4 µs

Tabelle 1: Zeitenübersicht

Es kann also garantiert werden, dass die Timer ISR nicht dazu führt, dass ein Byte verloren geht: 44us > 33us +4us. Was passieren kann ist, dass die Multiplexzeiten zweier Zeilen um 4 µS verschoben werden, sprich eine einzelne Zeile leuchtet 4 µs länger als normal und die andere kürzer als normal. Dies ist aber in der Praxis nicht sichtbar und kann vernachlässigt werden. Die Verschiebung des Datenempfangs durch die Timer-ISR um maximal 33 µs ist auch hier aufgrund der Trägheit des menschlichen Auges und des Nachleuchtens der LEDs irrelevant und nicht sichtbar.

Software auf dem Gnublin: clockd

Die Aufgabe der Software auf dem Gnublin ist es, jegliche Daten so aufzubereiten und an den Displaycontroller (= Atmega; im Folgenden einfach nur ‚Display') zu senden, damit ein Bild auf der LED-Matrix entsteht. Dabei wird zwischen drei verschiedenen Modi unterschieden:

- Zeitanzeige
- Textanzeige
- Rawdaten Anzeige / Bandit

Die Software besteht aus einem Server bzw. Daemon und aus theoretisch unbegrenzt vielen Clients, die Daten zur verfügung stellen. Der Daemon selbst stellt einen Grundsatz an Funktionen bereit, jegliche andere Funktion kann mit Hilfe entsprechender Clienten nachgerüstet werden: Anzeige der aktuellen Zeit sowie Anzeige und Verwaltung von einer unbegrenzten Menge an Texten, die nacheinander dargestellt werden. Ausführen einer Animation, die einen einarmigen Banditen darstellt zur Anzeige von Texten und Rawdaten wird eine Schnittstelle für die Clients in Form eines Unix-Sockets bereitgestellt, wobei hier eine grundsätzliche Priorisierung vorliegt:

- Text vor Uhrzeit
- Bandit vor Text

Clients mit Rawdaten vor interner Ausgabegenerierung Clients werden auf dem Socket nicht unterschieden und Daten werden in der Reihenfolge angezeigt, in der sie am Socket ankommen. Auch werden Daten, die auf dem Socket empfangen werden sofort ausgegeben, also ist der Client selbst für das Timing zuständig. Wird eine Ausgabe von einer höherpriorisierten Ausgabe unterbrochen, so wird die vorherige Aufgabe an der Stelle fortgeführt, an der unterbrochen wurde (Ausnahme: Anzeige der Uhrzeit; hier wird natürlich immer die aktuelle Zeit angezeigt).

Die Software besteht aus mehreren Threads, die alle verschiedene Aufgaben bewerkstelligen:

- Socket Kommunikation und Verarbeitung der externen Daten
- Thread für interne Datenausgabe
- Interrupt Thread, der GPIO-Pin 14 überwacht und den Banditen startet
- Signalhandler, der eingehende Signale empfängt und bei SIGTERM dafür sorgt, dass der Daemon ordnungsgemäß heruntergefahren wird

Folgende Aufgaben werden außerdem vom clock daemon erledigt:

- Logging von Fehlern und grundsätzlichen Events
- Initialisieren von externen Schnittstellen
- Umwandeln von Uhrzeit und Texten mit den integrierten Schriftarten

Da die Software als Daemon im Hintergrund läuft und an kein Terminal oder Standardinput mehr gebunden ist und trotzdem die Möglichkeit bestehen soll,

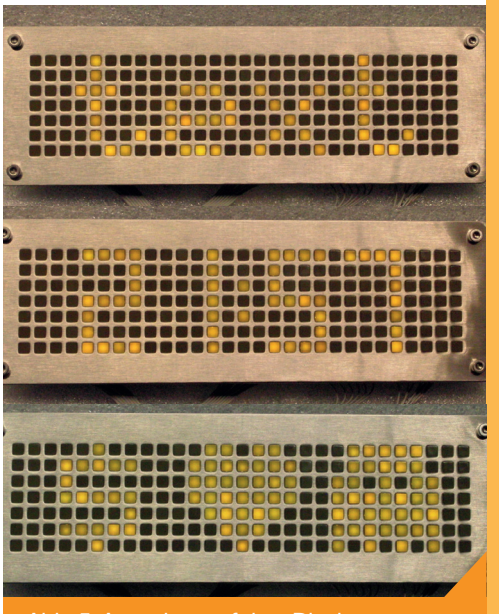


Abb. 5: Ausgaben auf dem Display

dass man das Programm sicher beenden können soll, gibt es die Möglichkeit Signale zu verwenden. Das Signal SIGTERM ist normalerweise für diesen Zweck vorgesehen. Das Signal kann zum Beispiel mit dem Programm kill gesendet werden. Um das Signal abzufangen ist ein Singalhandler notwendig. Abgesehen davon sollen andere Signale komplett ignoriert werden. Diese werden über eine Singalmaske deaktiviert. Anfangs wurde mit einem einfachen Signalhandler gearbeitet, der aber später in der Multithreadanwendung Probleme bereitete und kein sicheres Abfangen und Handlen der Signale ermöglichte und somit die sichere Beendigung des Programms nicht gewährleisten konnte. Daraufhin wurde ein eigener Thread für das Handlen von

