www.elektor-magazine.de

magazine

Januar/Februar 2015 | Nr. 529/530

elektor

EXTRA-STARKE WINTER-AUSGABE
132 SEITEN PRAKTISCHE ELEKTRONIK!

ARM-Controller für Einsteiger • Platino-Funktionsgenerator • Thermometer mit Bluetooth • Prototyping-Board für Wireless-Module • Open-Source-Programmer mit Webinterface

J²B Synthesizer

+ Gratis Webinar

Open Source und Open Hardware!

Logiktester • LED-Treiber für Glühlämpchen • FM-Sender • FM-Empfänger • IIR-Sinus-Generator • Feuchte/Temperatur-Sensor Breakout-Board

G3078 E

(D) € 14,90 | CHF 27,00 | (A, B, L) € 16,00

4 198630 314904 02

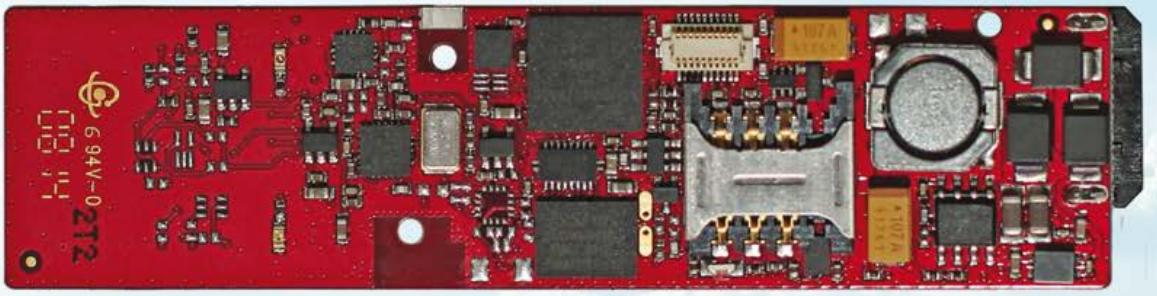
Wo ist Ihr Fahrzeug? Ihr Paket? Ihre Maschine? Finden Sie es mit TraceME!

Das neue Modell TM-203

TM-203 ist die neueste Ergänzung unserer TraceME-Produkt-Reihe.

Besondere Features:

- Sehr kleiner GPS-Tracker mit allen Features (87 x 22 x 7 mm)
- Wiegt nur 15,5 g, inkl. GPS-, Mobilfunk- und HF-Antennen
- Besonders niedriger Stromverbrauch, bis hinab zu 3 µA
- Positionsbestimmung via LBS sogar innerhalb von Gebäuden
- Portabel oder extern anschließbar zur Integration der Sensoren
- On-Board-Sensoren für Kompass, Temperatur und Beschleunigung
- Weltweite Abdeckung: Quad-Band-GSM/GPRS - Glonass/GPS – WLAN
- Entspricht zusätzlichen Standards von Versicherern (z.B. SCM und Insert)
- Entspricht den Sicherheits-Standards der Automobil-Industrie (E-mark)



Wir möchten Sie gerne zu einem Besuch auf unserer Webseite einladen. Lassen Sie sich von den Möglichkeiten inspirieren, die KCS-TraceME-Produkte für Ihr Projekt bringen können. Ob Sie an Personal Tracking, der Ortsbestimmung von Fahrzeugen, dem Objektschutz oder fortschrittlichen M2M-Lösungen interessiert sind – KCS TraceME hat die Lösung für Sie!

Wir bieten GPS-Tracker mit GPS/GPRS und erweiterter I/O. Damit können Sie eine breite Palette an Objekten, Maschinen oder Fahrzeugen aus der Ferne verfolgen oder aufspüren. KCS TraceME bietet aktuell die flexibelsten und anpassungsfähigsten Track-n-Trace-Lösungen. Unsere Kunden setzen TraceME-Hardware für die unterschiedlichsten Aufgaben ein. Wir bieten echte Allround-Lösungen, die andere Produkte in den meisten Aspekten übertreffen.

- KCS-TraceME-Produkte gibt es in vielen Ausführungen (Key fob, OWM, Alu- und kundenspezifische Gehäuse)
- GPRS – UMTS – SMS – GSM – WLAN – RFID – E-Mail
- Voll konfigurierbar mit Administrations-Tool
- Event-gesteuert, kann kundenspezifischen Code verarbeiten, fernkonfigurierbar
- Verbindet M2M mit jedem anderen Gerät
- G-Schock-Erkennung, Temperatur, Drehung, Bewegung, iButton, Handy, Kamera
- Die kompletten und detaillierten Optionen finden Sie auf unserer Webseite!

Sprechen Sie mit uns! Wir diskutieren gerne mit Ihnen über Ihre Ideen!

Anfragen von OEMs & Distributoren willkommen!

www.Trace.ME

Jetzt anmelden und
Tickets sichern!
embedded-world.de

Nürnberg, Germany
24. – 26.2.2015

embedded world 2015
Exhibition&Conference
... it´s a smarter world

DAS Treffen der Embedded-Community!

Die weltweit größte Veranstaltung für Embedded-Technologien bringt die Akteure der Embedded-Branche miteinander ins Gespräch.

Seien auch Sie dabei, wenn Kontaktpflege und Netzwerken auf internationaler Ebene groß geschrieben und Trends gesetzt werden.

Medienpartner

elektroniknet.de
computer-automation.de
energie-und-technik.de
MEDIZIN-UND-elektronik.DE
Markt&Technik
DESIGN & ELEKTRONIK KNOW-HOW FÜR ENTWICKLER
Elektronik
Elektronik automotive
ENERGIE & TECHNIK
Computer & AUTOMATION
MEDIZIN elektronik

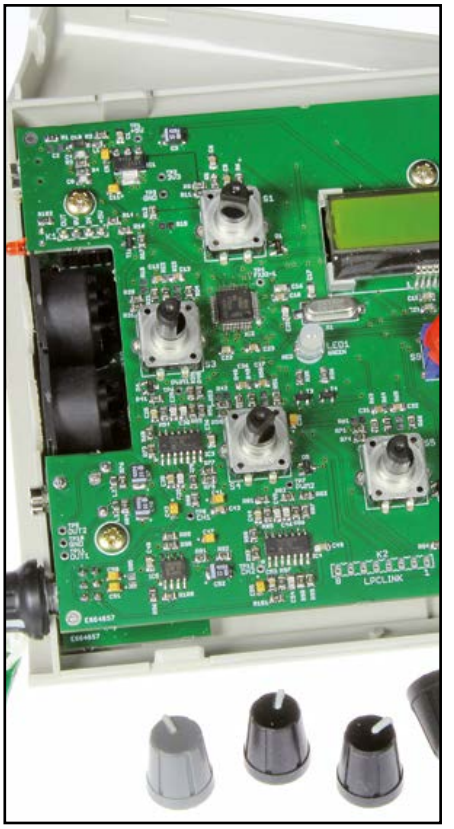
Veranstalter Fachmesse
NürnbergMesse GmbH
Tel +49 (0) 9 11. 86 06-49 12
besucherservice@nuernbergmesse.de

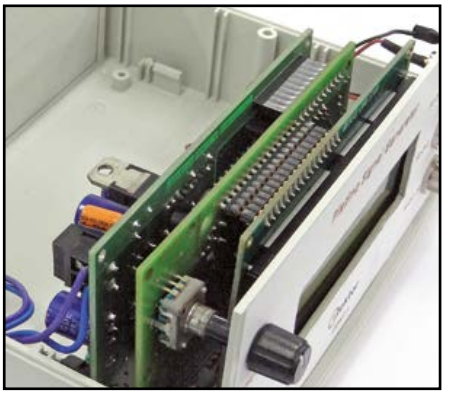
Veranstalter Kongresse
WEKA FACHMEDIEN GmbH
Tel +49 (0) 89. 2 55 56-13 49
info@embedded-world.eu

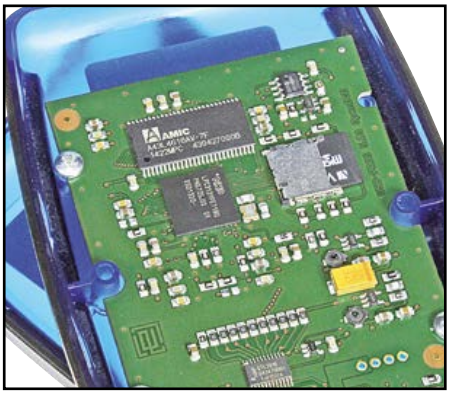
NÜRNBERG MESSE

Inhalt

elektor magazine







Labs

70 Microchip/Hillstar-Entwicklungskit

72 GestIC & 3D-TouchPad Kurs (2)
Diesen Monat geht es weiter mit dem Anschluss des Gesten-Erkennungs-Chips MGC3130 an einen Raspberry Pi.

74 DesignSpark Tipps & Tricks

76 Seltsame Bauteile (12)
Tetroden-Transistor

77 Der Programmer im Elektor-Labor

106 Intel Edison: What will you make?
Weniger als ein Jahr nach der Einführung des Galileo-Boards stellte Intel sein winziges Edison-Modul vor.

Projects

12 Von 8 auf 32 bit: ARM-Controller für Einsteiger (1)
Die M0+-Familie der ARM-Cortex-Controller ist leistungsfähig und preisgünstig; darüber hinaus gibt es kostenlose Tools, welche die Entwicklung stark vereinfachen. Was sollte dann einem Einstieg in die Welt der 32-bit-Controller im Wege stehen?

22 Platino-Funktionsgenerator
Dieser Funktionsgenerator eignet sich beispielsweise zur Signalverfolgung, als Taktgeber für digitale Schaltungen oder zum Testen von Audio-Elektronik.

30 CC2-eBoB
Unser Breakout-Board beherbergt den Feuchte/Temperatursensor ChipCap2. Das kleine Platinchen führt alle Anschlüsse des Chips auf eine 0,1“-Stiftleiste.

36 Drahtloses Thermometer mit Bluetooth
Dieses Außenthermometer besitzt ein wasserdichtes Gehäuse und kommuniziert über Bluetooth Low Energy 4.0 mit einem aktuellen Smartphone. Es nutzt das Bluetooth-Modul BL600-SA von Laird Technologies, welches in BASIC programmiert werden kann.

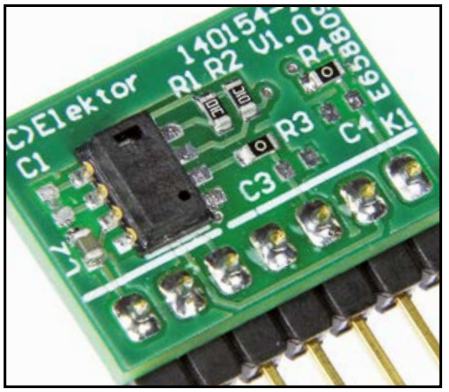
44 J²B-Synthesizer
Der Musik-Synthesizer Atmegatron von Soulsby Synthesizer ist mit einem ATmega328 aufgebaut. Clemens Valens aus dem Elektor-Labor hat das Projekt auf das J2B-ARM-Board von Elektor portiert und hier und da Verbesserungen eingebracht.

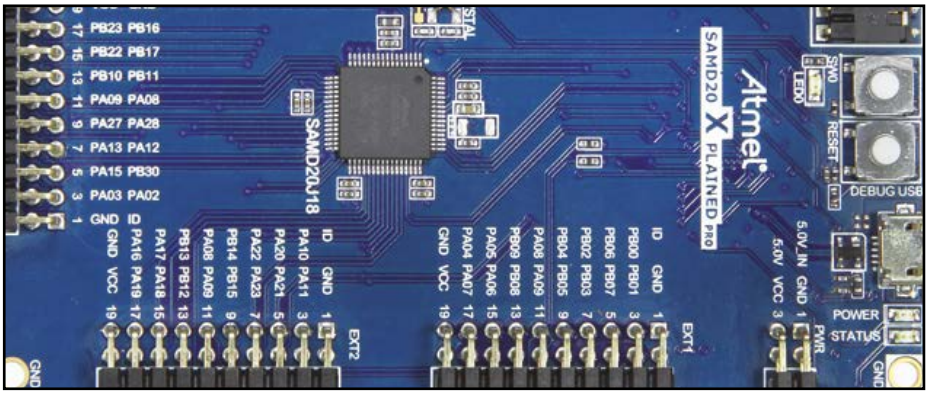
56 FM-Synchro-Sender

56 FM-Synchro-Empfänger

58 LED-Treiber für Glühlämpchen

61 Signalverstärker für USB-Oszilloskop





62 T-Board Wireless
Die drahtlose Datenübertragung hat auch auf der Ebene der Mikrocontroller-Systeme Einzug gehalten. Für Einzelanwendungen und Versuchsaufbauten, zum Beispiel auf Steckplatinen, haben wir eine Trägerplatine entworfen, die den Einsatz von Drahtlos-Modulen vereinfacht.

66 Beep
Ein Logiktester mit Pfiff!

69 Tristate-Level-Shifter

78 Abschaltverzögerung für Mikrocontroller

82 Entwickeln mit dem IoT-Kit WiSmart

84 VariLab 402 (3)
Im dritten und letzten Teil der Serie geht es um die Software unseres Labor-Netzgerätes. Wie wurde sie geschrieben und wie ist sie strukturiert? Am Schluss stehen der elektrische und mechanische Aufbau.

92 USBprog 5.0
Der Open-Source-Programmer USBprog erfreute sich bei Elektor-Lesern großer Beliebtheit, denn er war als „Multi-Tool“ für die unterschiedlichsten Controller zu gebrauchen. Hier kommt eine neue Version, die sogar mit einem Webserver ausgestattet ist.

97 Präzise Ausgangspannung

98 Modernes Messwerk in altem Gewand

102 Glühlampenelektronik

110 IIR-Sinus-Generator
Sinusförmige Signale kann man digital ziemlich einfach erzeugen, ohne dass man dafür auf Tabellenwerte und aufwendige Interpolationen zurückgreifen müsste. Der nötige Code passt selbst in einen kleinen Mikrocontroller.

114 Richtig programmieren!

120 EveryCircuit
App für Elektronik-Enthusiasten

Magazine

6 Impressum

8 electronica Impressionen

10 Gesehen auf der electronica
Neuheiten aus der Elektronikwelt

123 Hexadoku
Sudoku für Elektroniker

124 Retronik
NF-Generator Philips GM2308 (1950, 1964)

130 Vorschau
Nächsten Monat in Elektor

Impressum

46. Jahrgang, Nr. 529/530 Januar/Februar 2015
Erscheinungsweise: 10 x jährlich
(inkl. Doppelhefte Januar/Februar und Juli/August)

Verlag
Elektor-Verlag GmbH
Süsterfeldstraße 25
52072 Aachen
Tel. 02 41/88 909-0
Fax 02 41/88 909-77

Technische Fragen bitten wir per E-Mail an redaktion@elektor.de zu richten.

Hauptsitz des Verlags
Elektor International Media
Allee 1, NL-6141 AV Limbricht

Anzeigen:
Margriet Debeij (verantwortlich)
Tel. 02 41/88 909-13 / Fax 02 41/88 909-77
Mobil: +31 6 510 530 39
E-Mail: margriet.debeij@eimworld.com

Julia Grotenrath
Tel. 02 41/88 909-16 / Fax 02 41/88 909-77
E-Mail: julia.grotenrath@eimworld.com

Es gilt die Anzeigenpreisliste Nr. 45 ab 01.01.2015

Distribution:
IPS Pressevertrieb GmbH
Postfach 12 11, 53334 Meckenheim
Tel. 0 22 25/88 01-0 | Fax 0 22 25/88 01-199
E-Mail: elektor@ips-pressevertrieb.de

Druck: Senefelder Misset, Doetinchem (NL)
ISSN 0932-5468

Enter the next level!

Was haben ein Musik-Projekt, ein Mikrocontrollerkurs und ein Review eines neuen Programmiergeräts gemeinsam? In all den genannten Elektor-Artikeln in dieser Ausgabe steht ein ARM-Controller im Zentrum des Geschehens. Vielleicht geht es Ihnen wie mir? Von den Prozessor(kern)en, die als „Intellectual Property“ an fast alle namhaften Controllerhersteller verkauft werden, wusste ich bis vor kurzem nicht viel mehr, als dass sie auf einem beispiellosen Siegeszug sind. Neben Smartphones und Tablets wird inzwischen sogar der Servermarkt adressiert, traditionell die Domäne von Intel und Co. Am anderen Ende der Skala befinden sich die ARM Cortex-M-Controller, die auf typische Embedded-Anwendungen zielen. Die kleinen 32-bit-Controller sind leistungsstark, vielseitig ausgestattet und ähnlich preiswert wie 8-bit-Controller. Hier wird es für unsere Zeitschrift interessant, die traditionell von Profis (etwa Entwicklern kleinerer Ingenieurbüros) genauso gelesen wird wie von Studenten sowie engagierten Hobbyisten. Wer von diesen Lesern bereits (wie ich) Erfahrung in der Programmierung von 8-bit-Controllern gesammelt hat, wird (wie ich) darauf brennen, nun auch einmal einen 32-bit-Controller zu zähmen. Das Tor in diese Welt wollen wir mit unserem neuen Mikrocontrollerkurs aufstoßen, für den wir uns den kleinsten ARM-Cortex-Controllertyp M0 herausgesucht haben. Unsere Wahl fiel auf einen Controller und ein Board von Atmel, da viele Leser die dazugehörende, kostenlose Entwicklungsumgebung „Atmel Studio 6“ schon kennen dürften. Ein (ebenfalls kostenloses) Bibliotheks-Framework des Herstellers macht es auch ARM-Einsteigern wie mir einfach, zu ersten Erfolgserlebnissen zu kommen.

Machen Sie mit!

Jens Nickel
Chefredakteur Elektor

Unser Team

Chefredakteur:	Jens Nickel (v.i.S.d.P.) (redaktion@elektor.de)
Ständige Mitarbeiter:	Dr. Thomas Scherer, Rolf Gerstendorf
Leserservice:	Ralf Schmiedel
Korrekturen:	Malte Fischer
Internationale Redaktion:	Harry Baggen, Jan Buiting, Denis Meyer
Elektor-Labor:	Thijs Beckers, Ton Giesberts, Luc Lemmens, Jan Visser, Clemens Valens
Grafik & Layout:	Giel Dols

Germany
Ferdinand te Walvaart
+49 241 88 909-17
f.tewalvaart@elektor.de

United Kingdom
Carlo van Nistelrooy
+44 20 7692 8344
c.vannistelrooy@elektor.com

Netherlands
Ferdinand te Walvaart
+31 46 43 89 444
f.tewalvaart@elektor.nl

France
Denis Meyer
+31 46 4389435
d.meyer@elektor.fr

USA
Carlo van Nistelrooy
+1 860-289-0800
c.vannistelrooy@elektor.com

Spain
Jaime González-Arintero
+34 6 16 99 74 86
j.glez.arintero@elektor.es

Italy
Maurizio del Corso
+39 2.66504755
m.delcorso@inware.it

Sweden
Carlo van Nistelrooy
+31 46 43 89 418
c.vannistelrooy@elektor.com

Brazil
João Martins
+31 46 4389444
j.martins@elektor.com

Portugal
João Martins
+31 46 4389444
j.martins@elektor.com

India
Sunil D. Malekar
+91 9833168815
ts@elektor.in

Russia
Nataliya Melnikova
+7 (965) 395 33 36
Elektor.Russia@gmail.com

Turkey
Zeynep Köksal
+90 532 277 48 26
zkoksal@beti.com.tr

South Africa
Johan Dijk
+31 6 1589 4245
j.dijk@elektor.com

China
Cees Baay
+86 21 6445 2811
CeesBaay@gmail.com

Unser Netzwerk

elektor · VOICE COIL · circuit cellar · hifi.nl MUZIEK - FILM - LIFESTYLE · audioXPRESS · techthe future

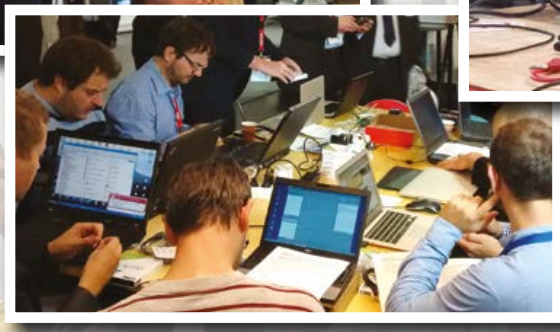
Die Elektor-Community

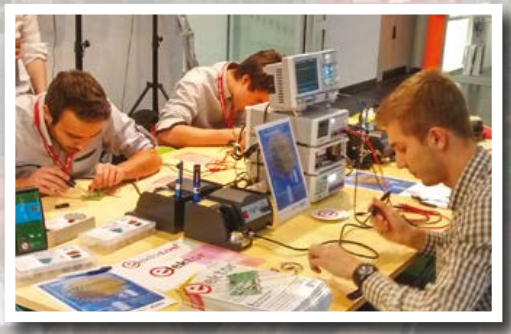
Unsere Partner und Sponsoren

Sie möchten Partner werden?