Signalen implementiert. Im Mainthread werden alle Signale deaktiviert, bevor die anderen Threads gestartet werden, was bewirkt, dass diese die Signalmaske erben und kein Thread Signale empfängt. Anschließend aktiviert man wieder die Signale im Handlerthread, die man tatsächlich abfangen möchte - in diesem Fall SIGTERM. Die Aufgabe dieses Threads ist es nun, nur auf dieses Signal zu warten und beim Eintreffen die Statusvariable auf STOP zu setzen und sich danach selbst sofort zu beenden. Ein weiterer Knackpunkt ist das saubere Beenden des Hauptthreads, der auf dem Socket horcht und solange blockiert, bis sich ein Client auch auf das Socket verbindet. Eine Möglichkeit diesen Thread „aufzuwecken“ ist einfach das Socket zu schließen. Allerdings bedeutet das, dass er mit einem Error-Code zurückkehrt. Da dies weiteres Errorhandling bedeutet hätte und die Lösung auch nicht ganz sauber ist, wurde hier der Ansatz gewählt, einfach dasselbe Socket im Signalhandler noch einmal zu öffnen und sofort wieder zu schließen. Das verursacht, dass der blockierende Thread zurückkehrt und dann aber nicht auf den Empfang von Daten wartet, sondern den Status überprüft und bei STOP das Socket ebenfalls schließt und die Anwendung sauber beendet. Das beenden des Displaythreads gestaltet sich nicht sonderlich schwer, da dieser einfach nach jedem Frame den Status prüft und sich so mit einer maximalen Verzögerung von 0,5s auch selbst beenden kann.

Abb. 6: Die fertig aufgebaute „Jägermeister-Uhr“ mit der LED Matrix als Display

Der Thread, der auf den Interrupt wartet, blockiert auch beim Aufruf von poll(), allerdings kann dies Funktion mit einem Timeout aufgerufen werden, und aufgrund der übrigen Ressourcen wurde dieser als Mittelweg zwischen ressourcenschonend und rechenintensiv auf 2s gestellt und somit beendet sich auch dieser Thread nach maximal 2 Sekunden. Der Code des Signalhandlers ist hier [1] zu sehen.

Um Interrupts an den GPIO-Pins des Gnublin zu empfangen bzw. darauf zu reagieren, muss zuerst die Hardware entsprechend initialisiert werden. Das kann entweder über das Sysfs oder über Memory Mapped IO erfolgen; hier wurde der Zugriff über das Sysfs gewählt. Grundsätzlich müssen folgende Befehle ausgeführt werden um per Sysfs bei einem Interrupt benachrichtigt bzw. geweckt zu werden (hier die Shellbefehle):

```
echo 14 > /sys/class/gpio/export

echo „in“ > /sys/class/gpio/gpio14/direction

echo „falling“ > /sys/class/gpio/gpio14/edge
```

Zuerst wird der Pin 14 im Sysfs durch den Export angelegt, anschließend auf „Eingang“ gestellt und Interrupts auf fallenden Flanken aktiviert. Um nun beim Interrupt selbst benachrichtigt zu werden, kann jetzt mit der Funktion poll() auf dem Filedescriptor des Pin 14 gewartet werden, bis ein Interrupt auftritt. Zu guter Letzt muss oder kann der Pin beim Beenden wieder „unexportiert“ werden:

```
echo 14 > /sys/class/gpio/unexport
```

Das Ganze kann natürlich auch direkt aus dem C-Code [1] erreicht werden, wie es hier verwendet wird .Beachtet werden musste hier vor allem in welchem Modus die Dateien geöffnet werden, da zum Beispiel die export-Datei nur schreibend und nicht lesend geöffnet werden darf, da sonst der open()-Aufruf fehlschlägt. In der aktuellen Implementierung benutzt der clockd Unix Sockets, auf die ein oder mehrere Clients schreiben können um Daten an das Display auszugeben. Beispiele für solche Clients ist das Webinterface, bei dem das CGI-Skript die Textnachricht in das Socket schreibt oder Nachrichten, die per Cronjob zeitgesteuert angezeigt werden sollen. Hier ist der Code für einen einfachen Textclient, der den Text als ersten Parameter erwartet:

```
send(sock, &type, sizeof(char), 0);

send(sock, &length, sizeof(int), 0);

send(sock, argv[1], sizeof(char)*length, 0);
```

Der Webserver läuft ja schon, jetzt muss nur noch die Dateiendung .cgi in der Konfigurationsdatei richtig eingestellt werden. In der Datei /etc/lighttpd/conf-enabled/10-cgi.conf wird die folgende Zeile eingetragen, um CGI-“Programme“ aus C-Code in jedem Verzeichnis ausführen lassen zu können: cgi.assign = („.cgi“ => „“). Das gesamte Webinterface besteht im Moment aus zwei Dateien, nämlich einer Startseite (reine HTML-Datei) mit einem Formular und der CGI-Datei [1], der der eingegebene Text als Zeichenkette übergeben wird. Sie dekodiert den Text und übergibt ihn über das Socket an den Daemon.

Der Quellcode des Banditen entstammt einer früheren Programmierübung und wurde nur minimal angepasst und in das Programmkonstrukt eingepflegt. Zusätzlich zu den enthaltenen Symbolen (Abb. 5) wurde noch das GNUBLIN-Logo eingefügt (Abb. 1).

Einbindung in ein größeres Projekt, aktueller Status und Ausblick

Das Display war von Anfang an nur als Teilprojekt eines größeren privaten Gesamtprojekts geplant, wobei als Projektarbeit in der Vorlesung ‚Embedded Linux' eben nur das Display und die Software behandelt wurden. Das Gesamtprojekt stellt eine „Jägermeister-Uhr" dar, die aus 12 beleuchteten Jägermeisterflaschen, sowie einem Logo mit einstellbarer RGB-Beleuchtung und eben dem Display besteht. Die Uhr wiegt etwa 25kg und misst 70cm im Durchmesser (Abb. 6 und 8).