Kontaktieren Sie uns bitte unter m.debeij@elektor.de (Tel. 02 41/88 909-13).

electronica Impressionen





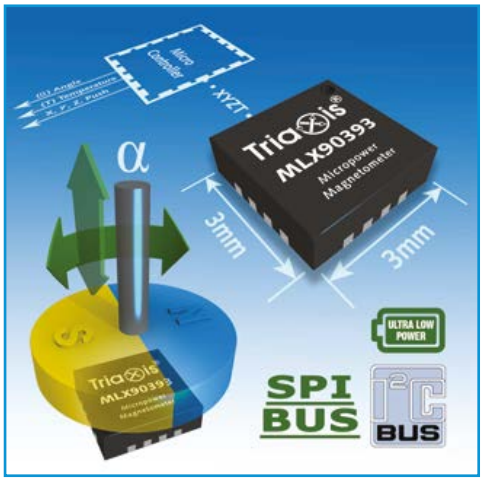
Mitte November fand in München die weltgrößte Elektronikmesse statt – die „electronica“ feierte dieses Mal 50-jähriges Jubiläum. Natürlich war auch Elektor vor Ort, mit einem Stand, der eine Nummer größer war als sonst. Namhafte Firmen wie Rohde & Schwarz, National Instruments und Conrad hatten unseren „Maker Space“ mit modernen Oszilloskopen, Netzgeräten, Lötstationen und vielen weiteren Tools ausgestattet. Auch die Münchner Messeleitung gab nach etwas Bedenkzeit (wegen der heißen Lötkolben) ihr OK, und so konnte man an unserem Stand vier Tage lang ganz praktisch zu Werke zu gehen. Unseren Kaffee und natürlich die bekannte Goodie-Bag nahmen die Besucher ebenfalls gut an. Einzig das kostenlose WLAN funktionierte wegen technischer Probleme nicht ganz durchgängig – wir hoffen, dass wir in zwei Jahren Gelegenheit bekommen, alles nochmals zu optimieren!

Eines der Highlights an unserem Stand waren sicher die kostenlosen Seminare. Von Retro-Elektronik bis zur modernen Messwert-Erfassung reichte die Palette; am Mittwoch abend hatten

die Besucher sogar Gelegenheit, an einem live ausgesandten Webinar zum Thema Oszilloskope und Stromversorgungen teilzunehmen. Die Firma iFixit gab seltene Einblicke in Tablet-Computer und zeigte, wie solche Alltags-Elektronik mit clever gemachten Werkzeugen zu reparieren ist. Am Donnerstag gab es gar ein Doppelseminar zum Thema Parallelprogrammierung mit XMOS-Prozessoren. Mehr als ein Dutzend Software-Enthusiasten ließen sich die Chance nicht entgehen, hier ihr Wissen zu erweitern – mit einer theoretischen Einführung und praktischen Programmier-Übungen.

Am letzten Messetag, dem „Student Day“ kamen besonders viele jüngere Elektroniker an den Stand. Das Studium der angehenden Ingenieure ist ja häufig etwas theorielastig – da nutzte man gerne die Gelegenheit, mit den Jungs aus dem Elektor-Labor zu fachsimpeln oder sich im Löten zu üben. Doch auch den Kolleginnen Chantalle und Julia hat es viel Spaß gemacht, eines unserer LED-Dreiecke aufzubauen!

(140466)



Magnetometer-IC

Melexis stellte in München einen programmierbaren, kompakten Sensor-Chip vor, der den Magnetfluss in x-, y- und z-Richtung genau misst. Der MLX90393 ist laut Hersteller der erste 3D-Magnetsensor, der eine lineare Empfindlichkeit über einen vollen, wählbaren 16-bit-Bereich von ±5 bis ±50 mT bietet. Damit eignet er sich vor allem für die Erfassung der Neigung und/oder Position eines kleinen Magneten, womit sich fast unbegrenzte Anwendungs-Möglichkeiten ergeben: vom Joystick, Schiebe-/Druck-/Zug-/Drehschalter, der Messung und Überwachung einer Neigung bis hin zu komplexen 3D-Positionserfassungssystemen. Der Sensor eignet sich für stromsparende Anwendungen, denn er verbraucht nur 2,5 µA Strom im Leerlauf. Über seine SPI- und I²C-Schnittstellen haben Entwickler Zugriff auf die verschiedenen Betriebsmodi des MLX90393: kontinuierliche Messung, Einzelmessung, Aktivierung bei Ereignissen und Burst-Modus. Die Einstellungen lassen sich während des Betriebs je nach Applikationsanforderung ändern.

www.melexis.com

Elektronik auf Glas gedruckt

Die Turck Duotec GmbH, Spezialist in der Entwicklung und Fertigung kundenspezifischer Elektronik, zeigte in München auf Glas gedruckte Schaltungen. Die Leiterbahnen gelangen per Siebdruck auf das Glassubstrat. Es lassen sich verschiedene Farbschichten auf das Glas aufbringen, die nachher um den funktionalen Druck von Leiterbahnstrukturen und elektronischen Schaltungen ergänzt werden.

Dies eröffnet zahlreiche Anwendungsmöglichkeiten, wie etwa hinterleuchtete Schalter und Taster mit kapazitiver Touch-Funktion, Bedieneinheiten und Eingabepanels. Leucht-, Anzeige- sowie Bedienfunktionen sind in ein einziges Element integrierbar, das dann zusammen mit der Glasfläche eine stabile, platzsparende, leicht zu reinigende sowie optisch hochwertige Einheit bildet.

Auf der electronica zeigte Turck Duotec als praktische Anwendung eine kapazitiv arbeitende Bedieneinheit, die per Funk mit einer LED-Steuerung kommuniziert.

www.turck-duotec.com

Elektor auf der Embedded World

Schon naht das nächste Messe-Highlight: Vom 24. bis 26. Februar 2015 findet in Nürnberg die Messe Embedded World statt. Die „Embedded" ist die größte Ausstellung ihrer Art – ein Muss für alle, die mit der Entwicklung, dem Einkauf und der Anwendung von Mikrocontrollertechnologie betreut sind. Es werden mehr als 650 Aussteller erwartet, zu den Themenbereichen Controller, Peripherie, Tools und Software. Hier darf Elektor natürlich nicht fehlen! An unserem Stand werden die neuesten Embedded-Projekte aus unserer Zeitschrift gezeigt. Bei einer guten Tasse Kaffee können Sie aber auch einfach nur mit Redakteuren und Entwicklern aus dem Elektor-Labor fachsimpeln. Außerdem wird Elektor für die Messe ein Business Special „Embedded" mit interessanten Artikeln aus der Welt der Mikrocontroller herausgeben, eine Gratis-Ausgabe gibt es ebenfalls bei uns am Stand. Für unsere Abo-Mitglieder halten wir wie immer ein kostenloses Überraschungspaket bereit.

Elektor-Stand: Halle 5, Stand-Nummer 288.

www.embedded-world.de

Bluetooth-Münzen

EM Microelectronic, ein Low-power-Halbleiterhersteller und zur Swatch-Gruppe gehörend, zeigte kleine Bluetooth-Sender in einem münzenähnlichen Gehäuse. Die wetterfest gekapselten „COiNs“ übermitteln in erster Linie eine eigene ID nach dem iBeacon™-Standard. Mehrere Beacons bekannter Position lassen sich somit zur Indoor-Navigation einsetzen; beim Annähern eines Smartphones können dort auch automatisch Aktionen ausgelöst werden (Museen, Galerien, Marketing). Über die integrierte Leiterbahnantenne sind veritable Reichweiten möglich. Bei 0 dBm Ausgangsleistung kann das Modul in 75 Meter Entfernung von einem iPhone 5S detektiert werden, bei maximaler

Ausgangsleistung sind es sogar 120 Meter. Das COiN enthält bei Auslieferung bereits eine CR2032-Knopfzelle von Renata; bei einem durchschnittlichen Stromverbrauch von weniger als 20 µA sind mehr als 18 Monate Betrieb möglich. Die COiNs sind überdies vorprogrammiert und lassen sich so einfach in eigenen Anwendungen einsetzen. Auf der untersten Ebene kann man die Firmware leicht modifizieren, um die UUID, MAJOR ID, MINOR ID, Ausgangsleistung und das Beacon-Intervall zu ändern. Sollten umfangreichere Firmware-Änderungen gewünscht werden, bietet EM ein komplettes Entwicklungspaket. Das COiN Development Kit enthält fünf COiN-Beacons, ein Programmierboard und Programmierkabel. Die COiNs lassen sich dann beispielsweise zu drahtlosen Sensor-Knoten umprogrammieren, um Daten wie Temperatur, Lichtstärke, Batteriespannung oder andere physikalische Größen übertragen.

www.emdeveloper.com/?page_id=369

Integriertes Touch-Display

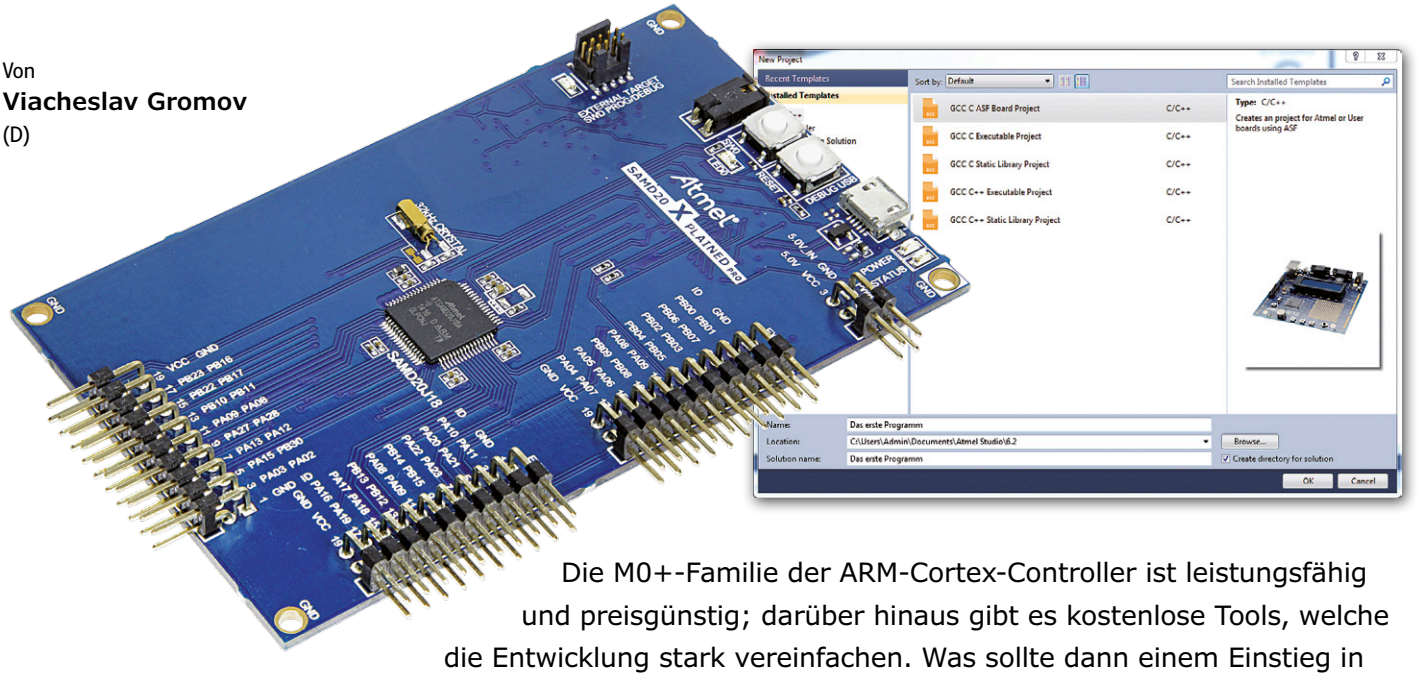
Advanced Display Application Module (kurz: A.D.A.M.) nennt Glyn seine integrierte Displaylösung. Der Distributor kombiniert dabei ein 4,3“- bzw. 3,5“-Display mit kapazitiver Touch-Funktion und einen Grafik-Chip EVE FT801 von FTDI. Über EVE (Embedded Video Engine) haben wir schon berichtet: Der intelligente Grafiktreiber nimmt direkt Kommandos zum Zeichnen von Grafiken entgegen, die ihm über SPI oder I²C von einem Hostcontroller übermittelt werden. Die Auswertung des Touchpanels übernimmt der EVE-Chip ebenfalls. Glyn liefert darüber hinaus noch eine Display-Library für Hostcontroller mit.

Von 8 auf 32 bit:

ARM-Controller für Einsteiger (1)

Das Board, die Software und ein erstes Programm

Von
Viacheslav Gromov
(D)



Die M0+-Familie der ARM-Cortex-Controller ist leistungsfähig und preisgünstig; darüber hinaus gibt es kostenlose Tools, welche die Entwicklung stark vereinfachen. Was sollte dann einem Einstieg in die Welt der 32-bit-Controller im Wege stehen? Unser Kurs, bei dem Erfahrungen mit 8-bit-Mikrocontrollern vorausgesetzt werden, hilft Ihnen dabei!

Der Programmierkurs soll Ihnen die Welt der ARM-Cortex-M0+-Controller näherbringen, wie immer bei Elektor legen wir dabei den Schwerpunkt auf die Praxis. Es gibt mittlerweile viele kostenlose Entwicklungsumgebungen und günstige Entwicklungsboards. Unser Kurs basiert auf dem Board „SAM D20 Xplained Pro“ mit einem Low-Power-Mikrocontroller SAM D20.

Dank Unterstützung des Controller-Hersteller Atmel können wir interessierten Elektor-Lesern 1000 Boards zu einem reduzierten Preis anbieten (siehe Infokasten).

In dieser ersten Kursfolge geht es los mit einem kleinen Überblick über Board und Mikrocontroller, danach wird die Entwicklungsumgebung Atmel

Studio 6.2 installiert und – damit man schon ein kleines Erfolgserlebnis hat - das erste Projekt erstellt. Wir ziehen manchmal auch Vergleiche mit der verbreiteten 8-bit-AVR-Familie, die in mancher Hinsicht ähnlich ist. In den nächsten Folgen werden dann die wichtigsten Peripherieelemente mit ihren Möglichkeiten in kleinen Projekten vorgestellt.

Das Board

Das Blockdiagramm des Boards (**Bild 1**) sieht auf den ersten Blick etwas unspektakulär aus. Man sieht, dass zahlreiche Pins des 64-Pin-Mikrocontrollers auf Header nach außen geführt sind. Die Belegung der Stiftleisten und anderer Anschlüsse ist in Tabellenform wiedergegeben.
Das Board kann wahlweise über die *USB*-Buchse oder den *PWR*-Header mit +5 V versorgt werden. Bei Versorgung über USB stehen an *PWR* 5 V beziehungsweise 3,3 V zur Versorgung von angeschlossenen Schaltungen zur Verfügung. Wenn das Board von *PWR* aus gespeist wird, schaltet sich der On-Board-Debugger EDBG (siehe Kasten) automatisch aus, um Strom zu sparen. Trotzdem ist es empfehlenswert, dass die Stromversorgung (oder der USB-Anschluss) mindestens 500 mA zu Verfügung stellen kann.
Auch an den Stiftleisten *EXT1* bis *EXT3* liegt eine Spannung von 3,3 V zur Versorgung von Erweiterungsboards. Auf jedem Extension-Header ist der Pin 1 namens ID für den Anschluss eines ID-Chips auf dem angeschlossenen Erweiterungsboard reserviert. Der EDBG erkennt dadurch, um was für ein Erweiterungsboard es sich handelt. Diese Information wird dann an die Entwicklungsumgebung auf dem PC weitergegeben.
Außer dem 32-kHz-Quarz (eine der Taktquellen des Hauptcontrollers) und dem Anschluss *DEBUG USB* für einen externen Debugger ist noch ein *RESET*-Taster, der Taster *SW0* und eine gelbe *LED0* auf dem Board vorhanden. *SW0* und *LED0* sind an PA15 respektive PA14 angeschlossen und stehen dem Anwender zur freien Verfügung. Der Jumper neben *SW0* verbindet die Ausgangsspannung vom Spannungsregler auf dem Board mit dem Mikrocontroller. Schließt man statt des Jumpers ein Messgerät an, kann man den Stromverbrauch des Mikrocontrollers messen.
Die (nicht eingezeichnete) Power-LED und die Status-LED neben der USB-Buchse sind beide an den EDBG angeschlossen. Die Power-LED leuchtet, wenn das Board mit Strom versorgt wird, die Status-LED blinkt, wenn der Hauptcontroller SAM D20 gerade durch den EDBG beim Debuggen oder sonstigen Vorgängen angesprochen wird. Wenn die Firmware des Debuggers aktualisiert wird, blinken beide LEDs gleichzeitig. Das User Manual zu diesem Board finden Sie unter [1].

Der Mikrocontroller

Der SAM D20J18 ist ein interessantes Mitglied der ARM-Cortex-M0+-Familie. Er besitzt 64 Pins, 256 kB Flashspeicher, 32 kB SRAM und wird mit einer maximalen Frequenz von 48 MHz getaktet. Er ist stromsparend, schnell, besitzt viel Peripherie und ist daher sehr universell einsetzbar. Er verbraucht nur 70 µA / MHz bei 1,62…3,63 V. Besonderheiten sind das *Event-System* und der

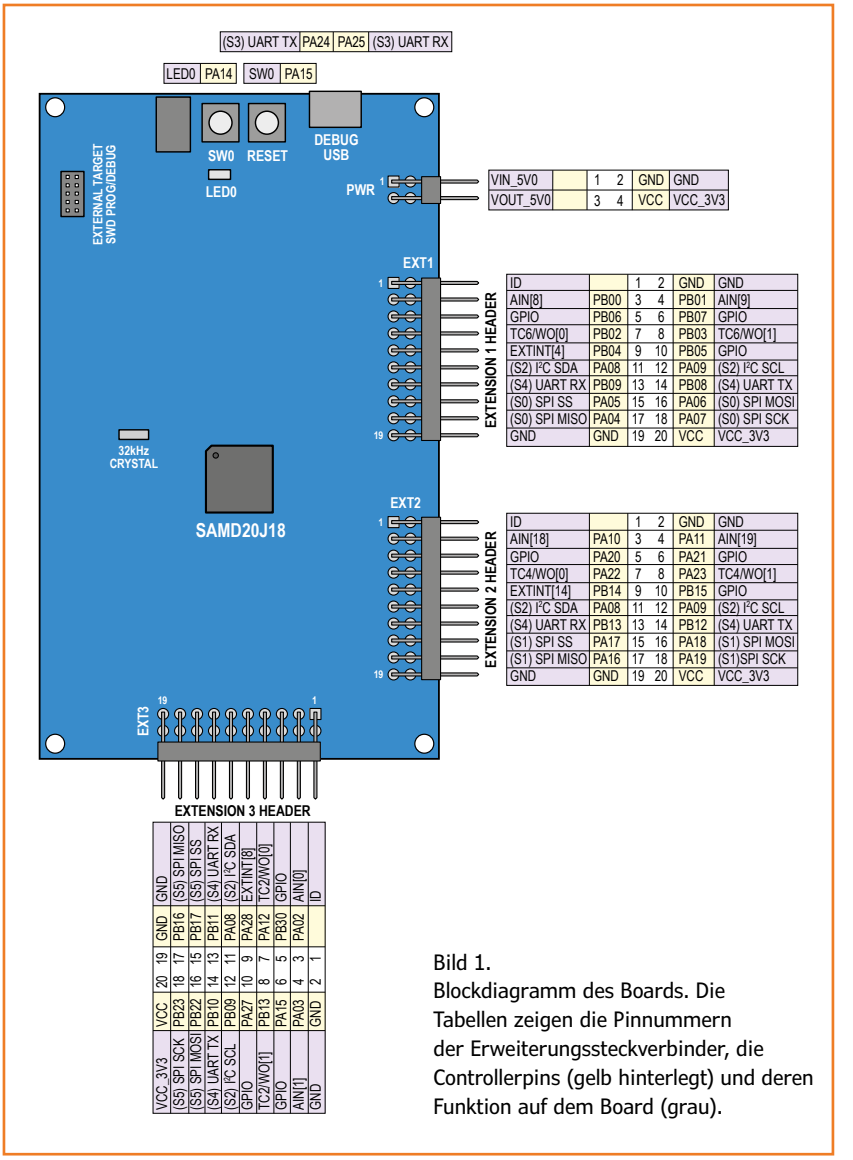


Bild 1.
Blockdiagramm des Boards. Die Tabellen zeigen die Pinnummern der Erweiterungssteckverbinder, die Controllerpins (gelb hinterlegt) und deren Funktion auf dem Board (grau).