Zum Zeitpunkt der Fertigstellung des Berichts Mitte Juli waren noch nicht alle Probleme, was zum Beispiel das Logo oder Platzprobleme anging, gelöst und noch nicht alle Hauptfeatures vorhanden. Die Probleme wurden inzwischen alle gelöst und folgende Funktionen bzw. Eigenschaften wurden noch implementiert:

- Generierung von Leuchtmustern auf den Flaschen
- Zusätzliche Symbole und Highscoregenerierung für den einarmigen Banditen
- Designtechnische Aufwertung und Anpassung des Webinterfaces auf mobile Endgeräte (siehe Abb. 7)

Zusätzlich wurde für die Aktivierung der Slotmachine ein Buzzer auf einem Ständer an die Uhr angeschlossen, der für den nötigen Spielspaß sorgt und die Uhr auf eine Grundplatte montiert, mit der sie später an einer Wand ihren endgültigen Bestimmungsort finden soll. Folgende Wunschliste existiert dennoch für zukünftige Erweiterungen:

- Highscorelisten auf dem Webinterface mit Namenseingabe und Datenbankanbindung
- Erweiterung auf Internet-Sockets inklusive Android-App zur Texteingabe statt dem Webinterface
- evtl. weitere Spiele und Anbindung weiterer IO-Hardware; Spielbeispiele: Snake, Tetris, ...

Selbst die Firma Jägermeister war begeistert von der Uhr und hat freundlicherweise die Einweihungsfeier etwas gesponsort ;) Das Projekt war eine sehr zeitintensive, aber auch lehrreiche Erfahrung, denn wie heißt es so schön: „Übung macht den Meister".

Abb. 7: Neues Webinterface

Abb. 8: Das fertige Projekt

Links

[1] https://elk.informatik.fh-augsburg.de/dav/elinux-13-berichte/Fink/

[2] Timer: http://www.embedded-linux.co.uk/tutorial/periodic_threads

[3] Autoload Module: http://wiki.ubuntuusers.de/Kernelmodule#start Daemonize:

[4] http://www.itp.uzh.ch/~dpotter/howto/daemonize

[5] http://infohost.nmt.edu/~eweiss/222_book/222_book/0201433079/ch13lev1sec5.html

[6] Syslogd: http://www.codealias.info/technotes/syslog_simple_example

[7] GPIO/Interrupts: http://wiki.gnublin.org/index.php/GPIO

[8] Debugging mit cgdb: http://sourceware.org/gdb/onlinedocs/gdb/Forks.html

[9] 5x7 Font: http://sunge.awardspace.com/glcd-sd/node4.html

[10] CGI: http://www.cs.tut.fi/~jkorpela/forms/cgic.html

[11] Locale: http://www.thomas-krenn.com/de/wiki/Perl_warning_Setting_locale_failed_unter_Debian Init-Skript http://wiki.ubuntuusers.de/Dienste http://wiki.debian.org/LSBInitScripts

GNUBLIN für die Automatisierung

Ein Projekt kurz vor der Serienreife

Martin Trapp, Dipl. Ing (FH) <info@it-trapp.com>

Wie in jeder Branche, so auch im Maschinen- und Anlagenbau, steigt schon seit einigen Jahren der Preisdruck rasant – weltweit. Es ist also ständig notwendig preiswerte (nicht billige) Komponenten zu finden. Sehen wir uns in diesem Zusammenhang an worüber in oben genanntem Maschinen- und Anlagenbau in Bezug auf die Steuerungs-Elektronik geredet wird: Windows und SPS. Bis auf wenige Ausnahmen hört und sieht man zur Zeit nichts von Linux Alternativen.

Einleitung

Uns Techniker und Ingenieure kann es eigentlich nicht so unendlich erfreuen. Denn wir wissen um die Stabilität und um den Preis und um die Flexibilität und um die Leistungsbreite und ... von Linux. Und die Kaufleute besonders die Einkäufer sollte es eigentlich auch nicht freuen - in sofern sie offenen Auges durch die Welt gehen.

Soweit die betroffenen Personenkreise. Und die Technik?

SPS? Hand auf's Herz: Es tut sich einiges; aber das was zur Zeit für immerhin noch sehr nennenswerte Kosten auf dem Markt angeboten wird entspricht längst nicht immer zwingend dem Stand der Technik.

Windows? Die Lizenzkosten sind normalerweise nur sehr bedingt eine Freude. Zumindest nicht für Maschinen- und Anlagenbauer. Und für die Endkunden ganz sicher auch nicht. Das Preis – Leistungs Verhältnis ist also in beiden Fällen aus den jeweiligen Gründen eher weniger attraktiv. Also Linux. Und welche Hardware?

GNUBLIN?

Ein leistungsfähiges Stück Elektronik. Der Preis ist fair (das macht ein über durchschnittlich gutes Preis/Leistungsverhältnis). Und es bietet von sich aus schon einige Schnittstellen, die in der Automatisierungsbranche bekannt sind und die sehr häufig eingesetzt werden.

Was braucht man mehr? Noch mehr standardisierte Schnittstellen. Fein. Dadurch dass die bekanntesten und wahrscheinlich auch meist verwendeten Schnittstellen SPI und I2C bereits integriert sind ist die Integration weiterer Schnittstellen kein Hexenwerk. Es sind außerdem noch einige digitale I/O's verfügbar (zumindest beim GNUBLIN DIP) mit denen man zur Not auch exotische Schnittstellen realisieren kann.

Um GNUBLIN DIP geht es in diesem Zusammenhang sowieso (wegen der digitalen I/O's – von denen man nie genug haben kann). Aber auch aus einem anderen Grund.

Die Maschinen- und Anlagenbauer lieben keine elektronischen Konstrukte, die aus vielen kleinen Platinchen bestehen, die möglicherweise untereinander noch unterschiedliche Abmessungen haben. Die Maschinen und Anlagen sind oft starken Vibrationen ausgesetzt (die sie häufig selbst produzieren). Viele Platinen bedeuten teure mechanische Verbindungselemente, teure mechanische Montage und teure Bauteile für die elektrischen Verbindungen.

Die Gängige und wohl auch einzig akzeptable Lösung ist daher: alles auf eine PCB. Jetzt kommen wir zum Punkt: die DIP Version kann leicht auf einer Trägerplatine verlötet werden, auf der dann die weitere Elektronik verbaut ist, mit der man den Maschinen- und Anlagenbauern ein attraktives Controller Board liefern kann. Hier sind wir schon bei der Konzeption. Die Gedanken dazu sollen jetzt folgen.

Grundlegendes zur Hard- und Software Konzeption

Die erste Frage lautet: Was gibt es denn aus Praxis-Sicht an grundlegenden Anforderungen an eine Steuerungs-Elektronik (unabhängig davon welches Betriebssystem eingesetzt wird)? Wichtigste Antwort: die Stromversorgung.

Egal welches Filesystem wir uns aus diesem Sichtwinkel anschauen (und alle modernen Systeme scheinen sehr ausfallsicher zu sein); wirklich sicher ist keines. Der Grund dafür liegt aber nicht in den Speichersystemen an sich sondern in der oben erwähnten Stromversorgung. Und egal wie pfiffig das Filesystem ist: Ein Stromausfall zum ‚richtigen' Zeitpunkt ist der ideale Killer. Sorgen wir also für eine möglichst intelligente aber bezahlbare Stromversorgung um dieses Risiko zu umgehen. Es bietet sich einer von diesen kleinen 8 Bit Mikrocontrollern an um die Eingangsspannung und kritische Spannungen zu überwachen und die verschiedenen Versorgungsspannungen an- und abzuschalten (siehe auch Abbildung 1). Solche Controller sind auch bei Einzelabnahme unter 2 Euro pro Stück erhältlich. Die restlichen Bauteile zur Realisierung einer solchen Beschaltung dürfte kaum mehr als 10 Euro kosten; das sollte uns die Betriebssicherheit wert sein. Immerhin soll der Controller möglicherweise Maschinen oder Anlagen steuern deren Anschaffungspreis leicht mehrere zehntausend Euro erreichen oder in manchen Fällen auch übersteigen kann. Ein Produktionsausfall von mehreren Stunden kann noch weit höhere Kosten verursachen; und dafür lässt sich niemand gerne haftbar machen.

Welche ‚Teilnehmer' sollten sonst noch auf dem Controller versammelt sein?

Über möglichst viele Schnittstellen sprachen wir schon. GNUBLIN selbst bietet ja schon eine beachtliche Anzahl von Schnittstellen. Eine USB Schnittstelle, ei-

nen I2C- , einen SPI BUS, einen gemultiplexten 8-Bit-A/D-8Wandler, einen PWM Ausgang und eine Reihe von digitalen I/O's.

Die SPI Schnittstelle könnte noch einige Chip Select - Leitungen gebrauchen. Das erreichen wir am besten mit einer Port Erweiterung z.B. über I2C (ohne das wir digitale I/O's -verschwenden).

Dann könnten wir auf gleiche Weise (ebenfalls ohne Verschwendung digitaler I/O's) einen Baustein hinzufügen der es ermöglicht externe Interrupts zu verarbeiten.

Mit Hilfe der bereits vorhandenen Schnittstellen (SPI oder I2C) können wir noch einen 1-wire-Bus, einen CAN-Bus, eine RS232- und eine RS485 Schnittstelle hinzufügen. Eine Real Time Clock (RTC) sollte nicht fehlen. Sie erspart uns die lästige Eingabe von Datum und Uhrzeit bei jedem Neustart – auch dann wenn unser Controller nicht mit dem Internet verbunden ist (eine Internet-Verbindung muss vom Betreiber schließlich nicht immer erwünscht sein). Bleibt noch die Frage: Wie soll dieses Controller-Board angesteuert werden?

Es wird zur Zeit an einem einfachen Interpreter geschrieben, der Kommandos von einer (Kommando-) Schnittstelle z.B. RS232 entgegen nimmt, analysiert und an die jeweiligen Schnittstellen weiter leitet und der in umgekehrter Richtung Informationen von den Schnittstellen liest und und an die (Kommando-) Schnittstelle übergibt.

Mit dieser Konzeption ist der komplette Schnittstellenvorrat des Controller-Bords von außerhalb erreichbar. Das Controller-Board mitsamt der/den daran angeschlossenen Maschine(n) kann in eine größere Steuerungs-Struktur eingebunden werden.

Der Interpreter (wenn fertig gestellt) kann aber nicht nur als ausführbares Programm installiert werden, was den Betrieb (s.o.) ermöglicht. Er steht auch als Unit zur Verfügung (er ist vollständig in free pascal codiert). Mit Hilfe dieser Unit ist es möglich den Controller eigenständig zu betreiben um mit ihm eine Maschine oder Anlage zu steuern.

Nun ist auf dem GNUBLIN DIP reichlich Speicherplatz vorhanden – und selbst wenn der nicht ausreicht lässt er sich z.B. über USB mit externem Speicher wie etwa USB Sticks oder Festplatten praktisch beliebig erweitern. Dies ermöglicht große Programmkonstrukte mit denen sich auch sehr komplexe Abläufe programmieren lassen.