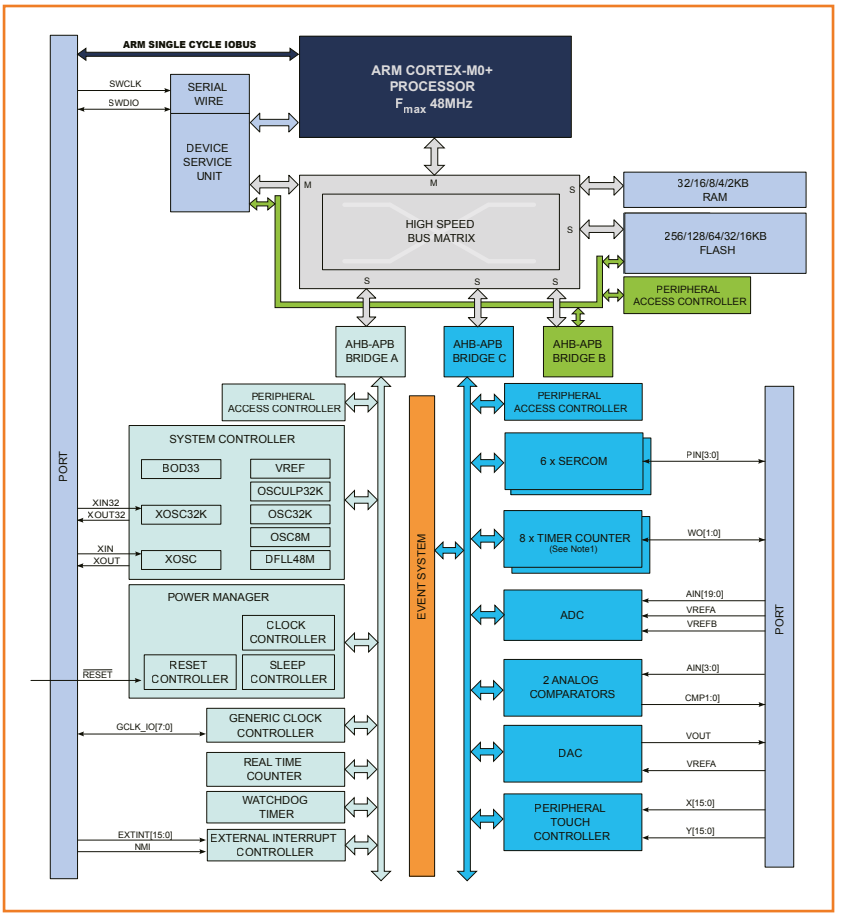


Bild 2.
Blockschaltbild der SAM-D20-Controller (Quelle: Atmel).

Peripheral Touch Controller (PTC). Der PTC wird in einer der späteren Kursfolgen genauer erläutert und in Betrieb gesetzt. Das *Event-System* kann, ähnlich wie bei den ATxmegas, je nach Konfiguration zum Beispiel die CPU aus dem Schlaf wecken, wenn ein Peripherieelement wie der ADC ein Event auslöst (allerdings ist nicht jedes Peripherieelement dazu in der Lage). Der Mikrocontroller kennt zwei Schlafmodi: Im Idle-Mode ist nur die CPU ausgeschaltet, während im Standby-Mode die ganze Taktversorgung und alle Elemente (außer denen, die im Programm anders konfiguriert wurden) in den Schlaf verfallen.

In **Bild 2** ist ein Blockschaltbild dieser Mikrocontrollerfamilie dargestellt. Auf der linken Seite erkennt man ganz oben den *ARM Single Cycle I/O*-Bus, der dem Prozessor einen ganz schnellen Zugriff auf die GPIOs gewährt. Darunter ist die serielle *Debug*-Schnittstelle mit direktem Zugriff auf den Prozessor zu sehen. Unter der *High Speed Matrix*, an die rechts die Speicher mit dem entsprechendem Controller als Slaves angeschlossen sind, erkennt man (im Gegensatz zu einem üblichen 8-bit-Mikrocontroller) mehrere Datenbusse und *Peripheral Access Controller*, die bei Bedarf das Beschreiben von Peripherie-Registern blockieren können. Am Bus *APB-C* liegt die wichtigste Peripherie, das interessanteste davon sind sicher die sechs *SERCOM*-Blöcke, die der seriellen Kommunikation dienen und als serielle Schnittstellen (USART, I²C, SPI) an verschiedenen Pins konfiguriert werden können. Die restlichen Peripherieelemente (abgesehen vom PTC) sind auch bei den meisten 8-bit-Mikrocontrollern vorhanden, allerdings mit weniger Leistung und in geringerer Anzahl. Die acht *TIMER COUNTER* können übrigens jeweils als 2x8-bit-Counter, 1x16-bit-Counter oder (zwei zusammen) als 32-bit-Counter eingestellt werden. Die andere Seite des Blockschaltbildes ist weniger spektakulär, es ist hauptsächlich nur die Strom- und Taktversorgung zu sehen. Auf die Möglichkeiten, Einstellungen und Anwendungen der genannten Peripherieelemente werden wir in den nächsten Folgen Schritt für Schritt praxisnah angehen. Das Datenblatt mit circa 700 Seiten finden Sie unter [3].

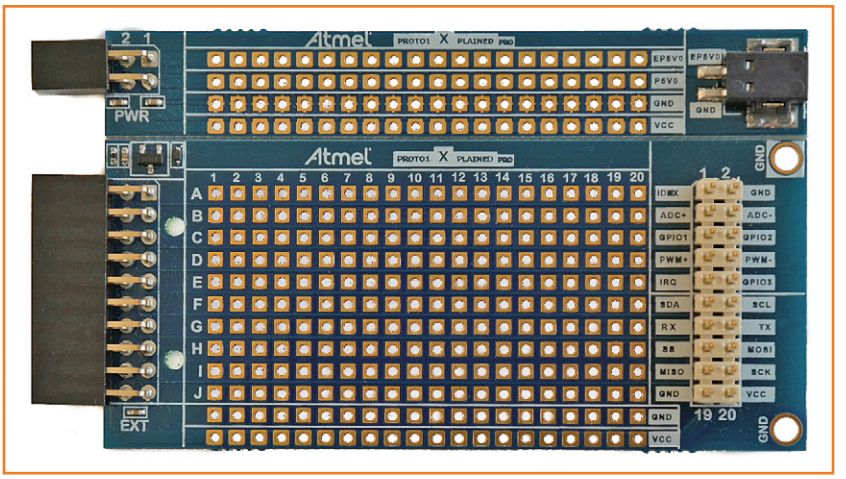


Bild 3.
Das einfachste bei Atmel erhältliche Erweiterungsboard ist ein Prototyping-Board mit Lötinseln.

Die Erweiterungsboards

Atmel hat für das *Xplained Pro-Board* einige Erweiterungs-Boards entwickelt, die beim Einstieg

und in der Prototyping-Phase helfen sollen, schnell zum Erfolg zu kommen und den Mikrocontroller besser kennen zu lernen. Die Erweiterungsboards werden praktischerweise direkt auf die Stiftleisten der Hauptplatine aufgesteckt. Jedes Erweiterungsboard besitzt einen dreibeinigen Crypto-Authentication-Chip ATSHA204, der dem EDBG-Chip auf dem *Xplained-Pro-Board* Informationen über das angeschlossene Zusatzboard mitteilt, zum Beispiel die Spannungsbereiche und den maximalen Strom für den Betrieb. Der EDBG-Chip gibt diese Daten dann an das Atmel Studio weiter. Das Studio öffnet ein Fenster mit Links zu Datenblättern, Bibliotheken oder Beispielprogrammen.

Wer einfach seine selbstgebaute Schaltung an ein Xplained-Pro-Board anschließen möchte, ist mit dem **PROTO1 Xplained Pro** [4] (**Bild 3**) bestens bedient. Es besitzt insgesamt 200 Lötpads und wird auf die Stiftleisten EXT1 und PWR gesteckt. An der rechten Seite gibt es einen Anschluss für die so genannten Xplained-Top-Module, der nur anders geroutet ist. Der obere Platinenteil mit der Spannungsversorgung kann an einer Sollbruchstelle leicht abgetrennt werden.

Mit dem Erweiterungsboard in **Bild 4**, dem **IO1 Xplained Pro** [5] lernen Sie die wichtigsten Peripherieelemente des Mikrocontrollers kennen. Es besitzt eine LED, einen Lichtsensor, einen Tiefpass zum Ausprobieren von PWM und ADC, einen 12-bit-Temperatursensor mit 8-kB-EEPROM und Anschluss an den I²C-Bus sowie einen MicroSD-Karteneinschub, der am SPI angeschlossen ist. Die MicroSD-Karte wird mitgeliefert. Außerdem finden sich noch ein paar herausgeführte Pins.

Wer Daten auf einem Display anzeigen möchte, benötigt das Erweiterungsboard **OLED1 Xplained Pro** [6], das Sie in **Bild 5** sehen, mit einem SPI-OLED-Display (128 x 32 Pixel). Außerdem stellt das Board noch drei LEDs und drei Taster zur Verfügung. Dieses Erweiterungsboard ist für den Anschluss an EXT3 gedacht.

Da der SAM D20 (wie auch einige andere Controller des Herstellers) als Besonderheit einen PTC

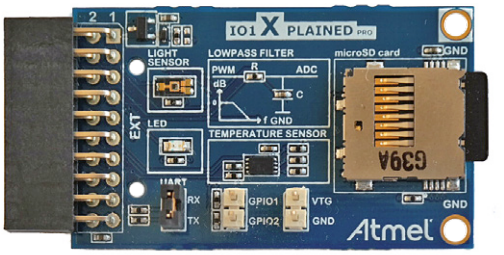


Bild 4.
Universelles Erweiterungsboard für Test- und Lernzwecke.

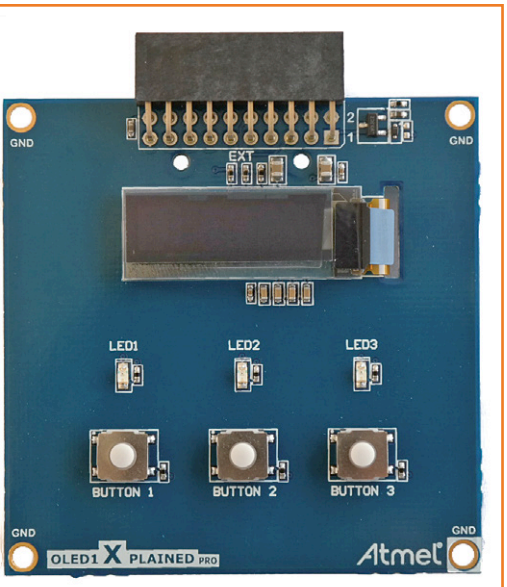


Bild 5.
OLED-Erweiterungsboard für fortgeschrittene Projekte.

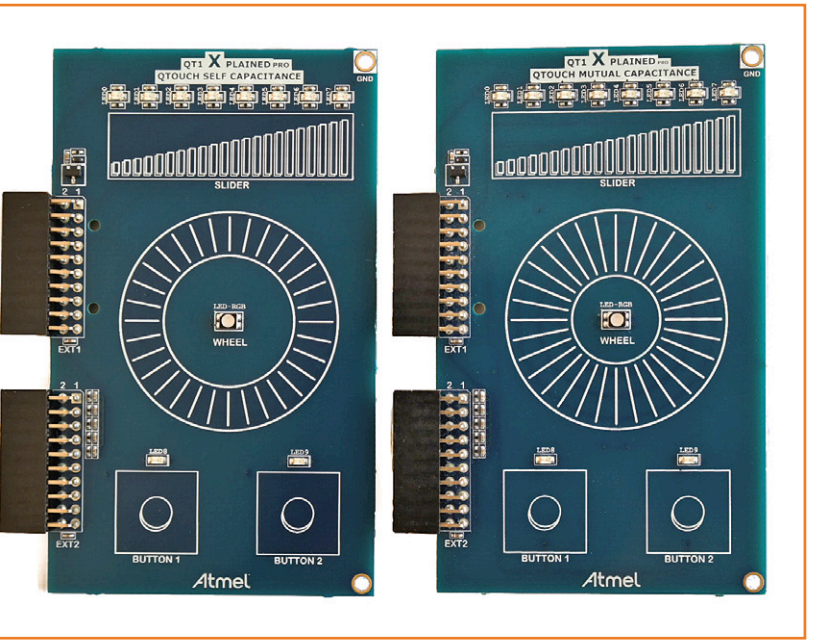


Bild 6.
Mit den beiden QTouch-Boards werden wir in einer späteren Kursfolge arbeiten.

Bild 7.
Sie sollten auf das obere Icon klicken, falls Sie Atmel Studio auf einem Computer ohne permanenten Internetzugang installieren wollen (alle Screenshots: Atmel).

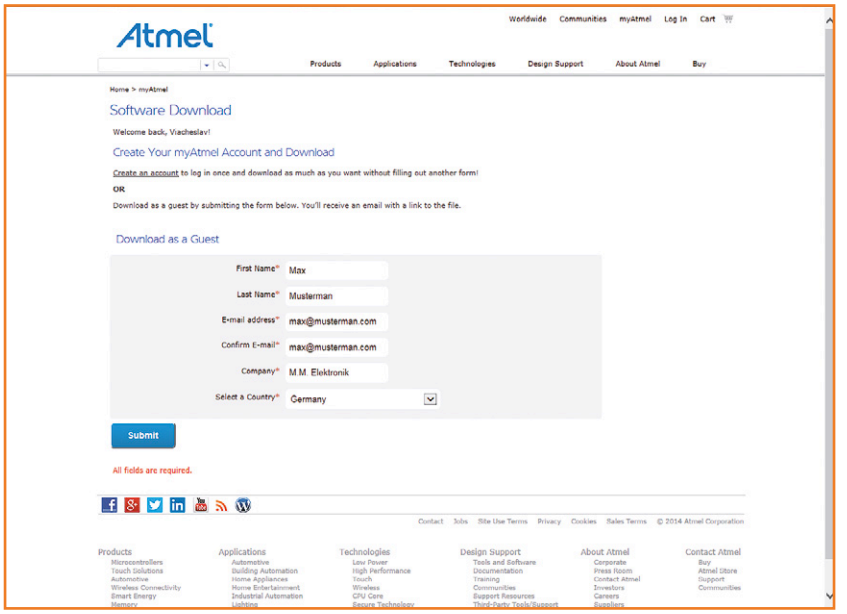


Bild 8.
Hier müssen Sie entweder auf „Create an account" klicken oder Ihre Daten eingeben.

besitzt, bietet Atmel ein Kit (**Bild 6**) namens **QT1 Xplained Pro** [7] mit gleich zwei Erweiterungsboards an. Vom Aussehen sind die Boards gleich, nutzen allerdings verschiedene Touch-Technologien: QTouch Self Capacitance und QTouch Mutual Capacitance. Die genauen Unterschiede und die Vor- und Nachteile dieser Touch-Technologien werden wir in einer der letzten Folgen kennenlernen. Auf jedem Board befinden sich neben den Touch-Elementen Rad, Slider und zwei Tastern insgesamt zehn gelbe und eine RGB-LED.

Die Entwicklungsumgebung

Einen Atmel-Mikrocontroller kann man grundsätzlich immer mit einem Programmer/Debugger und dem Atmel Studio programmieren und debuggen. Da diese IDE kostenlos ist, direkt aus den Händen des Herstellers kommt und viele wertvolle

Debugging und der EDBG

Vielleicht kennen Sie den Begriff Debugging noch nicht, da diese Technologie bei 8-bit-Mikrocontrollern wenig zum Einsatz kommt. Es ist eine Möglichkeit, Fehler in der Software (Bugs) zu finden und diese zu beheben.
Damit das Debugging funktioniert, sind zwei Komponenten nötig: Ein Tool in der Entwicklungsumgebung und ein Debugger, der die Brücke zwischen dem Computer und dem System (Mikrocontroller) darstellt. Man kann mit einem Debugger Variablen, Arbeitsspeicher und oft sogar Register während der Programmausführung einsehen und manipulieren. Es können dazu Breakpoints gesetzt werden, bei denen das Programm stoppt und Sie den gegenwärtigen Status der Hardware abrufen können. So können Fehler schnell und günstig gefunden werden, auch wenn der Controller schon im fertigen Prototyp eingebaut ist. Der Debugger muss dabei immer mit dem Mikrocontroller verbunden sein [9].
Der EDBG (**E**mbedded **D**e**b**u**g**ger) wird nicht nur bei allen Xplained-Pro-ARM-Boards von Atmel verbaut, sondern kommt auch oft bei AVR-Boards zum Einsatz. Es ist ein Debugger, der fest in die Boards integriert ist; Atmel hat ihn extra für Entwicklungskits entworfen. Außer der Möglichkeit, einen Mikrocontroller programmieren oder debuggen zu können, bietet er noch eine ganz besondere Funktion, die Ihnen später als sehr nützlich erscheinen dürfte - das Data Gateway Interface, das eine Brücke zwischen mehreren verschiedenen Schnittstellen beziehungsweise GPIOs und dem Computer darstellt. Man kann damit, während der Mikrocontroller in Funktion ist, Zustände von ausgewählten GPIOs sehen und über einige der Schnittstellen auch Daten empfangen, was die Entwicklung viel einfacher gestaltet.
Auf dem SAMD20-Xplained-Pro-Board ist dieses Interface mit den SPI-Pins von SERCOM5, den I²C-Pins des SERCOM2 und den GPIO-Pins PA27, PA28, PA20 und PA21 verbunden. Der Debug-Chip steuert auch die Status-LEDs und liest die IDs der Erweiterungsboards aus. Und schließlich kann der EDBG auch einen COM-Port über USB emulieren, da er an UART-Pins (SERCOM3) des SAM D20 angeschlossen ist [2].

Funktionen für Atmel-Produkte bietet, ist sie für den Einstieg geradezu ideal. Vielleicht kennen Sie das Atmel Studio ja auch schon.

Um die neueste Version 6.2 zu installieren, müssen Sie auf [8] gehen und auf das CD-Icon neben dem Eintrag *Atmel Studio 6.2 sp1 (build 1502) Installer* (**Bild 7**) klicken. Es erscheint ein Fenster wie im **Bild 8**, in dem Sie sich entweder registrieren oder als Gast anmelden müssen, damit Sie das Studio von der Atmel-Seite herunterladen können (eine Registrierung ist ratsam, da Sie sich auch im weiteren Verlauf dieses Kurses ab und zu anmelden müssen). Falls Sie schon ein Konto bei Atmel haben, müssen Sie sich rechts oben unter *Log In* einloggen. Nun wird ein Link eingeblendet, auf den Sie klicken müssen, um das Programm herunterzuladen. Wenn nun eine Meldung Ihres Browsers erscheint, in der Sie gefragt werden, ob das Programm lediglich heruntergeladen oder gleich ausgeführt werden soll, wählen Sie am besten *Ausführen* aus. Unter Windows wird nach einer Weile eine Sicherheitsmeldung erscheinen, bei der Sie am besten den Haken bei *Software von „Atmel Norway" immer vertrauen* setzen und schließlich auf *Installieren* drücken. Wenn auf Ihrem Computer noch kein Microsoft .NET Framework und/oder Visual Studio installiert ist, wird sich das Studio sofort melden und Sie zu einer Installation auffordern. Folgen Sie bitte den Anweisungen. Beim Installieren beider Programme werden Sie auch Lizenzen akzeptieren und auf jeden Fall immer die Full Version auswählen müssen. Den vorgeschlagenen Installationspfad können Sie übernehmen, sofern Sie keine Einwände haben. Danach sollte Ihnen der InstallShield-Wizard ein Fenster einblenden, das Sie zu einer Installation der USB-Treiber auffordert, was Sie mit *Install* bestätigen müssen (**Bild 9**).

Nach dem Bestätigen der Lizenz im nächsten Fenster (**Bild 10**) startet die Installation der Treiber, was ein paar Minuten dauern kann. Danach erscheint (hoffentlich) die Erfolgsmeldung, dass die Treiber erfolgreich installiert wurden. Dies sollten Sie auch bestätigen.

Nun beginnt die Installation des Atmel Studios. Beim ersten Fenster (**Bild 11**) sollten Sie auf *Next>* klicken, beim zweiten die Lizenz akzeptieren und das Fenster ebenfalls mit *Next>* verlassen (**Bild 12**). Im nachfolgenden Schritt (**Bild 13**)

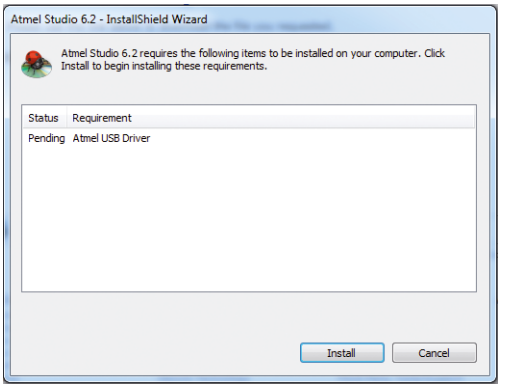


Bild 9.
Das Studio verlangt die Installation der USB-Treiber.

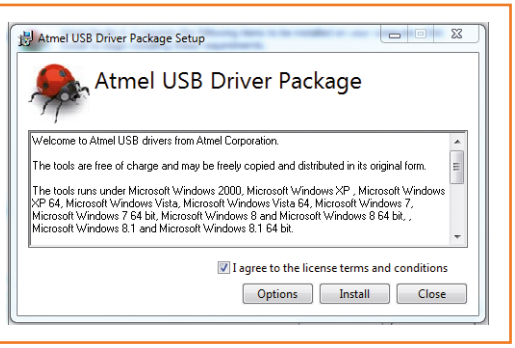


Bild 10.
Lesen Sie sich bitte die Lizenz gründlich durch, bevor Sie sie bestätigen.

Bild 11.
Die Hauptinstallation beginnt.

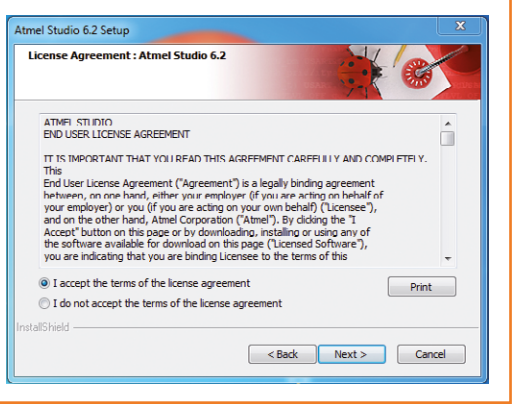


Bild 12.
Ohne die Lizenz zu akzeptieren, kommen Sie nicht weiter.

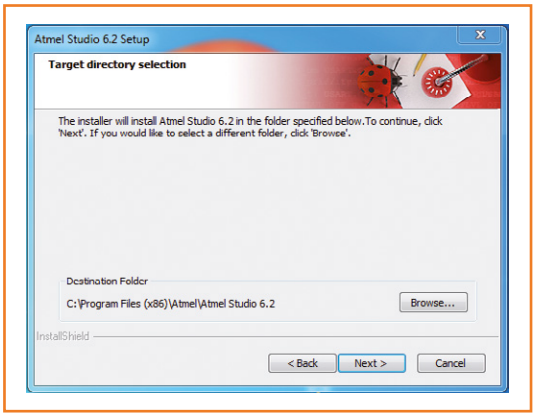


Bild 13.
Der vorgeschlagene Dateipfad ist meist gut gewählt.

Bild 14.
Sie haben die Installation hinter sich!

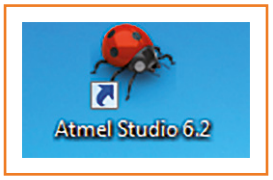


Bild 15.
Das Icon des Atmel Studio 6.2.

wählen Sie den Installationspfad für das Studio. Sie müssen auf *Browse…* drücken, falls Ihnen der vorgeschlagene Dateipfad nicht passt und das Fenster wieder mit *Next>* verlassen.
Der letzte Schritt vor dem Installationsvorgang ist eine Zusammenfassung von dem, was und wo installiert werden soll. Dieses Fenster wird ebenfalls mit *Next>* verlassen. Nun kann es einige Zeit dauern, bis die Installation vollendet ist (**Bild 14**).

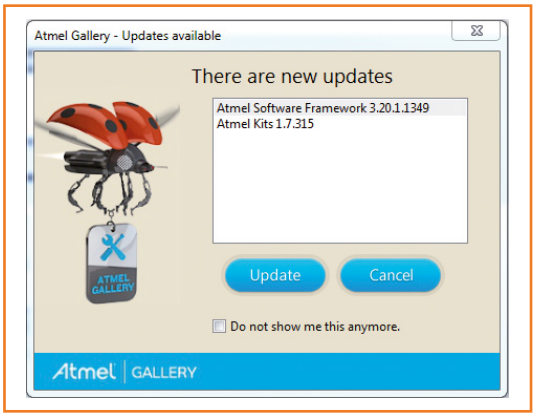


Bild 16.
Die Updates sollte man unbedingt durchführen.

In der angezeigten Checkbox können Sie noch einen Haken setzen (bevor sie auf Finish drücken), falls Sie keine andere Entwicklungsumgebung auf Ihrem Computer installiert haben, die die aufgelisteten Dateiarten auch benutzt. Der InstallShield-Wizard verschwindet und auf Ihrem Desktop erscheint ein Icon wie im **Bild 15**. Es ist empfehlenswert, jetzt einen Neustart durchzuführen, da sonst Probleme auftauchen könnten.

Mein erstes Programm

Das Entwicklungsboard ist mit einem „USB-A-Stecker auf USB-Micro-B-Stecker“-Kabel am PC angeschlossen. Der Gerätemanager meldet sich und installiert die Treiber, die Sie zusammen mit dem Studio geladen haben. Falls Ihr Gerätemanager keine Treibersoftware findet, müssen Sie den Pfad dem Manager manuell mitteilen.
Wir nehmen nun das Atmel Studio das erste Mal in Betrieb. Klicken Sie bitte doppelt auf das dazugehörige Icon auf dem Desktop. Nach kurzer Zeit sehen Sie schon das Begrüßungsfenster. Nun taucht bestimmt auch eine Meldung wie in **Bild 16** auf, die Ihnen ein Update mehrerer Tools anbietet.

Darunter ist auch das *Atmel Software Framework* (ASF), wovon Sie in Zukunft viel Gebrauch machen werden. Klicken Sie also bitte auf *Update*, damit Sie zum *Extension Manager* (**Bild 17**) weitergeleitet werden, in dem die nötigen Updates einzeln nacheinander durchgeführt werden.
Am Anfang werden Sie aufgefordert, sich anzumelden, damit Sie die Updates installieren können. Da sich die Installationsabläufe der Updates unterscheiden, können wir hier darauf nicht detailliert eingehen. Sie werden Sicherheitsmeldungen erhalten, Lizenzen akzeptieren, die Updates mit dem Studio verlinken, je nachdem. Das alles verläuft sehr stressfrei, folgen Sie einfach immer den Anweisungen des Installationsprogramms. Wenn Sie nun alle Updates hinter sich haben, können Sie den Extension Manager schließen und der Empfehlung des Managers folgen, das Studio neu zu starten.
Und jetzt können wir endlich mit dem ersten Projekt loslegen! Zunächst wollen wir die IDE kennenlernen. Nach dem erneuten Öffnen des Atmel Studios müssen Sie oben die Startseite auswählen und auf *New Project…* klicken, um ein neues Projekt zu erstellen (**Bild 18**). Alternativ können Sie auch auf *Project…* unter *Folder/New* klicken.

Über den Autor

Viacheslav Gromov ist mit 15 Jahren einer der jüngsten Elektor-Autoren, doch schon seit Jahren begeistert von analoger und digitaler Elektronik. Sein Hobby übt er meist in seinem gut ausgestatteten Hobbykeller aus. Er hat schon einige Bauteil-Tipps bei Elektor veröffentlicht. Darüber hinaus hat er Bücher geschrieben, die sich unter anderem mit ARM-Cortex-Controllern beschäftigen.

Er möchte sich an dieser Stelle nicht nur bei seiner Familie für die Unterstützung seines Hobbys, sondern auch bei Andreas Riedenauer von der Ineltek Mitte GmbH für die Unterstützung mit Informationen und Boards bedanken.

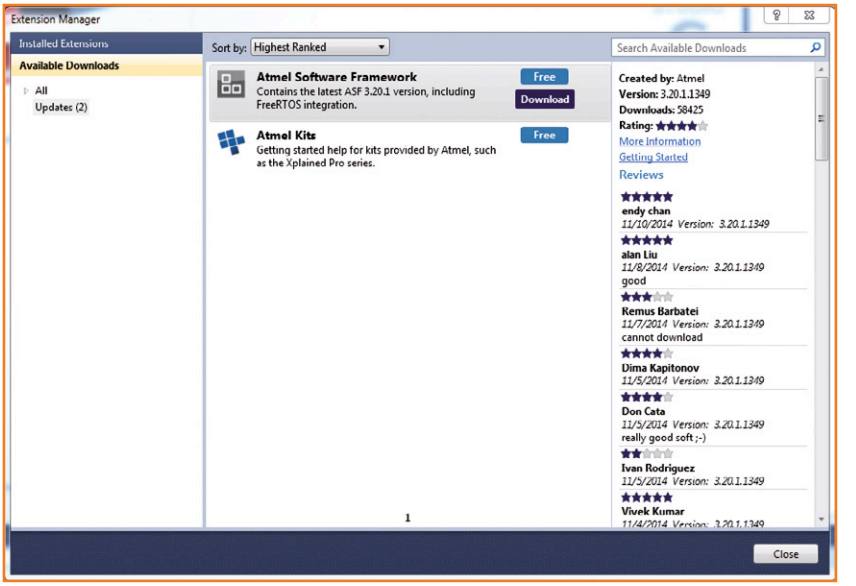


Bild 17.
Im Extension-Manager finden Sie viele zusätzliche Software-Tools, die Sie installieren können.

Jetzt erscheint im Studio das *New Project*-Fenster, das Sie nach der Art, dem Namen und dem Pfad des zu erstellenden Programms fragt. Wählen Sie bitte wie im **Bild 19** das *GCC C ASF Board Project*, das, wie der Name schon sagt, auf der Programmiersprache C und dem *Atmel Software Framework* beruht. Zum ASF kommen wir im nächsten Monat zurück. Im nächsten Fenster (**Bild 20**) wird das entsprechende Board mit Hilfe der Suchfunktion ausgewählt.

Nun wird es eine Weile dauern, bis das Projekt erstellt ist und Sie die Datei main.c im Ordner src öffnen können. Dort sehen Sie den Quellcode der generierten Mini-Anwendung (**Listing 1**). Am Anfang wird die Header-Datei des ASF eingebunden, und danach in der main-Routine das System initialisiert. Schließlich wird in der Endlosschleife mit einer if-Abfrage geschaut, ob der Taster SW0 betätigt ist und dann die LED0 entsprechend eingeschaltet. Unser Projekt ist also schon in dieser Beispieldatei realisiert und die Funktionen sind gut verständlich.

Nun kurz zu den anderen Dateien. Die Datei asf.h stellt nur eine „Umleitung“ zu anderen Header-Dateien dar. Das ist sehr nützlich, denn sonst müsste man alle nötigen Dateien einzeln in die main-Datei einbinden. Die Datei samd20_

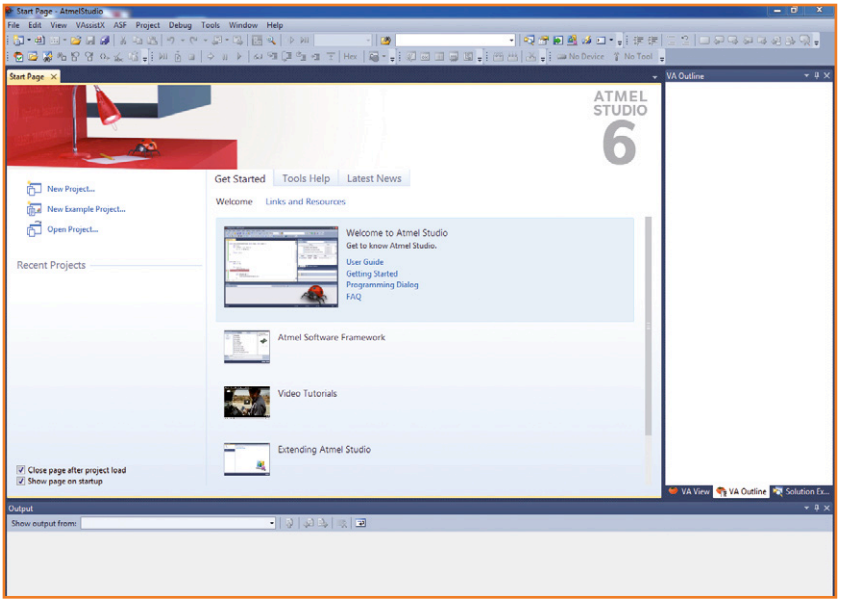


Bild 18.
So sieht die Startseite im Atmel Studio 6.2 aus.

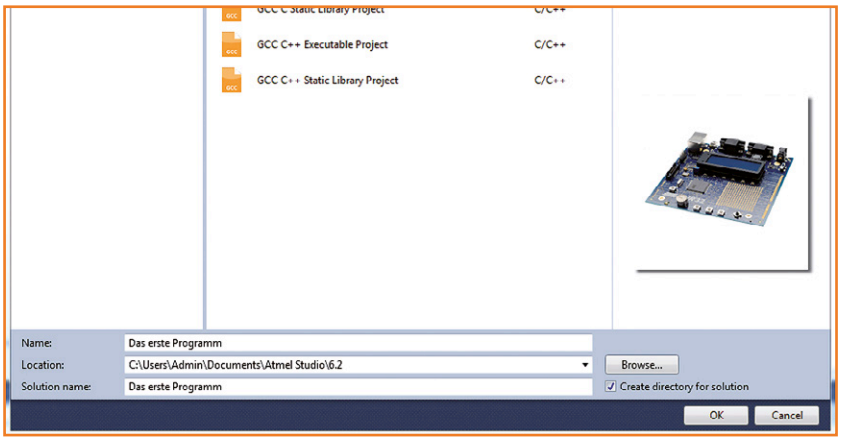


Bild 19.
Hier muss man seinem Projekt einen Namen geben.

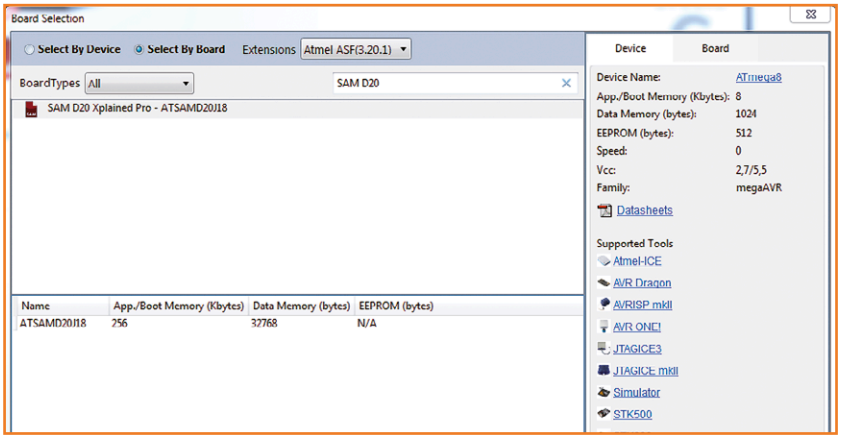


Bild 20.
Wählen Sie bitte vor der Suche „Select By Board“ aus.

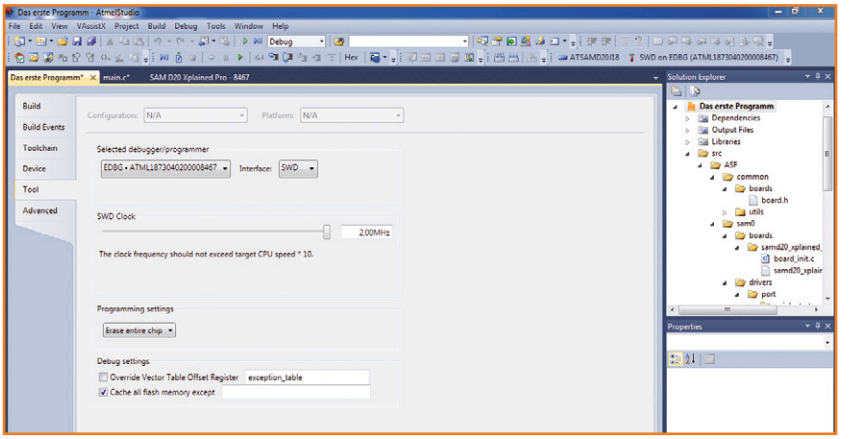


Bild 21.
Das Studio sollte den Debugger auf dem Board problemlos erkennen können.

xplained_pro.h ist unter src\ASF\sam0\boards\samd20_xplained_pro zu finden und definiert Namen für wichtige Pins auf dem Board (**Listing 2**). Diese Datei wurde vom Studio automatisch erstellt, da es sich um ein Board-Projekt handelt.

Weitere wichtige Dateien für dieses Projekt sind system.c, in der die Funktion system_init(); definiert ist und port.h, in der die GPIO-Funktionen deklariert sind. Schauen Sie sich das einfach einmal an.

In den nächsten Folgen werden wir noch näher auf die einzelnen Funktionen eingehen, aber jetzt wollen wir das Projekt kompilieren und auf das Board übertragen. Dazu müssen Sie auf den grünen Pfeil *Start Without Debugging* drücken, damit das Projekt lediglich kompiliert und übertragen, aber nicht debuggt wird. Wählen Sie bitte davor noch den Debugger oben rechts wie in **Bild 21** gezeigt aus. Die restlichen Einstellungen können so belassen werden. Nach dem Kompilieren kann noch eine Meldung wie in **Bild 22** erscheinen, die Ihnen vorschlägt, die Firmware des Debuggers aufzufrischen. Drücken Sie dann auf Upgrade und warten Sie, bis die Firmware auf dem Board ist. Dann drücken Sie nochmal auf den grünen Pfeil, so dass unser Programm nun endlich übertragen wird. Nun müssten Sie unten im Output-Fenster eine Erfolgsmeldung wie folgt erhalten (gekürzt):

```
Program Memory Usage : 1628 bytes  0,6 % Full
Data Memory Usage    : 8256 bytes 25,2 % Full

Build succeeded.
========== Build: 1 succeeded or up-to-
date, 0 failed, 0 skipped ==========
```

Unser erstes Programm für einen 32-bit-ARM-Mikrocontroller können wir nun nach einem Druck auf den Reset-Taster ausprobieren.

Schöne Aussichten

Nun ist die erste Folge auch schon zu Ende, wir hoffen, dass sie Ihnen viel Freude bereitet hat. Für eine Rückmeldung oder Fragen steht Ihnen das Elektor-Forum unter [10] offen. In der nächsten Ausgabe geht es dann mit der GPIO-Ansteuerung und mit der U(S)ART-Schnittstelle weiter. Bis dahin können Sie sich mit dem Board, dem Mikrocontroller und auch mit dem Studio weiter

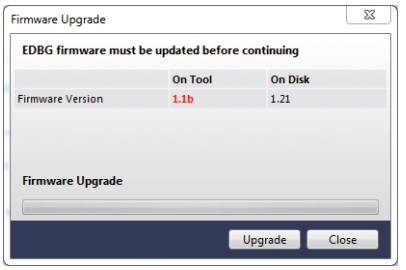


Bild 22.
Beim Firmware-Upgrade des Debuggers sollten beide Status-LEDs auf dem Board blinken.

anfreunden und einige der zahlreichen Beispielprogramme ausprobieren. Diese finden Sie unter *File/New/Example Projekt…* beziehungsweise unter *New Example Projekt…* im Begrüßungsfenster. Viel Spaß damit und bis zur nächsten Folge!