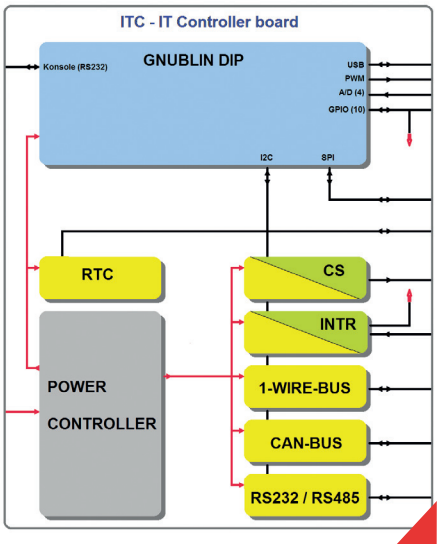


Abb. 1: Blockschaltbild

Denkbar wäre auch, durch wechseln eines USB Sticks bei laufender Produktion, die Produktionsdaten zu ändern. Gleiches würde sich auch durch Einspielen von Daten über ein Netzwerk realisieren lassen. Damit könnte der Controller nicht nur zur Steuerung an einem autarken Arbeitsplatz sondern auch problemlos an halbautomatischen oder automatischen Produktionstrassen eingesetzt werden.

Schlussgedanken

Die Entwicklung ist noch nicht komplett abgeschlossen und es werden möglicherweise an der einen oder anderen Stelle noch leichte Änderungen vorgenommen. Ein erster Entwurf der Platine ist in Abbildung 2 zu sehen.

Im Vordergrund der Konzeption standen folgende Anforderungen: Es sollte ein zuverlässiges und flexibles System entwickelt werden. Dieses System sollte einen attraktiven Preis haben. Durch den Einsatz von GNUBLIN DIP ist das gelungen.

Um der Flexibilität auch im Preis gerecht zu werden ist geplant verschiedene Bestückungsvarianten anzubieten. So sollen nicht benötigte externe Schnittstellen auf Wunsch unbestückt bleiben. So kann der Preis des Controllers noch günstiger werden.

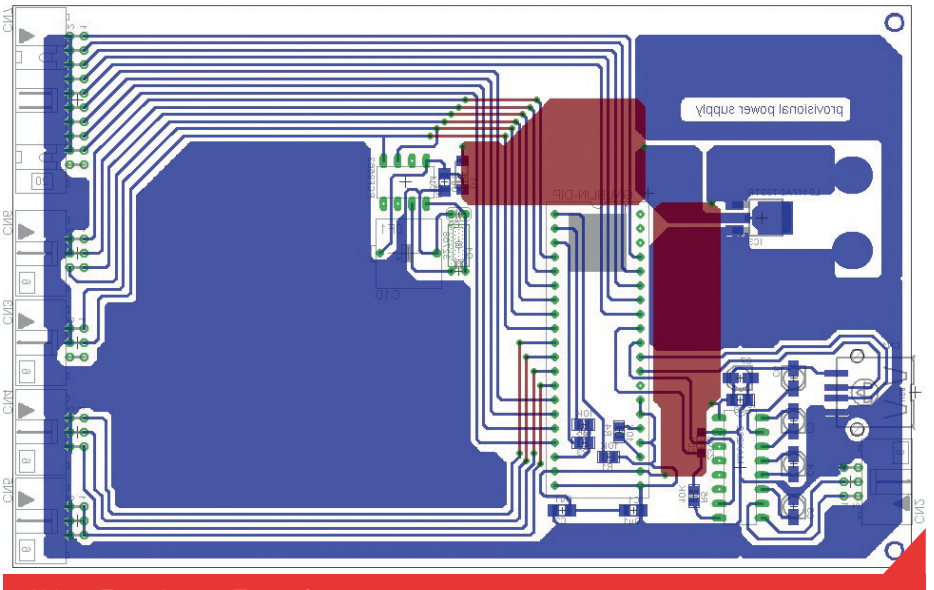


Abb. 2: Erster Layout Entwurf

Zur Software:

die Units zum Ansteuern der einzelnen Schnittstellen werden (wenn sie fertig getestet sind) auf www.it-trapp.com veröffentlicht werden.

Es ist geplant, dass die ersten Controller Anfang 2014 geliefert werden können.

GNUBLIN-Case

Anleitung zum Bau eines Gehäuses vfür die GNUBLIN Linux Boards

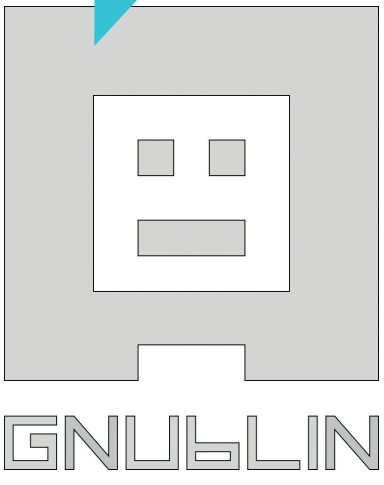
Torsten Hees <A1200@gmx.net>

Fertigung der Plexiglasscheiben

Der einfachste Weg die Plexiglasscheiben (Abb. 2) zu fertigen ist diese sich passend zugeschnitten bei einem Händler zu bestellen. Das passende Maße für die GNUBLIN Linux Boards ist 97 mm x 80 mm. Bei diesen Maßen ragt die Scheibe auf allen Seiten 5 mm über das Board. Die abgerundeten Ecken haben ebenfalls einen Radius von 5 mm.

Tipp: Wenn die erste Scheibe gebohrt ist, diese als Schablone für die zweite Scheibe nutzen.