(140037)

Weblinks

[1] www.atmel.com/Images/Atmel-42102-SAMD20-Xplained-Pro_User-Guide.pdf

[2] www.atmel.com/Images/Atmel-42096-Microcontrollers-Embedded-Debugger_User-Guide.pdf

[3] www.atmel.com/images/Atmel-42129-SAM-D20_Datasheet.pdf

[4] www.atmel.com/tools/atproto1-xpro.aspx

[5] www.atmel.com/tools/atio1-xpro.aspx

[6] www.atmel.com/tools/atoled1-xpro.aspx

[7] www.atmel.com/tools/ATQT1-XPRO.aspx

[8] www.atmel.com/tools/atmelstudio.aspx

[9] http://de.wikipedia.org/wiki/Debugger

[10] www.elektor.de/forum

Listing 1. Unser erstes Programm (Ausschnitt).

```
#include <asf.h>

int main (void)
{
  system_init();

  // Insert application code here, after the board has been
  // initialized.

  // This skeleton code simply sets the LED to the state of
  // the button.
  while (1) {
    // Is button pressed?
    if (port_pin_get_input_level(BUTTON_0_PIN) == BUTTON_0_ACTIVE) {
      // Yes, so turn LED on.
      port_pin_set_output_level(LED_0_PIN, LED_0_ACTIVE);
    } else {
      // No, so turn LED off.
      port_pin_set_output_level(LED_0_PIN, !LED_0_ACTIVE);
    }
  }
}
```

Listing 2. Definitionen für ein einfaches Ansprechen der LED und des Buttons.

```
#define LED0_PIN                PIN_PA14
#define LED0_ACTIVE             false
#define LED0_INACTIVE           !LED0_ACTIVE

#define SW0_PIN                 PIN_PA15
#define SW0_ACTIVE              false
#define SW0_INACTIVE            !SW0_ACTIVE
#define SW0_EIC_PIN             PIN_PA15A_EIC_EXTINT15
#define SW0_EIC_MUX             MUX_PA15A_EIC_EXTINT15
#define SW0_EIC_PINMUX          PINMUX_PA15A_EIC_EXTINT15
#define SW0_EIC_LINE            15

#define LED_0_NAME              “LED0 (yellow)”
#define LED_0_PIN               LED0_PIN
#define LED_0_ACTIVE            LED0_ACTIVE
#define LED_0_INACTIVE          LED0_INACTIVE
#define LED0_GPIO               LED0_PIN

#define BUTTON_0_NAME           “SW0”
#define BUTTON_0_PIN            SW0_PIN
#define BUTTON_0_ACTIVE         SW0_ACTIVE
#define BUTTON_0_INACTIVE       SW0_INACTIVE
#define BUTTON_0_EIC_PIN        SW0_EIC_PIN
#define BUTTON_0_EIC_MUX        SW0_EIC_MUX
#define BUTTON_0_EIC_PINMUX     SW0_EIC_PINMUX
#define BUTTON_0_EIC_LINE       SW0_EIC_LINE
```

Platino-Funktionsgenerator

Für Rechteck, Sinus, Sägezahn, Rauschen und mehr

Dieser Funktionsgenerator wurde eher für einfachere Anwendungen entwickelt und ist eine nützliche Ergänzung für ein kleines Elektronik-Labor. Er basiert auf dem Platino-Board von Elektor und eignet sich beispielsweise zur Signalverfolgung, als Taktgeber für digitale Schaltungen oder zum Testen von Audio-Elektronik und mehr.

Von Sunil Malekar (Elektor-Labor Indien)

Ein Funktionsgenerator liefert periodische Signale mit unterschiedlichen Kurvenformen und einstellbarer Frequenz und Amplitude. Zusammen mit einem Multimeter, einem Labornetzteil und einem Oszilloskop gehört ein Funktionsgenerator zur notwendigen Grundausstattung, wenn elektronische Geräte gebaut oder entwickelt werden. Das hier beschriebene Selbstbaugerät beinhaltet das Platino-Board [1] von Elektor als Steuerung und wurde von einer früheren Elektor-Serie inspiriert: dem ausführlich beschriebenen SDR-Projekt mit AVR-Controller aus dem Jahre 2012 [2].

Technische Daten:

- ATmega1284P Mikrocontroller auf Platino
- LCD mit 20 x 4 Zeichen
- Versorgungsspannung: 18...20 V Gleichspannung
- BNC-Buchsen für Ausgänge und Eingang
- Ausgänge:
  - Takt, max. 10 MHz, umschaltbar zwischen 5 V und 3,3 V
  - Sinus, Rechteck, Dreieck, Sägezahn, invertierter Sägezahn, Impulse, arbiträres und zufälliges (Rauschen) Signal; max. 100 kHz, max. 5 V an 50 Ω
- Eingang: Frequenzmodulation (125 kHz ±50 Hz für SDR)
- Bedienung: Drehencoder mit Taster, Zurück-Taster
- Setup mode für Arbitrary und Clock mode
- Normal Mode für andere Kurvenformen
- Einstellung von Kurvenform, Pegel Frequenz und Offset wirkt sich sofort aus
- Nur ein einfacher Abgleich

Die Zutaten

Wenn man das Platino-Board als Baustein betrachtet, werden für digital/analoge Hybrid-Geräte nur wenige Erweiterungen benötigt. Damit vermeidet man Entwicklungsprobleme wie enorm lange Stücklisten oder andere Komplexitäten. Der auf Platino basierende Funktionsgenerator hat demnach zwei Schwerpunkte: Hardware und Software. Beide sind gleichermaßen für die Stabilität und Genauigkeit des Ausgangssignals verantwortlich. Zentrales Bauelement ist sicherlich der Mikrocontroller ATmega1284P samt seiner Firmware. Die Hardware hat zwei Teile: Das Platino-Board mit LCD und das Generator-Add-on-Board. „Generator“ ist hier lediglich symbolisch gemeint, denn generiert wird hier gar nichts, weil die Signalerzeugung ja von Platino vorgenommen wird.

Platino-Board mit LCD

Das Platino-Board ist zusammen mit dem Add-on-Board für alle inneren und externen Signale des

Funktionsgenerators zuständig. Die Verbindungen zwischen den beiden Boards sind deutlich in **Bild 1** zu sehen. Man sieht in dieser Schaltung klar, wie Platino und die Platine des Generator-Boards über die Steckverbinder gekoppelt sind.

Falls Sie es nicht wissen: Platino hat ein hintergrundbeleuchtetes LCD mit 20 x 4 Zeichen für Menüs, Textausgabe und die Anzeige anderer Daten. Zur Eingabe von Werten dient ein Drehencoder mit integriertem Taster. Damit kann zwischen Kurvenformen etc. gewählt werden. Ein weiterer Taster dient als Zurück-Knopf, um gewählte Menüs wieder zu verlassen. Die Belegung von Platino-Pins durch das Add-on-Board ist in **Tabelle 1** zusammengefasst.

Add-on-Board

Wie in Bild 1 zu sehen besteht die Zusatzplatine hauptsächlich aus einem Konverter, einem Verstärker, dem Netzteil sowie den Bauteilen für unterschiedliche Kurvenformen. Jede der fünf Funktionen wird nachfolgend getrennt beschrieben:

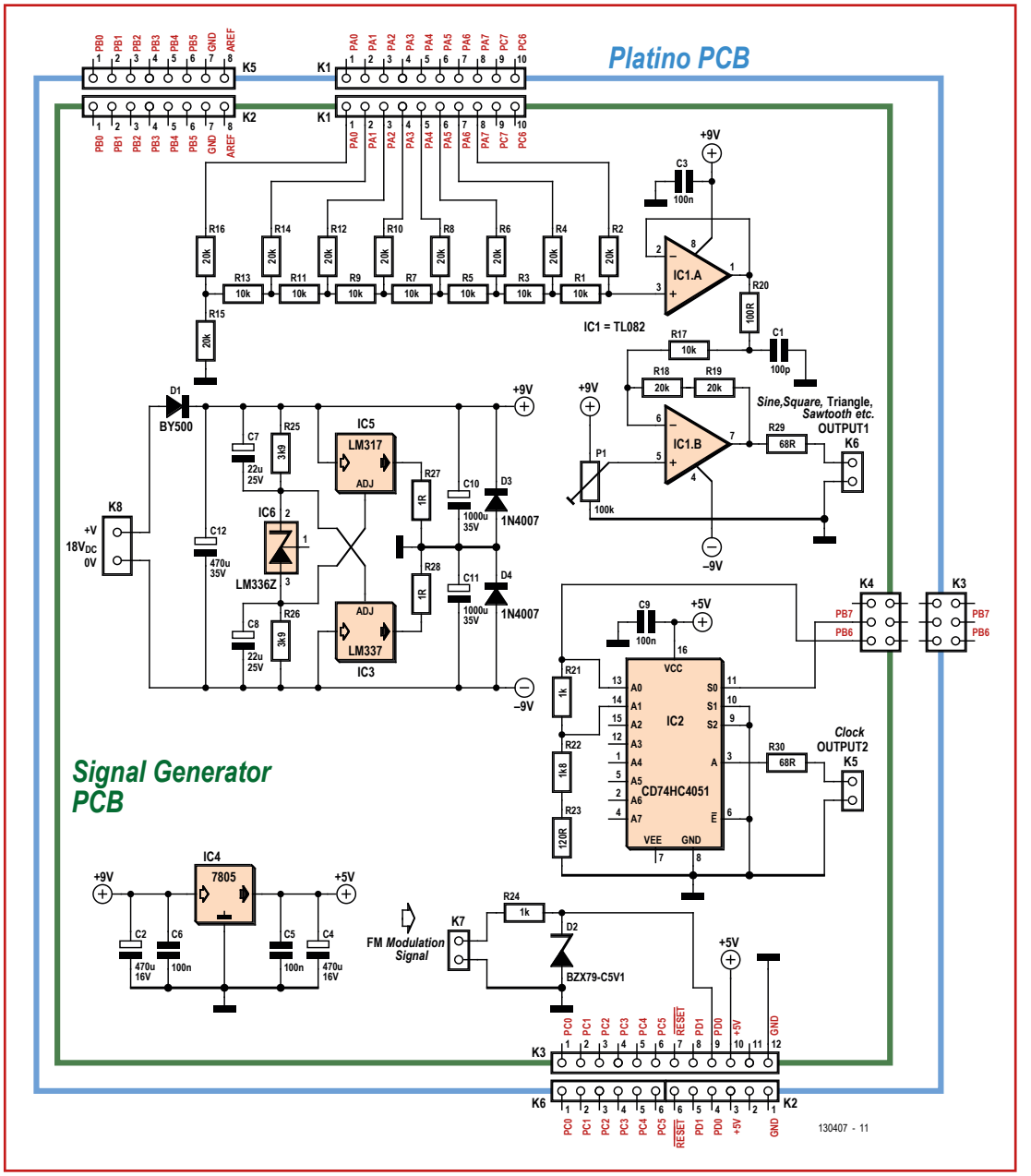


Bild 1.
Die Schaltung des Add-on-Boards mit angedeutetem Platino (Steckverbinder und blaue Linie außen).

Tabelle 1. Verbindung von Platino mit Add-on-Board	
Port / Pin	**Funktion**
PORTA	Steuert R2R-Netzwerk
PB5	Steuert LCD-Beleuchtung
PB3	Zurück-Taster
PB6	Taktausgang
PB7	Steuert Taktpegel zwischen 3,3 V und 5 V
PC0..PC2	Drehencoder mit Taster
PD0	FM-Eingang (5 V digital)

DAC

Der DAC (**D**igital to **A**nalog **C**onverter) besteht aus einem *R2R-Netzwerk*, das von PORTA des ATmega1284 des Platino-Boards angesteuert wird. Das Netzwerk aus R1...R16 konvertiert jeden digitalen 8-bit-Wert in eine analoge Spannung. Der Bereich von 0...5 V ist in 256 Schritte unterteilt.

Verstärker

Die vom *R2R-Netzwerk* gelieferte Spannung gelangt an den Verstärker IC1.A, einer Hälfte des Dual-Opamps TL082. Sein hochohmiger Eingang belastet das *R2R-Netzwerk* kaum und sein niederohmiger Ausgang stellt das Signal für den Tiefpass aus R20 und C1 zur Verfügung, wo störende HF-Reste unterdrückt werden. Von der zweiten Hälfte IC1.B wird das Signal dann vierfach verstärkt. Ein Signalpegel von 5 V_{SS} wird so zu einem Pegel von 20 V_{SS}, der symmetrisch einem theoretischen Spannungsbereich von -10 V bis +10 V entspricht. Mit Trimmpoti P1 wird das mittlere Potential des Signals relativ zur virtuellen Masse eingestellt. Das so erzeugte Signal steht dann unabhängig von der Form (Sinus, Rechteck, Sägezahn oder Dreieck) am Ausgang K6 zur Verfügung.

3,3/5-V-Taktpegel

Die Umschaltung des Pegels für den Taktausgang wird von IC2 erledigt. Bei IC2 handelt es sich um einen analogen Multiplexer/Demultiplexer des Typs 74HC4051. Eingang A0 ist mit dem Platino-Taktausgang PB6 verbunden, an dem ein

Listing 1. BASCOM-Code (Ausschnitt)

```
Select Case Freq     'According to the frequency selects the prescaler value
   Case Is =< 80 : Config Timer1 = Timer , Prescale = 1024
               Pres = 1024
   Case 81 To 280 : Config Timer1 = Timer , Prescale = 256
               Pres = 256
   Case 281 To 2250 : Config Timer1 = Timer , Prescale = 64
               Pres = 64
   Case 2251 To 18000 : Config Timer1 = Timer , Prescale = 8
               Pres = 8
   Case Is => 18001 : Config Timer1 = Timer , Prescale = 1
               Pres = 1
End Select

Fraction = Freq
Fraction = Fraction / 1000000
Fraction = Fraction * 256
Fraction = Fraction * Pres
Fraction = Fraction / 20


Select Case Screen:
      Case 2 : For C = 0 To 1024
                  C1 = C * Fraction
                  C2 = C1
                  If Signal = 5 Then
                  C1 = Dutycycle / 100
                  C1 = 256 * C1
                  B = C1
                  End If
                  If C2 =< 255 Then
                  Select Case Signal:
                     Case 0 : Sig(c + 1) = Lookup(c2 , Sine )       'store sine values to signal variable
                     Case 1 : Sig(c + 1) = Lookup(c2 , Square )     'store Square values to signal variable
                     Case 2 : Sig(c + 1) = Lookup(c2 , Triangular ) 'store Triangular values to signal variable
                     Case 3 : Sig(c + 1) = Lookup(c2 , Sawtooth )   'store Sawtooth values to signal variable
                     Case 4 : Sig(c + 1) = Lookup(c2 , Inv_sawtooth ) 'store Inv_sawtooth values to signal variable
                     Case 5 : If B >= C2 Then
                                 Sig(c + 1) = &HFF       ''store Pulse values to signal variable
                              Else
```

Taktpegel von 5 V anliegt. Eingang A1 hingegen holt den mit dem Spannungsteiler aus R21...R23 auf 3,3 V heruntergeteilten Pegel. IC2 schaltet einen der beiden Pegel abhängig von Zustand von Pin PB7 zum Ausgang A durch. Der zwischen 3,3 V und 5 V umschaltbare Takt steht dann an K5 an.

FM-Eingang

Zur Frequenzmodulation des Generatorsignals wird ein Signal herangezogen, das über K7 eingespeist werden kann. Geeignet sind hierfür digitale Signale mit einem Pegel von 5 V und einem Frequenzbereich von 0...20 kHz. Diode D2 schützt den Mikrocontroller vor Schäden durch zu hohe oder negative Spannungen an Pin PD0.

Stromversorgung

Zur Versorgung des Funktionsgenerators wird an K8 eine stabilisierte oder „rohe“ Gleichspannung von 18 V angelegt. Wie für andere Geräte dieser Serie eignet sich dazu gut ein Vorgängerprojekt aus dieser Reihe, das Platino-Experimentier-Netzgerät [3]. Die Elektronik des Add-on-Board splittet diese Spannung in einen positiven und einen negativen Zweig mit gemeinsamer virtueller Masse.

Die Versorgungsspannung wird zunächst über die beiden Widerstände R25 und R26 sowie die Spannungsreferenz IC6 in zwei Hälften geteilt. Die einstellbaren Spannungsregler IC3 und IC5 erzeugen eine virtuelle Masse für das Netzteil. Damit können die erzeugten Signale theoretisch Pegel von –9 V und +9 V bzw. 18 V_{SS} erreichen, obwohl nur ein preiswertes Netzteil mit einer unsymmetrischen Ausgangsspannung von 18 V eingesetzt wird. Die Dioden D1, D3 und D4 schützen vor Verpolung. Die virtuelle Masse für den

```
                              Sig(c + 1) = &H00
                           End If
                  Case 6 : Sig(c + 1) = Arbitary(c2 + 1) + 127       'store Arbitary values to signal variable
                  Case 7 : Sig(c + 1) = Rnd(255)      'store Random values to signal variable for noise generation
                  Case 8 : Enable Pcint3              'Enable pin change interrupt for FM
                  Case Else : Sig(c + 1) = 127
               End Select

               C1 = Ampl / 10                         'Amplitude calculation using user input
               C3 = 127 * C1
               C3 = 127 - C3
               C1 = Sig(c + 1) * C1
               C1 = C1 + C3

               C3 = Offset / 10                       'Offset calculation using user input
               C3 = -1 * C3                           'compensate 180 degree phase shift by OP-Amp
               C3 = C3 * 127
               C1 = C1 + C3                           'Adding offset value to the signal

               If C1 > 255 Then                       'Limit the signal value if overflow happens
               C1 = 255
               End If
               If C1 < 0 Then
               C1 = 0
               End If
               Sig(c + 1) = C1
               C21 = C - 1
               End If
         Next
  Case Else : Disable Pcint3                  'Disable pin change interrupt at the FM input
              For C = 0 To 1030
                  Sig(c + 1) = 127
              Next
              C21 = 1024
End Select


Flag = 0
Disable Timer0
End If
Return
```

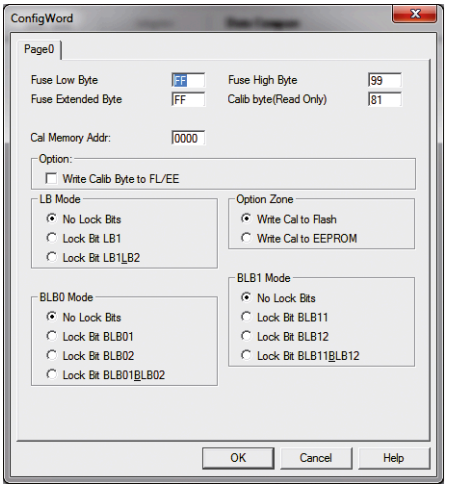


Bild 2.
Screenshot der Einstellung der Fuses für den Mikrocontroller ATmega1284P.

analogen Schaltungsteil liegt an der Verbindung von R27 und R28 an.
Die +9-V-Schiene speist außerdem auch den 5-V-Spannungsregler IC4, der den digitalen Teil samt Platino mit den typischen 5 V versorgt.
Falls die Schaltung mit einer höheren Spannung von z.B. 20 V versorgt wird, ergeben sich zwei Spannungen von +10 V und -10 V (theoretisch). Die Symmetrie bleibt also gewahrt.

Software

Die Firmware für dieses Projekt wurde mit BASCOM AVR geschrieben. Source-Code und Hex-Datei sowie andere Zutaten stehen wie immer kostenlos auf der Webseite zu diesem Artikel [4] zum Download zur Verfügung. Ein Teil des Codes ist in **Listing 1** zu sehen. Bezüglich der Fuses des Mikrocontrollers richtet man sich nach den Angaben im Screenshot von **Bild 2**. Die Software für den Funktionsgenerator ist in zwei Abschnitte geteilt, die getrennt erläutert werden.

Normal Mode

Im normalen Modus wählt man mit dem Drehencoder und dem Back-Taster zwischen den verschiedenen Kurvenformen und stellt die Frequenz, die Amplitude und den Offset ein. Alle Einstellungen werden sofort wirksam.
Im normalen Modus gibt es auch die FM-Funktion. Wenn der Mikrocontroller an seinem Pin PD8 ein „high“ vorfindet, wird ein Rechtecksignal von 125 kHz +50 Hz generiert. Bei einem „low“ ändert sich die Frequenz auf 125 kHz –50 Hz. Diese Funktion ist vor allem für Experimente mit einem Software Defined Radio gedacht.

Setup Mode

Im Einstellungsmodus gibt es zwei Optionen:

1. Arbitrary Mode: Hier kann man Werte nach Wunsch eingeben und so ein spezielles Signal generieren. Das arbiträre Signal muss im normalen Modus ausgewählt werden.
2. Clock Mode: In diesem Modus kann man die Frequenz des zusätzlichen Takts einstellen, der parallel zum eigentlichen Signal an K5 ausgegeben wird.