Abb. 1: Der Gnublin Aufkleber



Wenn die passenden Plexiglasscheiben bereit liegen, müssen diese nur noch mit Löchern für die Schrauben versehen werden. Die genauen Stellen für die Bohrungen können der Zeichnung entnommen werden.

Als nächster Schritt müssen die Bohrlöcher noch mit einem Senkkopfbohrer bearbeitet werden, so dass sich die Senkkopfschrauben komplett im Plexiglas versenken lassen und zwar dreimal auf der Oberseite der oberen Plexiglasscheiben und dreimal auf der Unterseite der unteren Plexiglasscheibe.

Benötigte Komponenten

- 2x Plexiglas Scheiben der Stärke 2 mm
- 3x Abstandsbolzen M3 25mm mit 2x Innengewinde
- 3x Abstandsbolzen M3 10 mm mit 1x Innen-/Außengewinde
- 6x Senkschrauben M3
- 4 Gummifüße
- Klebefolie

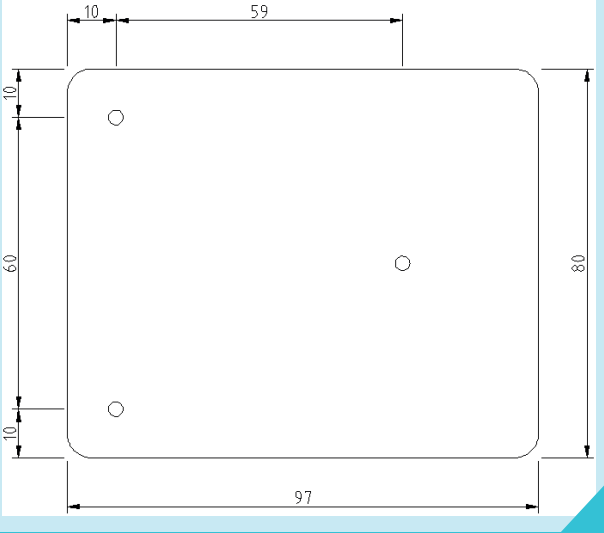


Abb. 2: Die Plexiglasscheibe

Zusammenbau

Die Schutzfolie von den Plexiglasscheiben entfernen. Die 10 mm Abstandsbolzen auf die untere Plexiglasscheibe schrauben und das GNUBLIN-Board auf die Abstandsbolzen legen und mit den 25 mm Abstandsbolzen verschrauben. Die obere Plexiglasscheibe nun auf die Abstandsbolzen schrauben und die Gummifüße auf die Unterseite kleben. Tipp: Die Gummifüße nicht über die Schrauben kleben, so kann das Gehäuse bei Bedarf ohne Probleme wieder zerlegt werden.

Aufkleber

Die Grafik (Abb. 2) für die Oberseite liegt im Format für das Open Source Vektor Zeichenprogramm Inkscape vor.

Nun gibt es zwei Möglichkeiten die Grafik auf das Gehäuse zu bekommen. Entweder man lässt sich die Grafik im Geschäft per Folien-Cutter ausschneiden oder man druckt sie spiegelverkehrt aus, klebt diese auf die Rückseite der Klebefolie auf und schneidet die Folie per Hand.

Die fertig ausgeschnittene Folie nun auf die obere Plexiglasscheibe kleben.

Tipp: Für das übertragen der Folie auf die Plexiglasscheibe sollte man eine Transferfolie verwenden.

Viel Spaß mit dem Gehäuse

Abb. 3: Die fertigen Gnublin Gehäuse

Anhang

[1] CAD Plexiglasscheiben (LibreCAD) und GNUBLIN Logo im Inkscape Format
https://github.com/embeddedprojects/gnublin-downloads/tree/master/gnublin-case

www.MEsstechnik24.de

Deutschlands großer B2B Messtechnik-Web-Shop

Riesen-Spektrum an Tisch- und USB-PC-Oszilloskopen, Multimetern, Signalgeneratoren, Stromversorgungen, Spektrum-Analysatoren, TDR, Komponenten für die Messwerterfassung, Motorsteuerung, Schnittstellen.

Produkte namhafter Hersteller: Acromag, Adlink, Agilent, CableEye, Cleverscope, HBM, Icron, LabJack, Pico Technology, Pro-Dex OMS, Rigol, VSCOM/Vision Systems und viele mehr.

www.meilhaus.de

www.MEsstechnik24.de

MEILHAUS ELECTRONIC

Meilhaus Electronic GmbH
Fischerstraße 2
82178 Puchheim/Germany
www.meilhaus.com
www.MEsstechnik24.de
TEL **(0 89) - 89 01 66-0**
FAX **(0 89) - 89 01 66-77**
E-Mail **sales@meilhaus.com**

Marktplatz / Neuigkeiten

Die Ecke für Neuigkeiten und verschiedene Produkte

Aufruf zum Wettbewerb: Open-Source Protokoll für das Internet der Dinge!

Man möchte einfach eine Lampe per Browser ansteuern oder schnell über die Ferne die Rollos schließen. Ein einfaches Protokoll für das praktische Internet der Dinge wird benötigt. Nach kurzer Recherche stellt man fest, dass eigentlich das „Hello-World" Beispiel etwas erweitert für das eigene Netzwerk locker ausreicht. Außerdem ist es ganz schön viel Arbeit sich für so ein „kleines Problem" in fremde Protokolle einzulesen. Und genau diese Gegebenheit war der Auslöser für die Idee für diesen Wettbewerb.

Die Suche nach dem IOT Protokoll!

In einer gemeinsamen Initiative von Elektor, embedded projects Journal, Partnern und Open-Source Projekten suchen wir nach einem minimalen Protokoll für die Verknüpfung der vielen einzelnen existierenden Lösungen.