Zur Erzeugung eines Signals mit der gewünschten Frequenz verwendet man zwei Timer und eine 255 Bytes große Lookup-Tabelle, in der die Wellenform abgespeichert ist. Der erste (schnelllaufende)

Tabelle 2. Platino Jumper (Lötbrücken)

Jumper	Pin
JP4	PB0
JP5	PB1
JP6	PB2
JP7	PB3
JP11	PB7
JP12	PB6

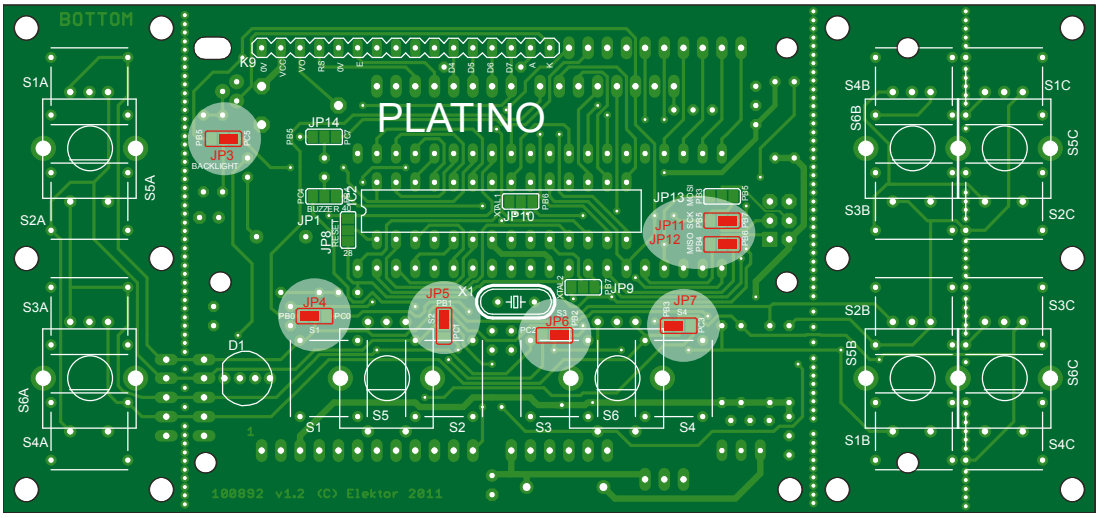


Bild 3.
Lötbrücken für den Einsatz von Platino in diesem Funktionsgenerator.

Stückliste

Add-on-Board

Widerstände:
Standard: 5 %, 0,25 W
R1,R3,R5,R7,R9,R11,R13,R17 = 10 k
R2,R4,R6,R8,R10,R12,R14,R15,R16,R18,R19 = 20 k
R20 = 100 Ω
R21,R24 = 1 k
R22 = 1k8k
R23 = 120 Ω
R25,R26 = 3k9
R27,R28 = 1 Ω, 2 W
R29,R30 = 68 Ω, 1 %
P1 = 100 k, Mehrgang-Trimmpoti, vertikal

Kondensatoren:
C1 = 100 p
C2,C4 = 470 µ / 16 V, radial
C3,C5,C6,C9 = 100 n, radial
C7,C8 = 22 µ / 25 V, radial
C10,C11 = 1000 µ / 35 V, radial
C12 = 470 µ / 35 V, radial

Halbleiter:
IC1 = TL082ACP
IC2 = (CD)74HC4051
IC3 = LM337KCSE3
IC4 = MC7805
IC5 = LM317TG
IC6 = LM336BZ-5.0
D1 = BY500-800-E3/4
D2 = BZX79-C5V1
D3,D4 = 1N4007

Außerdem:
K1 = 10-pol. Buchsenleiste, einreihig, gerade
K2 = 8-pol. Buchsenleiste, einreihig, gerade
K3 = 12-pol. Buchsenleiste, einreihig, gerade
K4 = 6-pol. Buchsenleiste, zweireihig (2x3), gerade
K5,K6,K7 = 2-pol. Stiftleiste, gerade, RM 0,1”
K8 = 2-pol. Schraubklemme für Platinenmontage, RM 0,2”
IC-Sockel, DIP-16
IC-Sockel, DIP-8
Platine 130407-1

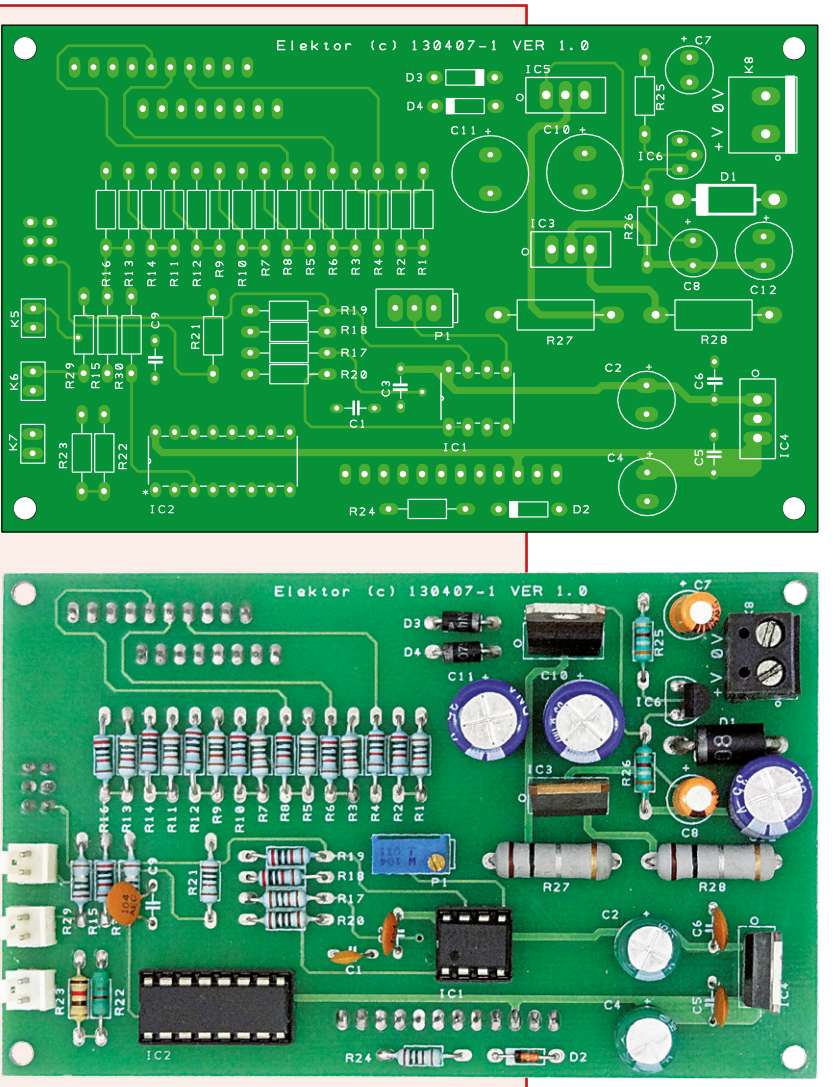


Bild 4.
Bestückungsplan der zweiseitigen Add-on-Platine.

Platino-Stückliste*

Widerstände:
Standard: 5 %, 0,25 W
R3 = 47 Ω
R4,R5,R6,R7,R10,R12 = 10 k
R11 = 4k7
P1 = 10 k, Trimmpoti, horizontal

Kondensatoren:
C1,C2 = 22 p / 50 V, C0G/NP0, RM 0,1”
C5,C6 = 100 n / 50V, X7R, RM 0,2”

Halbleiter:
IC1 = ATmega1284P-PU, programmiert
T1 = BC547C

Spulen:
L1 = 10 µ

Außerdem:
IC-Sockel, DIP-40
LCD1 = LCD, 20x4, 5 V, mit Hintergrundbeleuchtung
S5A = Drehencoder mit Taster
X1 = Quarz, 20 MHz, C_L = 18 p
K1,K2,K5 = 40-pol. Stiftleiste, einreihig, gerade
K3 = 6 Pins einer 80-pol. Stiftleiste, zweireihig, gerade
K9 = 16 Pins einer 36-pol. Buchsenleiste, einreihig, gerade
S4A = Taster

*Siehe [3] für Beschreibung von Platino.

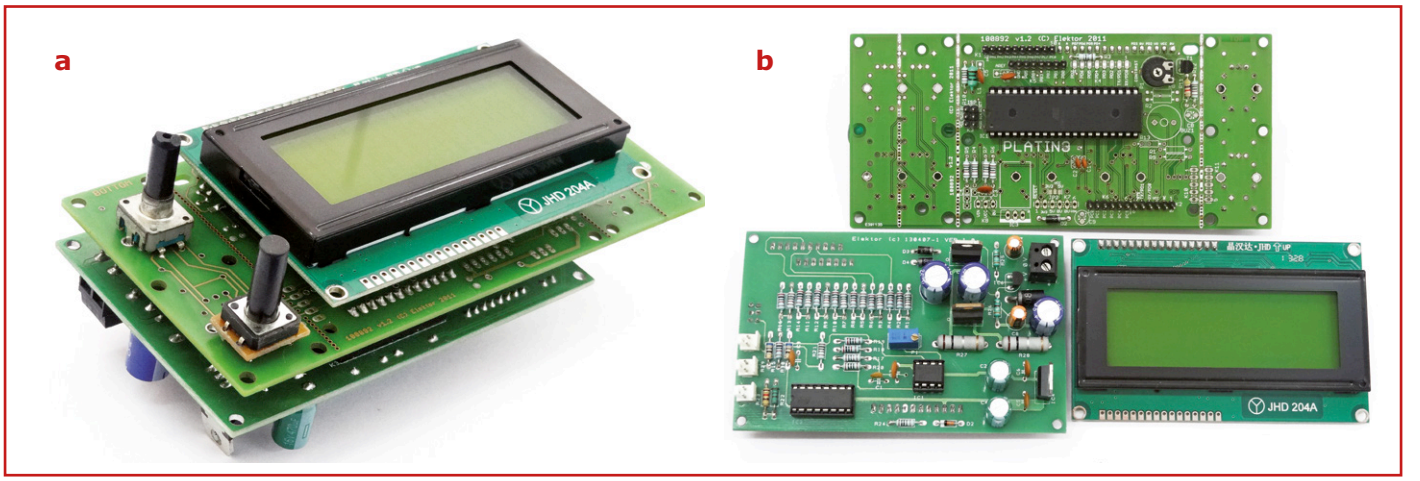


Bild 5.
(a) Diese drei Platinen werden aufeinander gesteckt (von oben nach unten): LCD, Platino und Add-on-Board.
(b) Die drei Boards separat.

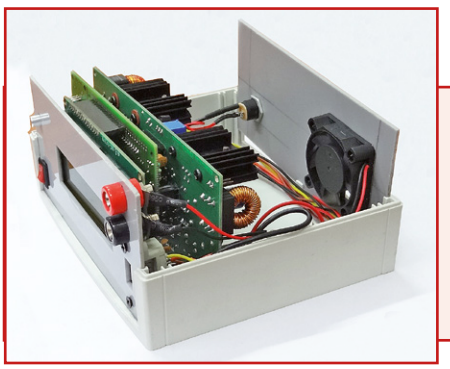
Timer ist für ein rasches, periodisches Sampling verantwortlich; der zweite Timer wird anhand der gewünschten Frequenz eingestellt und fungiert als Counter. Bei jedem Sampling wird der gegenwärtige Stand des Counters abgefragt und als Index für die Wellenform-Tabelle verwendet; der dort stehende Wert wird an den DAC weitergegeben.

Bau und Test

Zuerst baut man Platino mitsamt LCD, Drehencoder mit Taster sowie dem zusätzlichen Taster auf. Auch der Mikrocontroller ATmega1284P und die anderen Bauteile dürfen nicht fehlen. Dann wird das Board wie in **Tabelle 2** „gejumpert". Dies erledigt man mit Lötbrücken - siehe dazu den originalen Platino-Artikel [3] und **Bild 3**.
Auf das Add-on-Board kommen nur bedrahtete Bauteile, womit dessen Aufbau einfach ist. Der Bestückungsplan und die Bauteile sind in **Bild 4** zu sehen. Obwohl es sich um bedrahtete Bauteile handelt, werden sie doch auf beiden Seiten der doppelseitigen Platine bestückt. Im Zweifel richtet man sich einfach nach den Fotos dieses Artikels. Außerdem muss man auch auf die etwas spezielle Bestückung von Platino achten. Deshalb ist die Stückliste hierfür getrennt aufgeführt.
Nach der Bestückung empfiehlt sich die sorgfältige Kontrolle der Lötarbeiten – bevor man die Sache einschaltet. Wenn alles in Ordnung ist, kann man an K8 die erforderlichen 18 V Gleichspannung anlegen.
Bevor man nun ein Signal auswählt, sollte man den Funktionsgenerator kalibrieren. Das ist einfach, denn es betrifft lediglich P1, mit dem man 0 V an K6 einstellt. Mehr einzustellen gibt es nicht.
Die Add-on-Platine passt genau auf die Rückseite von Platino und wird einfach via K1...K4 aufgesteckt. Das fertige Gerät passt gut in ein Bopla-Gehäuse. Wie bei den anderen Geräten der Platino-Serie ist das Projekt aus drei Einheiten aufgebaut: LCD, Platino und das Add-on-Board (**Bild 5**). Das zusammengesteckte Paket kann man dann hinter eine Alu-Frontplatte wie in **Bild 6** montieren. Auf der Frontplatte befinden sich drei BNC-Buchsen: Wave out (F1 Out), Clock out (F2 Out) und FM in (MOD in).

Menü und Kurven

Die Menüstruktur und die Auswahl von Kurvenformen, Pegeln und Frequenzen sollte intuitiv sein. Man nutzt den Drehencoder von Platino



Andere Geräte dieser Reihe

Um die Flexibilität von Platino in Verbindung mit BASCOM zu demonstrieren hat Elektor Indien eine Reihe von Test- und Messgeräten für das Elektronik-Labor entwickelt. Der *Platino-Funktionsgenerator* in dieser Folge ist das zweite Gerät dieser Serie. Das erste Gerät war das *Platino-Experimentier-Netzteil* vom April 2014. Seien Sie gespannt: Es kommen noch vier weitere Geräte.

und den zusätzlichen Taster. Jede Änderung der Einstellungen wirkt sich unmittelbar aus. Egal wo man gerade im Menü ist, kommt man mit dem Zurück-Taster wieder auf den vorherigen Bildschirm zurück. **Bild 7** zeigt eine zufällige Auswahl an Texten auf dem LCD.

Für den Taktausgang kann entweder ein Pegel von 5 V oder 3,3 V gewählt werden. Diese Einstellung wird im Tab *Clock mode* im *Setup mode* vorgenommen.

Zum Schluss zeigt **Bild 8** eine Auswahl an mit dem Funktionsgenerator erzeugten Signalen.

(130407)

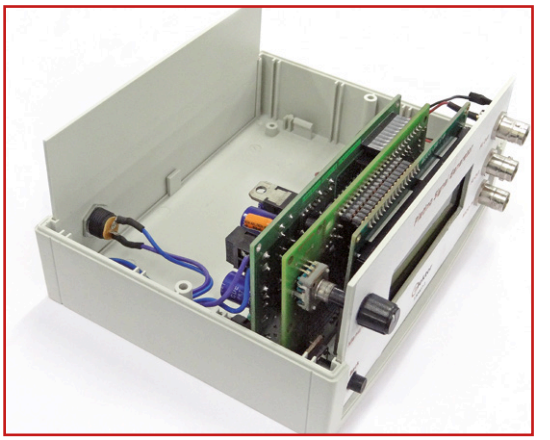


Bild 6.
Das Paket aus drei Platinen in einem Bopla-Gehäuse.

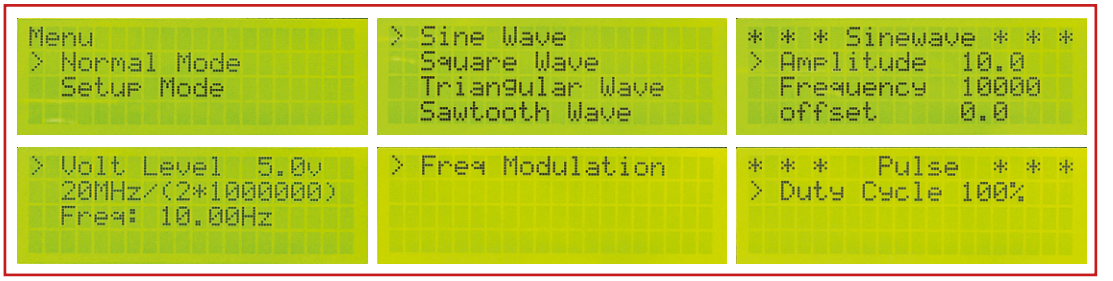


Bild 7.
Auswahl an Meldungen, die auf dem LCD erscheinen können.

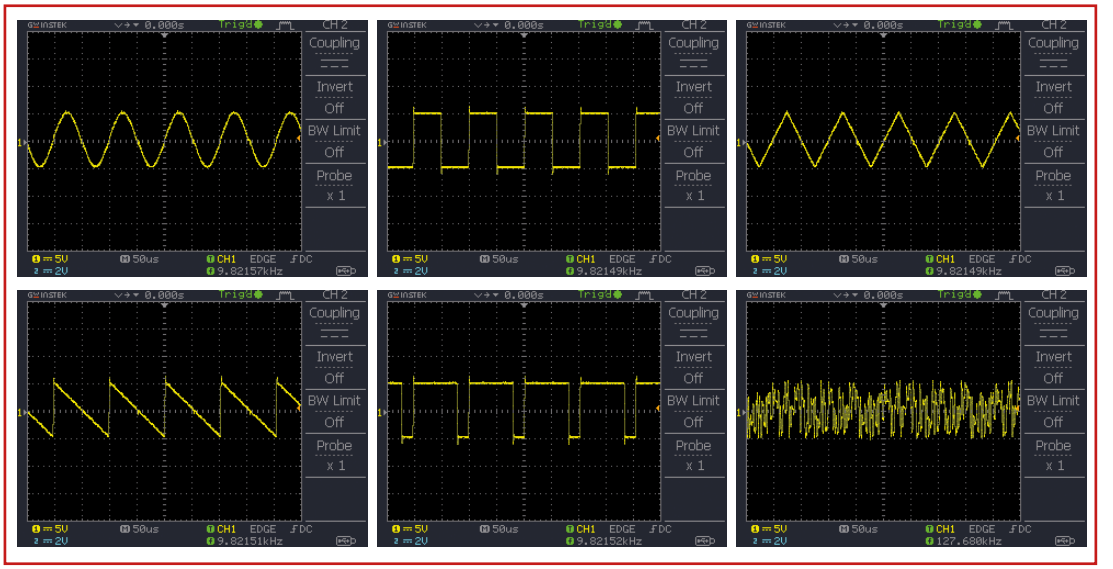


Bild 8.
Kurvenformen und Rauschsignal des Platino-Funktionsgenerators.

Weblinks

[1] Platino: Elektor Oktober 2011: www.elektor-magazine.de/100892

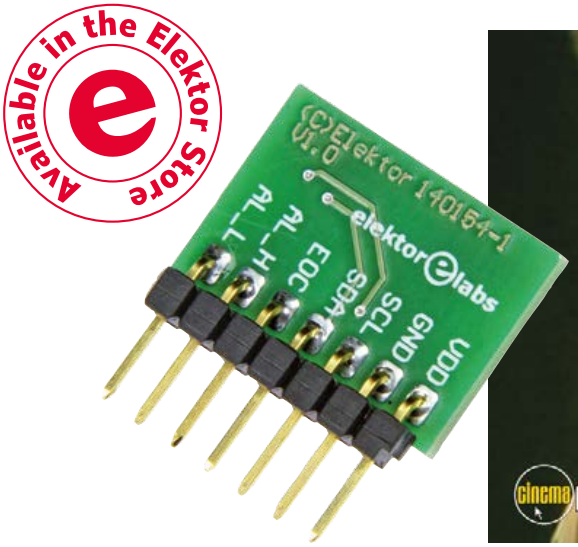
[2] AVR-SDR-Serie: Elektor März bis September 2012: www.elektor-magazine.de/100180 (erster Artikel)

[3] Platino-Experimentier-Netzgerät: Elektor April 2014, www.elektor-magazine.de/130406

[4] Projekt-Downloads: www.elektor-magazine.de/130407

CC2-eBoB

Feuchte/Temperatur-Sensor ChipCap2



Von **Luc Lemmens**
(Elektor-Labor)

Klar, sie können dem cleveren Feuchte/Temperatursensor-IC ChipCap2 mit dem Lötkolben zu Leibe rücken. Sein Footprint passt aber auf kein Steckboard dieser Welt. Ein kleines Platinchen löst das Problem. Es führt alle Anschlüsse des Sensors auf eine 0,1“-Stiftleiste und macht sie für den Rest der Elektronik-Welt zugänglich. Unser zweites (e-)BoB!

Das ist schon eine Versuchung: Der individuell kalibrierte und getestete Feuchte/Temperatur-Sensor ChipCap2 (auch CC2 genannt) weist einen Fehler von nur ±2% im Bereich 20...80% relativer Luftfeuchtigkeit oder ±3% über den gesamten Feuchtebereich auf. Er ist sehr einfach anzuwenden und erfordert keinerlei Kalibrierung oder Temperaturkompensation. Da läuft einem das Wasser im Munde zusammen! Ich will einen! Am besten auf unserem speziell entwickelten eBoB.

Sensoren im Keller

Wir haben diese kleine Platine als Teil eines größeren Projekts, bei dem innen und außen die relative Luftfeuchtigkeit (RH) und die Temperatur gemessen werden sollte, auf der Elektor.Labs-Webseite [1] veröffentlicht. Abhängig von den Messergebnissen wird ein Fenster geöffnet oder geschlossen und ein Lüfter ein- oder ausgeschaltet, um die Feuchtigkeit in einem geschlossenen Raum (Keller) in einem akzeptablen Bereich zu halten. Das System berechnet und berücksichtigt die absoluten Feuchtigkeitswerte und überwacht die Innentemperatur zum Frostschutz.
Im ursprünglichen Entwurf wurden zwei Feuchte/Temperatursensoren HYT939 von Innovative Sensor Technology (IST) verwendet. Dieser Sensor ist aber sehr teuer (und zwei noch mehr), so dass wir beschlossen, einen erschwinglichen ChipCap2-Feuchtesensor von Amphenol für etwa ein Fünftel der Kosten zu verwenden.
Beide Sensortypen sind mit dem Steuersystem über einen I^2C-Bus verbunden und weisen auch

den gleichen I²C-Adressbereich, die gleichen Befehle und so weiter auf. Es scheint, dass beide Sensoren mit nur geringfügigen Software-Änderungen verwendet werden können. Garantieren möchten wir das nicht, weil wir den HYT939 (bisher) noch nicht im Detail untersucht haben.

Analoge und digitale Schnittstellen

Der ChipCap2 kann entweder in I²C-Modus (Typ CC2D wie in unserer Anwendung) oder im Pulse-Density-Modus (PDM; Typ CC2A) arbeiten. Im zweiten Fall stehen an zwei Pins nach passiver Filterung analoge Temperatur- und Luftfeuchtigkeitssignale zur Verfügung. **Tabelle 1** listet alle verfügbaren Versionen auf. Unser Breakout-Board ist sowohl für analoge als auch digitale Versionen der Sensoren geeignet, nur einige Bauteile müssen geändert werden. Hier steht allerdings die I²C-Version CC2D im Vordergrund.

Tabelle 1. Varianten des CC2 (ChipCap2)

Bauteil	Beschreibung
CC2A25	analog, 2%, 5 V
CC2A23	analog, 2%, 3,3 V
CC2D23S	digital, sleep mode, 2%, 3,3 V
CC2D25S	digital, sleep mode, 2%, 5 V
CC2D23	digital, 2%, 3,3 V
CC2D25	digital, 2%, 5 V
CC2D35	digital, 3%, 5 V
CC2A33	analog, 3%, 3,3 V
CC2D33S	digital, sleep mode, 3%, 3,3 V
CC2D35S	digital, sleep mode, 3%, 5 V
CC2D33	digital, 3%, 3,3 V
CC2A35	analog, 3%, 5 V

Sensor-Pads

Die Power-Pads 6 und 7 (nicht: Pins) VDD und VSS werden für den Anschluss einer Versorgungsspannung von 2,7...5,5 V (abhängig von der genauen Version des CC2) verwendet. Die Pads werden mit einem 220-nF-Kondensator entkoppelt. VCORE ist eine interne Spannung des CC2. Hier sollte nur ein 100-nF-Entkopplungskondensator nach Masse angeschlossen werden. Daten werden zum und vom Chip über das SDA-Pad (3) übertragen, die Synchronisierung der Kommunikation zwischen ChipCap2 und dem Mikrocontroller (MCU) erfolgt über das SCL-Pad 4. Der interne temperaturkompensierte Oszillator des ChipCap2 stellt eine Zeitbasis für alle Operationen zur Verfügung. Es werden I²C-kompatible Kommunikationsprotokolle mit Bitraten von 100 kHz bis 400 kHz unterstützt. Es sind externe Pull-up-Widerstände erforderlich, um die Leitungen auf High zu ziehen.

An den Alarmausgängen (Pads 1 und 8) kann überwacht werden, ob der Sensorwert über oder unter vorprogrammierte Werte der relativen Luftfeuchtigkeit gestiegen respektive gesunken ist. Der Alarm kann selber als Treiber (full push pull) fungieren oder eine Open-Drain-Last an VDD (oder einer externen, noch höheren Spannung) steuern, wie **Bild 1** zeigt.

Die beiden Alarmausgänge können gleichzeitig und auch gleichzeitig mit dem I²C-Bus verwendet werden. Die Schwellwerte, die die Alarmausgänge aktivieren und deaktivieren, die Ausgangskonfiguration (Open-drain oder Push-pull) und der aktive Ausgangspegel werden im internen EEPROM der

Eigenschaften ChipCap2 (CC2)

Sensor: Relative Feuchte (% RH)

- Auflösung: 14 bit (0,01 % RH)
- Fehler: ±2,0 % RH (20...80 % RH)
- Reproduzierbarkeit: ±0,2 % RH
- Hysterese: ±2,0 % RH
- Linearität: <2,0 % RH
- Ansprechzeit: 7,0 s (auf 63 %)
- Temperaturkoeffizient: Max 0,13 % RH/°C (bei 10...60°C, 10...90 % RH)
- Anzeigebereich: 0...100 % RH (nicht betauend)
- Langzeitdrift: <0,5 % RH/a (Normale Bedingungen)

Sensor: Temperatur (°C)

- Auflösung: 14 bit (0,01°C)
- Fehler: ±0,3°C
- Reproduzierbarkeit: ±0,1°C
- Ansprechzeit: 5,0 s (auf 63 %)
- Anzeigebereich: -40...+125°C
- Langzeitdrift: <0,05°C/a (Normale Bedingungen)

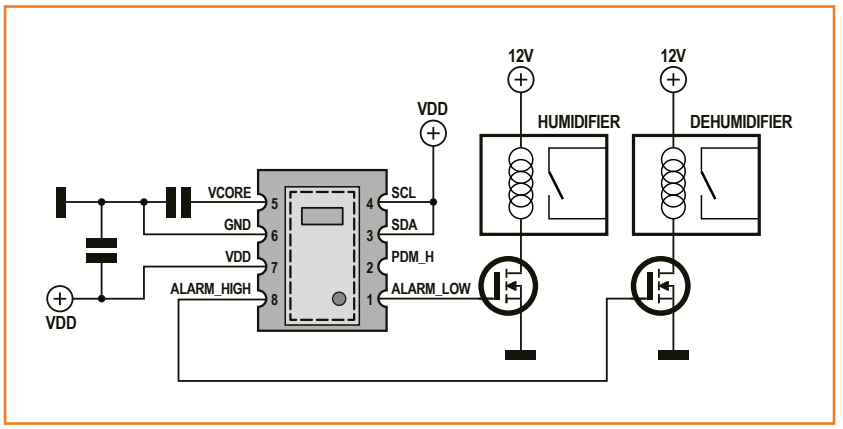


Bild 1.
ChipCap2 in einer Stand-alone-Applikation.

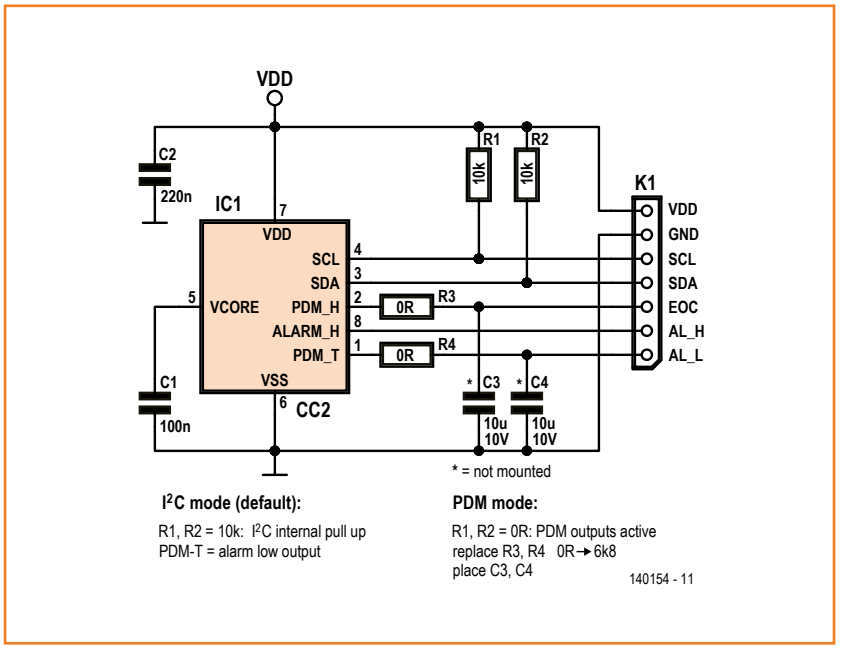


Bild 2.
Kleine Chips – kleine Schaltung! Hier der Feuchte/Temperatur-Sensor CC2A in seiner Minimalkonfiguration.

CC2A eingestellt (dazu gleich mehr). Sind die Anschlüsse über den I²C-Bus konfiguriert, kann ChipCap 2 im Stand-alone-Modus arbeiten. Nicht einmal die Hilfe eines Mikrocontrollers ist erforderlich, um nun eine einfache Feuchtesteuerung zu realisieren.

Eine steigende Flanke am Ready-Anschluss zeigt an, dass neue Daten von der I²C-Schnittstelle abgerufen werden können. Ready bleibt High, bis ein Data Fetch-Befehl (DF) gesendet wird und auch, wenn weitere Messungen vor dem DF durchgeführt werden. Der Typ des Ausgangstreibers, entweder Full-Push-Pull oder Open-Drain, wird mit dem `Ready_Open_Drain`-Bit im EEPROM-Wort `Cust_Config` eingestellt. Point-to-point-Kommunikation erfordert normalerweise den Full-Push-Pull-Treiber. Wenn der CC2 in einer Anwendung mit mehreren Zielen kommuniziert, verwendet man die Open-Drain-Option. Dann stattet man die Leitung mit einem Pull-up-Widerstand aus und verbindet sie mit allen in Frage kommenden Bauteilen.

Informationen zum PDM-Modus finden Sie im *ChipCap 2 Application Guide* [2]. In diesem Modus wird das TEMP_ALARM_L-Pad zur Temperaturan-

Messmodi

Der CC2 (ChipCap2) ist ab Werk so programmiert, dass er entweder im Sleep- oder im Update-Modus arbeitet. Im Sleep-Modus wartet der Sensor auf Befehle des Masters, um Messungen vorzunehmen.

Data Fetch im Update-Modus

Im Update-Modus werden die Daten aus dem digitalen Ausgangsregister mit dem Befehl Data Fetch (DF) über den I²C-Bus geholt. Durch Abfrage oder Überwachung des Ready-Anschlusses wird ermittelt, ob Daten zum Abholen bereitstehen. Die Statusbits geben Auskunft, ob die Daten gültig oder veraltet sind. Nach Beendigung eines Messzyklus können gültige Daten abgerufen werden. Wenn das nächste Data Fetch zu früh durchgeführt wird, sind die Daten die gleichen wie beim vorherigen Abruf, allerdings mit einem „veralteten“-Statusbit. Man kann auch die steigende Flanke an Ready verwenden, um immer gültige Daten abzurufen.

Data Fetch im Sleep-Modus

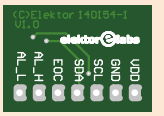
Im Sleep-Modus führt der CC2-Core dann Konvertierungen durch, wenn eine Messung angefordert wird (Measurement Request, MR). MR-Befehle können nur über I²C gesendet werden, so dass der Sleep-Modus nicht im PDM zur Verfügung steht.

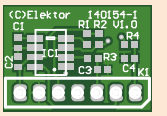
Stückliste

Widerstände:
R1,R2 = 10 k, 1 %, 100 mW, Gehäuse 0603
R3,R4 = 0 Ω, 100 mW, Gehäuse 0603

Kondensatoren:
C1 = 100 n, 16 V, 10 %, X7R, Gehäuse 0603
C2 = 220 n, 10 V, 10 %, X7R, Gehäuse 0603

Halbleiter:
IC1 = CC2D35 Humidity/Temperature Sensor, digital (Amphenol Advanced Sensors)





Außerdem:
K1 = 1x7-polige Stiftleiste, RM 0,1"
Oder:
Fertig aufgebautes CC2-eBoB:
Elektor-Shop 140154-91

Bild 3.
Bestückung der CC2-eBoB-Platine.

zeige verwendet, nur der Ausgang ALARM_H steht für den Feuchtewert zur Verfügung.

Von Pads zu Pins

Die Hardware des CC2-eBoBs in **Bild 2** ist einfach, es ist im Grunde nicht mehr als ein kleines Board, das die Sensoranschlüsse auf eine 7-polige Stiftleiste im 0,1"-Rastermaß führt. Für die meisten I²C-Anwendungen dürften vier Pins (SDA, SCL, VDD und GND) reichen. Die anderen drei Pins EOC/Ready, AL_H und AL_L können entfallen, um Platz auf dem Steckboard oder der Zielapplikation zu sparen.
Stromversorgung und Vcore-Pads sind wie im Datenblatt gefordert entkoppelt. Das Board verfügt über 10-kΩ-Pull-up-Widerstände an den Leitungen SDA und SCL, ausreichend für die meisten Standard-Geschwindigkeiten von I²C-Anwendungen. Bis zu vier CC2-eBoBs können an einem Bus ohne Modifikationen angeschlossen werden. Der Pull-up-Widerstand an beiden Leitungen beträgt dann 2,5 kΩ, was dem Minimalwert für I²C entspricht. Wenn mehr Sensoren benötigt werden, müssen deren Pull-up-Widerstände R1 und R2 ausgelötet werden.
Im PDM-Modus können die Null-Ohm-Widerstände R3 und R4 durch geeignete Werte ersetzt und C3 und C4 hinzugefügt werden, um die Temperatur- und Relative-Feuchte-Signale zu filtern. Um PDM zu aktivieren, müssen R1 und R2 kurzgeschlossen werden, SDA und SCL also mit VDD verbunden werden.

Software aus der Bibliothek

Bevor wir einige Software-Aspekte des CC2 diskutieren, sollten Sie sich in den Infokästen über die Funktionalität des **Measurement Mode** und des **Command Mode** informieren.
I²C-Bausteine besitzen feste Adressen auf dem I²C-Bus. Viele ICs besitzen einen oder mehrere externe Adress-Pins, um zu ermöglichen, dass mehrere identische Chips mit eindeutigen Adressen am gleichen Bus arbeiten können. Im ChipCap2 ist jedoch eine vordefinierte I²C-Adresse im internen EEPROM im Custom Configuration Register (siehe Tabelle) gespeichert. In der Werkseinstellung reagieren alle CC2s auf die gleiche Adresse, die nur durch Software verändert werden kann.

Bei [3] fanden wir eine sehr nützliche, umfangreiche Arduino- (und Python-) Bibliothek von Richard Wardlow, die die meisten Definitionen und Funktionen enthält, die man benötigt, um mit den Sensoren in I²C-Modus zu kommunizieren. Es gibt auch einen einfachen Beispiel-Sketch mit Sensoreinstellungen und dem Auslesen von Feuchte- und Temperaturwerten über den Serial Monitor der Arduino-IDE. Dieses Beispiel kann für erste Tests des Breakout-Boards verwendet werden. Natürlich müssen die I2C-Signale SDA und SCL an die entsprechenden Pins des Arduino-Boards angeschlossen werden.

Die Bibliothek enthält auch Funktionen für den READY-Anschluss (End Of Conversion), wir ziehen

CC2A im PDM-Modus

Obwohl in diesem Artikel der Fokus auf dem I²C-Modus liegt, kann der CC2-eBoB auch im PDM-Modus arbeiten. In diesem Fall liefern Pin 1 und 2 gepulste Signale, welche die vom Sensor gemessenen Temperaturwerte repräsentieren. Nach Glättung durch ein passives Filter erster Ordnung (mit R4/C4 und R3/C3, näher im CC2A-Application Guide beschrieben) können die Werte mit A/D-Wandlungen erfasst und berechnet werden.
Um diesen Modus zu verwenden, werden beide I²C Leitungen mit VDD verbunden. Dazu schließen Sie die 10-kΩ-Widerstände R1 (SCL) und R2 (SDA) auf dem eBoB einfach kurz.

Befehlsmodus

Befehle im Command-Mode (siehe **Tabelle 2**) können nur über das I²C-Protokoll abgesetzt werden. ChipCap2-Sensoren speichern Parameter für die Alarmpads (wie Schwellwerte und Pinkonfiguration), die Ready-Pad-Konfiguration, die Länge des Befehlsfensters und die I²C-Adresse im internen EEPROM.
Nach dem Einschalten öffnet sich ein kurzes Zeitfenster (Command Window), in dem der Command-Modus durch Senden einer Startbedingung aktiviert werden kann. Danach darf der Benutzer auf den EEPROM-Adressbereich von 0x16 bis 0x1F zugreifen (siehe **Tabelle 3**), bis der Befehl Start Normal Mode ausgegeben wird, der - wie der Name schon sagt - den Sensor in den Normalbetrieb versetzt.
Eine Änderung der EEPROM-Parameter oder zumindest der Eintritt in den Befehlsmodus muss sofort nach dem Einschalten der Schaltung erfolgen. Alternativ kann der VDD-Pin des Sensors per Software von einem Pin eines Mikrocontrollers geschaltet werden. So kann der Sensor jederzeit zurückgesetzt werden, wenn eine Änderung der Konfiguration während der Programmausführung erforderlich ist. Schauen Sie für solch weitgehende Benutzerkonfigurationen **Tabelle 4** an.

es aber vor, die Statusbits des Sensors über I²C abzufragen, um festzustellen, ob Feuchte- und Temperaturwerte bereit sind, gelesen zu werden. Leider unterstützt diese Bibliothek nicht die Re-Programmierung der I²C-Adresse. Wir haben deshalb dafür einen einfachen Sketch geschrieben (CC2A_set_I2C_address.ino). Das Ergebnis finden Sie in [4]. Beachten Sie, dass die Stromversorgung des Breakout-Boards an einen von der Software gesteuerten Digital-Ausgang (PB0)

Tabelle 2. Liste der Befehle und Bedeutung

Befehlsbyte 8 Befehlsbits (Hex)	Drittes/Viertes Byte 16 Data Bits (Hex)	Beschreibung	Response Time
0x16 bis 0x1F	0x0000	EEPROM Lesen der Adressen 0x16 bis 0x1F Nach Senden und Ausführen des Befehls müssen Sensor-Daten mit einem Data Fetch neu erfasst werden.	100 µs
0x56 bis 0x5F	0xYYYY (Y=data)	EEPROM Schreiben der Adressen 0x16 bis 0x1F Die beiden Datenbytes werden zur Adresse geschrieben, die in den letzten 6 Bits des Befehlsbytes genannt wird.	12 µs
0x80	0x0000	Start_Norm Beendet Befehlsmodus und Übergang zum Normalen Betriebsmodus	
0xA0	0x0000	Start_CM Start Command Modus: Startet den Befehls-Interpretationsmodus. Start_CM ist nur im Power-on-Fenster möglich.	100 µs

Tabelle 3. EEPROM-Adressbereich (Ausschnitt)

EEPROM Word	Bitbereich	Voreinstellung	Name	Beschreibung
16HEX	13:0	0x3FFF	PDM_Clip_High	PDM High Clipping-Grenze
17HEX	13:0	0x0000	PDM_Clip_Low	PDM Low Clipping-Grenze
18HEX	13:0	0x3FFF	Alarm_High_On	High Alarm Einschaltschwelle
19HEX	13:0	0x3FFF	Alarm_High_Off	High Alarm Ausschaltschwelle
1AHEX	13:0	0x0000	Alarm_Low_On	Low Alarm Einschaltschwelle
1BHEX	13:0	0x0000	Alarm_Low_Off	Low Alarm Ausschaltschwelle
1CHEX	15:0	0x0028	Cust_Config	Customer-Konfiguration
1DHEX	15:0	0x0000	Reserved	reserviert, nicht ändern
1EHEX	15:0	0x0000	Cust_ID2	Customer-ID Byte 2
1FHEX	15:0	0x0000	Cust_ID3	Customer-ID Byte 3

des Arduino angeschlossen wird. Im Sketch sind zwei Konstanten definiert:

1. `CURRENT_I2C_ADDRESS = 0x28` (Werkseinstellung!)
2. `NEW_I2C_ADDRESS = 0x22` (oder ein anderer eindeutiger Wert zwischen 0x00 und 0x7f)

Der Sketch ändert die Adresse durch einfaches Beschreiben der EEPROM-Speicherstelle 0x1C, das benutzerdefinierte Konfigurationsregister des CC2D.