Alle die sich berufen fühlen, oder gute Implementierungen kennen sind aufgerufen Ideen und Vorschläge einzureichen. Es soll am Ende ein ganz einfach zu verstehendes Protokoll gefunden werden, mit dem jeder genau die einfachen Sachen schalten kann. Wenn Sich viele Entwickler an das Protkoll halten, kann man einfach verschiedene Projekte und System verknpüfen.

Als zentrale Informationsseite haben wir die Homepage http://www.iot-contest.com eingerichtet.

Partner für den Wettbewerb

Neben Ideen und Vorschlägen suchen wir Partner, die den Wettebewerb begleiten möchten und anschliessend das Protokoll in eigene Projekte, Produkte oder Anwendungen integrieren.

Alle Partner werden auf der Homepage genannt und verlinkt. Damit das Internet der Dinge Protokoll sich auch gut verbreitet muss jeder Partner ebenfalls einen Link auf die Wettbewerbsseite http://www.iot-contest.com auf der eigenen Homepage erstellen.

Auf der Internetseit, im Embedded Projects Journal und in Elektor berichten wir regelmäßig über den aktuellen Stand. Bei Fragen stehen wir ebenfalls gerne zur Verfügung. Genaue Infos zur Roadmap, den Anforderungen usw. findet man auf der Homepage.

Einladung zum kostenlosen WE Training: Passiv trifft Elektromechanik 27.03.2014 bei embedded projects

Ideal für alle Entwickler und Interessierte zum Thema:

Speicherdrosseln, HF-Verhalten von Siftleisten, Filter für DC/DC Schaltegler und einer Live-Demonstration

Perfekt für alle die, die mal genau wissen wollen wie man Schaltungen korrekt befiltert und Schaltregler dimensioniert, bieten wir im Rahmen unserer kostenlosen Schulungen gemeinsam mit Würth Elektronik eiSos GmbH & Co. KG dieses Thema am 27.03.2014 in unserem Büro an.

In einer gemütlichen Runde ist wie immer für das leibliche und geistige Wohl gesorgt.

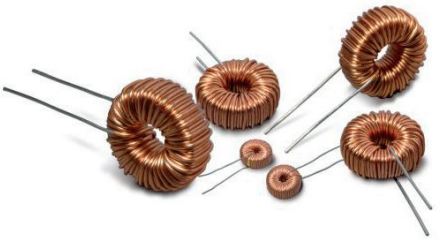
Das Training ist von Ingenieuren und Technikern für Ingenieure und Techniker auf praxisbezogene Inhalte konzipiert, um Ihnen eine Hilfestellung bei der Entwicklung und sicheren Markteinführung von Geräten zu geben.

Dabei werden sowohl Themen aus dem Bereich der Passiven Bauelemente wie auch Themen, die die Elektromechanik betreffen, behandelt.

Themenschwerpunkte sind:

- Designtipps für Speicherdrosseln
- HF Verhalten von Stiftleisten
- Filter für DC/DC-Schaltregler mit Live-Demonstration
- Tipps & Tricks für Entwickler

Bei Interesse melden Sie sich bitte unter info@embedded-projects.net per E-Mail an.



Interesse an einer Anzeige?

info@embedded-projects.net

embedded-projects
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Werdet aktiv!

Das Motto: Von der Community für die Community !

Das Magazin ist ein Open Source Projekt.

Falls Du Lust hast, Dich an der Zeitschrift durch einen Beitrag zu beteiligen, würden wir uns darüber sehr freuen. Schreibe deine Idee an:

journal@embedded-projects.net

Regelmäßig

Die Zeitschrift wird über mehrere Kanäle verteilt. Der erste Kanal ist der Download als PDF - Datei. Alle Ausgaben sind auf der Internetseite [1] verfügbar. Diejenigen, die lieber eine Papierversion erhalten möchten, können den zweiten Kanal wählen. Man kann sich dort auf einer Internetseite [2] in eine Liste für die gesponserten Abos oder ein Spendenabo eintragen. Beim Erscheinen einer neuen Ausgabe wird dank Sponsorengeldern an jeden auf der Liste eine Ausgabe des aktuellen Journal versendet. Falls man den Versandtermin verpasst hat, kann man das Heft auch über einen Online - Shop [2] beziehen.

[1] Internetseite (Anmeldeformular gesponserte Abos):
http://journal.embedded-projects.net

[2] Online - Shop für Journal:
http://www.embedded-projects.net

Sponsoren gesucht!

Damit wir weiterhin diese Zeitschrift für jeden frei bereitstellen können, suchen wir dringend Sponsoren für Werbe- und Stellenanzeigen. Bei Interesse meldet Euch bitte unter folgender Telefonnummer: 0821 / 2795990 oder sendet eine E-Mail an die oben genannte Adresse.

Bitte frei machen

embedded projects GmbH
Holzbachstraße 4
D - 86152 Augsburg

Name / Firma

Straße / Hausnummer

PLZ / Ort

Email / Telefon / Fax

◯ Ich möchte jede zukünftige Ausgabe erhalten

◯ Wir möchten als Hochschule / Ausbildungsbetrieb jede weitere Ausgabe bekommen. Bitte gewünschte Anzahl der Hefte pro Ausgabe ankreuzen. 5 10

◯ Ich möchten im embedded projects Journal werben oder eine Stellenanzeige aufgeben. Bitte schicken Sie mir Infomaterial, Preisliste etc. zu.

Impressum

embedded projects GmbH
Holzbachstraße 4
D-86152 Augsburg
Telefon: +49(0)821 / 279599-0
Telefax: +49(0)821 / 279599-20

Verteilt durch:

embedded projects GmbH
HARDWARE FOR PROJECTS

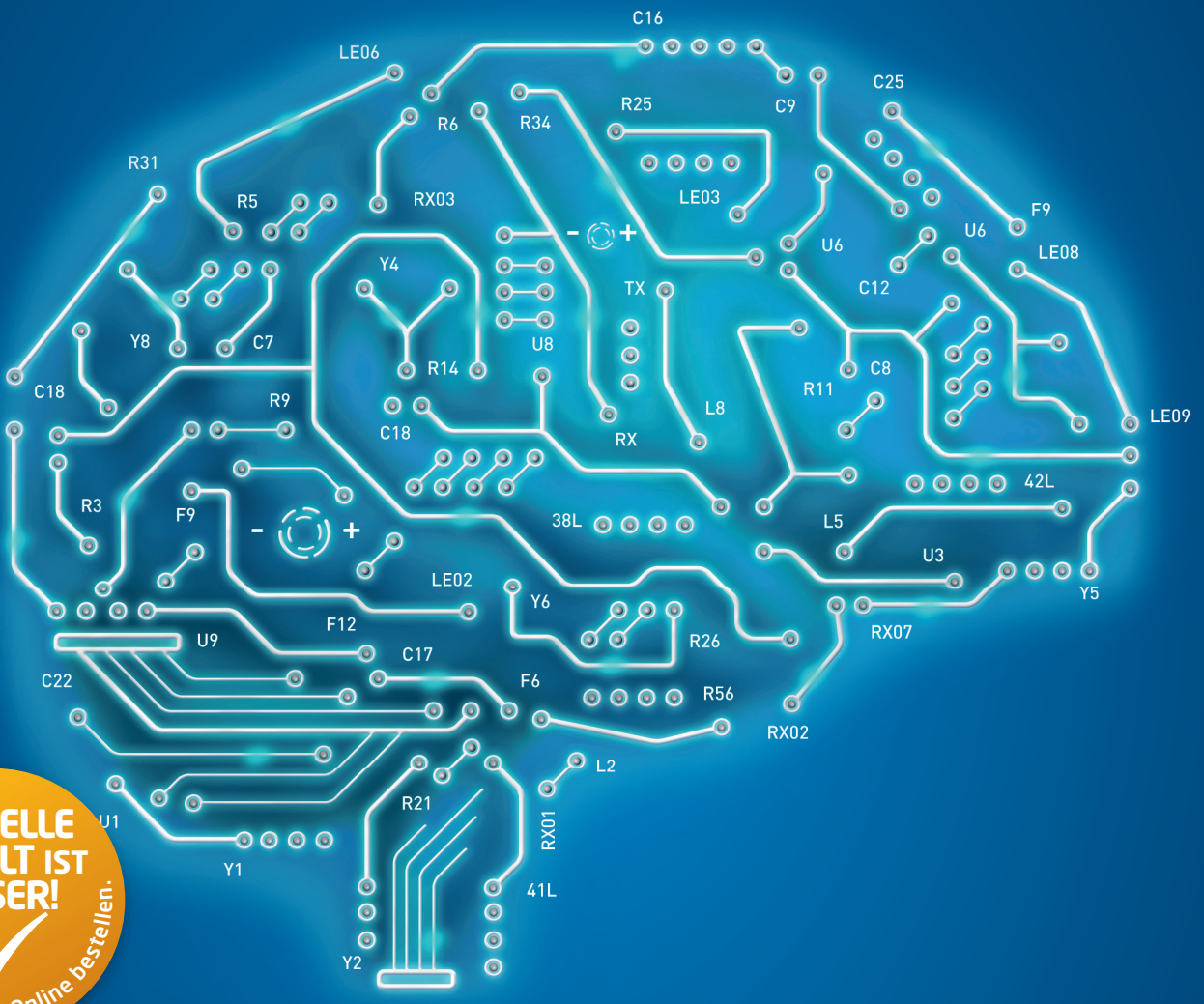
Veröffentlichung: 4x / Jahr
Ausgabenformat: PDF / Print

Auflagen Print: 2500 Stk.
Einzelverkaufspreis: 1 €

Layout / Satz: EP
Druck: flyeralarm GmbH
Titelbild: Torsten Hees

UNSER WICHTIGSTES WERKZEUG.

LEITERPLATTEN WEITER GEDACHT.



SCHNELLE VIELFALT IST BESSER!
Online kalkulieren. Online bestellen.

LEITON
RECHNEN SIE MIT BESTEM SERVICE

Wir denken weiter. Denn als lösungsorientiertes Unternehmen möchten wir unsere anspruchsvollen Kunden immer wieder mit neuen Ideen und sinnvollen Innovationen begeistern. Unsere Vision, **ein führender Online-Anbieter von Leiterplatten** zu sein, haben wir nicht nur im Kopf – wir setzen sie auch um. Und zwar mit erstklassigen Leistungen in den Bereichen **Service, Fairness, Technologie, Auswahl, Komfort, Geschwindigkeit und Zuverlässigkeit.** Unsere Webseite bietet eine enorme Auswahl an Lösungen in der Online-Kalkulation sowie weiterführendes Wissen, Informationen und Entwicklerwerkzeuge. Und sollte einmal etwas sehr Ausgefallenes gefordert sein, finden wir gerne **Ihre persönliche Lösung.** Für unsere Kunden heißt das: Sie können bei LeitOn immer mit bestem Service rechnen.

LeitOn GmbH	**www.leiton.de**	**kontakt@leiton.de**	Info-Hotline **+49 (0)30 701 73 49 0**