Beachten Sie, dass wir in ein 16-Bit-Register schreiben und die I²C-Adresse in den unteren sieben Bits steht. Alle höheren Bits sind ab Werk auf null eingestellt, die meisten beziehen sich auf besondere, vom Benutzer konfigurierbare Einstellungen der Alarmanschlüsse des Sensors. Diese Einstellungen werden vom Sketch auf die Werkseinstellungen zurückgesetzt! Es ist sicherlich eine gute Idee, einen kleinen Aufkleber mit der neuen I²C-Adresse auf die Rückseite des CC2-eBoB zu kleben!

(140154)

Weblinks

[1] Kellerventilation: www.elektor-labs.com/project/basement-ventilation-system-control-unit-140432.14275.html

[2] ChipCap2 Application Guide: www.digikey.com/document-redirector?doc=http://www.digikey.com/Web%20Export/Supplier%20Content/GESensing_45/PDF/ge-sensing-chipcap2-application-guide.pdf

[3] Arduino & Python Bibliotheken: https://github.com/circuitsforfun/ChipCap2

[4] Projekt-Supportseite: www.elektor-magazine.com/140154

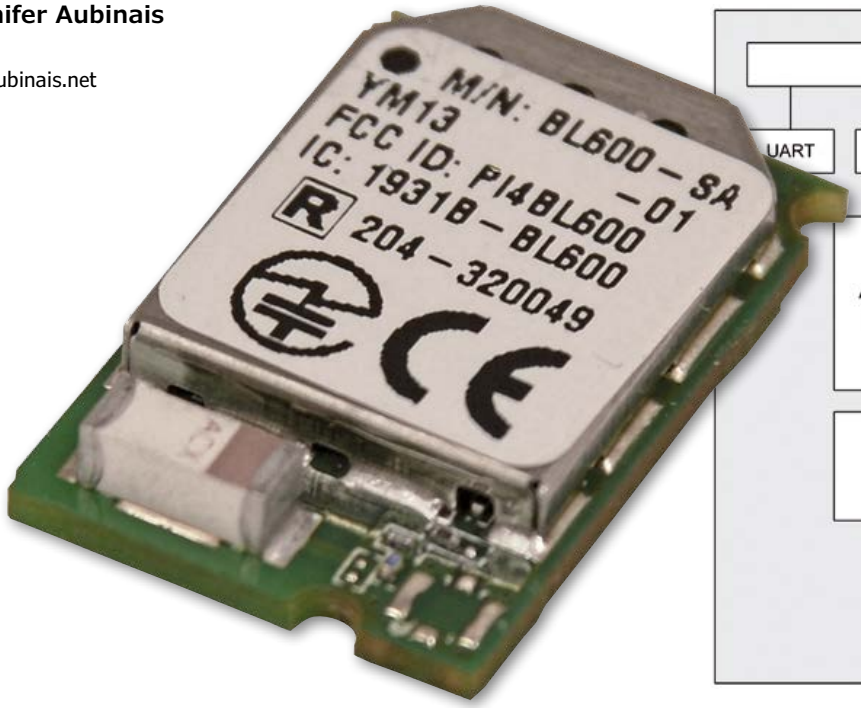
Tabelle 4. Zuweisung der Cust_Config-Bits

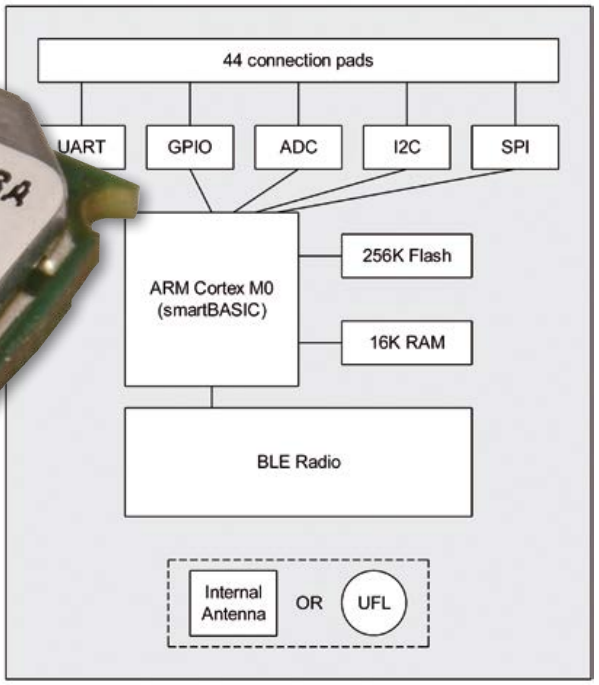
Bitbereich	Voreinstellung	Name	Beschreibung	
6:0	0101000	Device_ID	I²C-Slaveadresse	
8:7	00	Alarm_Low_Cfg	Konfiguration des Ausgangspins Alarm_Low:	
			Bits	**Beschreibung**
			7	Alarm Polarität 0 = Aktiv High 1 = Aktiv Low
			8	Ausgangskonfiguration 0 = Full Push-Pull 1 = Open Drain
10:9	00	Alarm_High_Cfg	Konfiguration des Ausgangspins Alarm_High:	
			Bits	**Beschreibung**
			9	Alarm Polarität 0 = Aktiv High 1 = Aktiv Low
			10	Ausgangskonfiguration 0 = Full Push-Pull 1 = Open Drain
12	0	Ready_Open_Drain	Ready-Pin ist: 0 = Full Push-Pull 1 = Open Drain	
13	0	Fast_Startup	Setzt die Länge des Befehlsfensters: 0 = 10 ms Befehlsfenster 1 = 3 ms Befehlsfenster	
15:14	00	Reserved	**Nicht ändern!** – Werkseinstellungen	

Drahtloses Thermometer mit Bluetooth

Fernanzeige der Temperatur auf Ihrem Smartphone

Von **Jennifer Aubinais**
(Paris)
elektor@aubinais.net





Dieses Außenthermometer besitzt ein wasserdichtes Gehäuse und kommuniziert über *Bluetooth Low Energy 4.0* (BLE) mit einem aktuellen Smartphone: iPhone 4S (oder höher), Android 4.3 (oder höher). Es nutzt das Modul BL600-SA von *Laird Technologies*, das entwickelt wurde, um drahtlose Kommunikation einfach verfügbar zu machen. Es gibt kein weiteres aktives Bauteil, alles ist in diesem Mini-Modul integriert, welches programmiert wird in… BASIC !

Ich bewundere Bluetooth und dieses Thermometer liefert mir einen guten Grund, die neueste Version dieses Standards einzusetzen. Diese ist besonders interessant, weil die Stromaufnahme deutlich geringer ist als bei den Vorgängerversionen (2.0 und 1.0), die aber nicht mehr kompatibel mit der neuen Norm sind.

Bluetooth und BASIC

Die Programmiersprache *smartBASIC* (ereignisgesteuert) des BL600-SA vereinfacht die Anwendung von Bluetooth nicht nur in Hinsicht auf die Steuerung direkt an das Modul angeschlossener Sensoren, es lassen sich auch die Messdaten direkt an jedweden Bluetooth-Empfänger senden

Dank an: Philippe und den Support von Laird Technologies

(Smartphones oder Tablets, PC, Interfaces…). Es ist schon fast ein Kinderspiel nunmehr mittels Radiowellen mit kleinen portablen Geräten zu kommunizieren, die nur durch AAA- bzw. Knopfzellen versorgt werden.
Das *A* in der Typenbezeichnung des Moduls steht für die integrierte Antenne.

Das Modul BL600-SA besticht durch seine geringe Stromaufnahme, es basiert auf der Chipserie nRFS1822 von *Nordic Semiconductor* und vereint alle Hard- und Software, die man für eine drahtlose Kommunikation mit 1 Mbit/s im 2,402 .. 2,480-GHz-Band braucht. Die wesentlichen Leistungsdaten sind:

- Interfaces UART, I²C, SPI
- 28 Allzweck-Ein-/Ausgänge (GPIO)
- 6 Analogeingänge (ADC mit 10 bits)
- Stromaufnahme :
  - 4 µA im Sleep-Modus
  - 5 µA in Bereitschaft
  - 10 mA während einer Übertragung
- einfache Programmierung in *smartBASIC*
- geringe Abmessungen : 19 x 12,5 x 3 mm
- Reichweite im Freien : bis zu 20 m

Beeindruckend, nicht wahr? Bemerkenswert ist außerdem der kleine Preis. Das Foto (**Bild 1**) zeigt das Modul auf der vom Hersteller angebotenen Leiterkarte (SDK). Man muss schon genau hinsehen, um es zu entdecken… Es scheint, als hätte der Hersteller vornehmlich medizinische Einsatzzwecke im Auge (Blutdruck, Herzrhythmus, Körpertemperatur…), aber es liegt nur an uns, das Einsatzgebiet zu erweitern.

Beim ersten Versuch habe ich keine Stunde gebraucht, um alle Nullen und Einsen in den BL600 zu schicken und eine LED zum Blinken zu bringen. Nach diesem ermutigenden Auftakt habe ich keine weiteren Stunden mehr gezählt, denn wer liebt, der zählt nicht. Schon gar nicht die zahlreichen Seiten der Dokumentation oder die auf der Webseite des Herstellers verbrachten Stunden [4]. Dann wäre da noch die häufig gestellte Frage nach der Miniaturisierung: die Packungsdichte des Moduls ist so hoch, dass es schwierig aber nicht unmöglich ist, es mit der Hand zu löten (es ist richtige Feinarbeit). Glücklicherweise hat *Laird Technologies* einen einfachen Kniff vorbereitet, um das Modul präzise einzupassen, ich komme später darauf zurück.

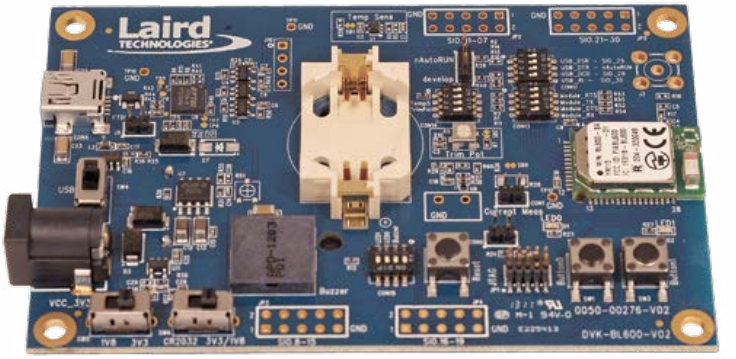


Bild 1.
BL600-Eval-Platine, das weiße Rechteck am rechten Rand ist das Modul.

Blinken
Mit diesem ersten - sehr einfachen - Programm konnte ich feststellen, dass das Modul bei nicht leuchtender LED nur 5 µA verbraucht.

Man kann auch die Bluetooth-Funktion des Moduls von Zeit zu Zeit abschalten, um sparsam mit der Spannungsversorgung umzugehen.

```
'//*************************************
'// Jennifer AUBINAIS 2014
'// Test sleep with led
'//*************************************

'//*************************************
' LONG TIME : 2 secondes => led off
'//*************************************

FUNCTION Func0()
Dim rc
' led off
rc = GPIOSetFunc(17,2,0)
TIMERSTART(1,2000,0)
ENDFUNC 1

'//*************************************
' SHORT TIME : 100 ms => led on
'//*************************************
FUNCTION Func1()
Dim rc
' led on
rc = GPIOSetFunc(17,2,1)
TIMERSTART(0,100,0)
ENDFUNC 1
```

```
'//*************************************
' event
'//*************************************
ONEVENT EVTMR0 CALL Func0
ONEVENT EVTMR1 CALL Func1

'*************************************
' main program
'*************************************
Dim rc
' output GPIO 17 and led on
rc = GPIOSetFunc(17,2,1)
TIMERSTART(0,100,0)
' SLEEP : close uart
uartclose()
rc = GPIOSetFunc(21,2,1) ,// TX
rc = GPIOSetFunc(23,2,0) ,// RTS
WAITEVENT
```

Schaltbild

Das Schaltbild ist rasch besprochen. Zum Messen der Temperatur setze ich den ADC des BL600-SA (MOD1 in der Schaltung in **Bild 2**) mit einem NTC an K3 ein. Dieser kann auch abgesetzt montiert werden, die Leitungslänge ist unkritisch. Die Messung wird über einen Spannungsteiler mit einem bekannten Widerstand R1 = 10 kΩ (0,1 %) durchgeführt. Die Gleichung zur Temperaturberechnung in Abhängigkeit vom NTC ist logarithmisch und übersteigt die mathematischen Fähigkeiten des BL600. Doch diese Berechnung, der eine Wertetabelle vom Hersteller des NTCs zugrunde liegt (Temperaturen, Widerstände, thermischer Koeffizient Alpha), kann mittels einer iOS- oder Android-Applikation auf dem Smartphone ablaufen. Das wird unten behandelt.

Der Spannungsteiler liegt an Pin SIO_19 des BL600. Die Stromaufnahme im Betrieb ist 5 µA

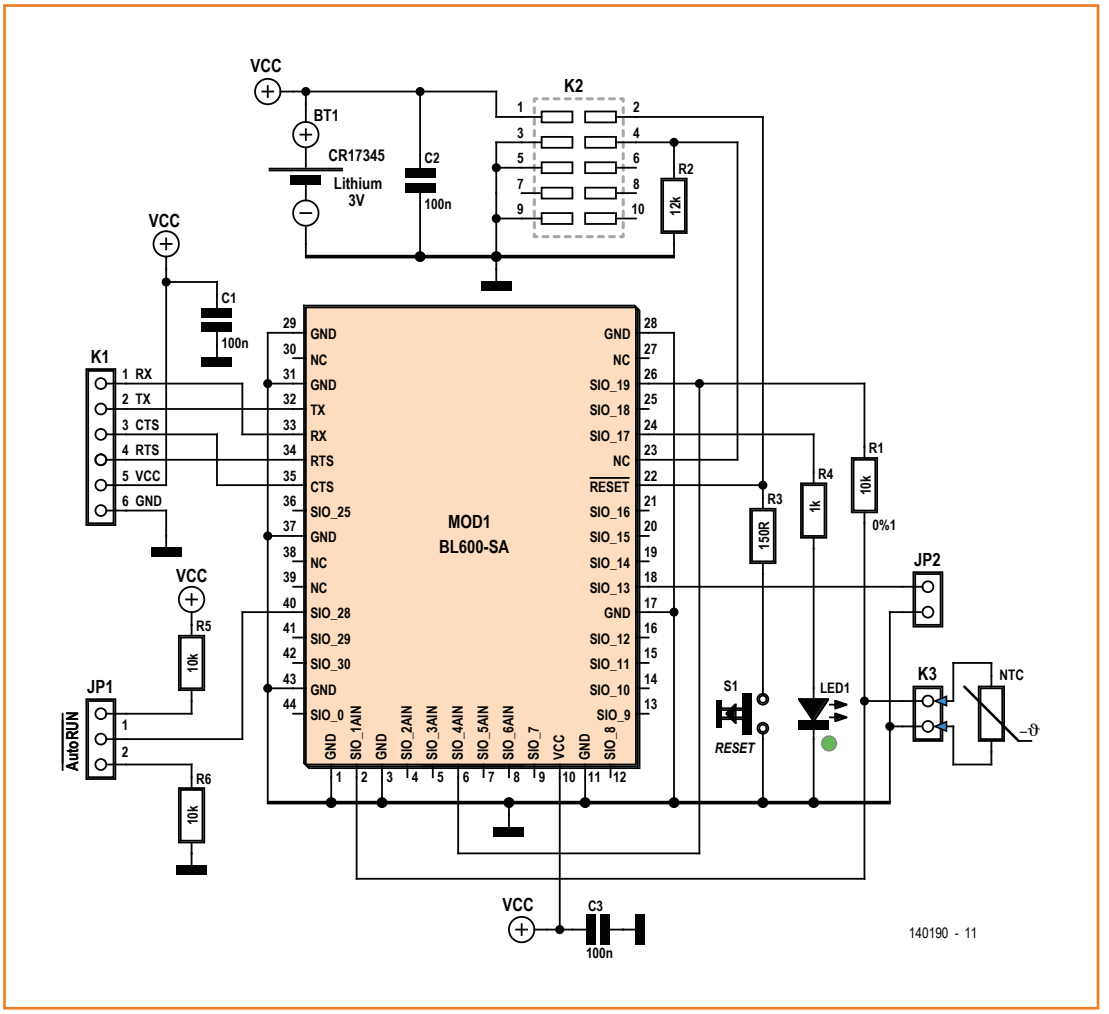


Bild 2.
Schaltbild des BT-Thermometers mit dem BL600-SA. Der BL600 kennt nur den LE-Modus (*low energy*), wohingegen der BT900 klassisches BT und BLE versteht, aber er ist nochmals kleiner und schwieriger zu löten!

(Bereitschaftsmodus). Im Sleep-Modus beträgt sie nur 0,4 µA, aber zurück in den Bereitschaftsmodus wechseln kann das Modul nur durch Reset oder bei Zustandsänderung an einem Eingang. In Bereitschaft wird die Bluetooth-Kommunikation periodisch durch Interrupts des Programms kurzzeitig wiederhergestellt, um Anmeldeversuche festzustellen. Ich habe ganz willkürlich 500 ms festgelegt (Parameter in der Funktion `TIMER-START`). Sobald die Verbindung hergestellt ist, steigt die Stromaufnahme auf 10 mA.

Verbinder K2 dient zum etwaigen Software-Update des BL600 über ein spezifisches JTAG-Interface. Im Prinzip ist diese Funktion für das vorliegende Projekt überflüssig (… aber man weiß ja nie). Verbinder K1 dient zum Programmieren des BL600 vom PC aus über ein serielles 3,3-V-Interface. Sehr gut geeignet ist hierfür der BOB FT232R [6], den Elektor in seinem Internetshop anbietet.

Die LED hat zwei Aufgaben:

- *Debugging* (mit Jumper JP2): sie blinkt die ganze Zeit und zeigt die Betriebszustände Bereitschaft und Bluetooth an
- *Normal* (ohne Jumper): sie blinkt kurz bei der Initialisierung auf, um den erfolgten Start des Programms anzuzeigen

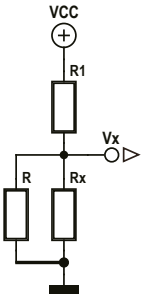
Bei meinen ersten Versuchen kamen die Daten nur verstümmelt an; jetzt hatte ich die Wahl, das Modul länger in Bereitschaft zu lassen oder weniger Daten zu übertragen. Wegen des Batteriemanagements habe ich mich für letzteres entschieden. Es folgt ein Beispiel für ein gesendetes Datenpaket:

PW3012V853C433
PW : Batteriespannung zur Anzeige durch die App
V : Eingangsspannung des Spannungsteilers
C : Abgriffsspannung des Teilers an Punkt R1 / NTC

Dies entspricht einer Temperatur von 24,3 °C (vgl. den folgenden Absatz **Berechnungen**).

Berechnungen

Hier sei noch einmal kurz erwähnt, dass die Berechnungen des NTC-Wertes auf einer Messung an einem Spannungsteiler beruhen



$$V_x = V_{cc} \times \left(\frac{\frac{RR_x}{R+R_x}}{R1+\frac{RR_x}{R+R_x}} \right)$$

$$\vdots$$

$$R_x = \frac{V_x R1 R}{V_{cc} R - V_x (R1+R)}$$

Ich beschränke mich für die Berechnung der Temperatur nicht auf eine einfache Gleichung zwischen Thermistor und Temperatur unter Verwendung eines Beta-Koeffizienten, wie es üblicherweise geschieht :

$$R_{ctn} = R_{25} \times e^{\beta \times \left(\frac{1}{T} - \frac{1}{298}\right)}$$

R_{25} : Widerstandswert bei 25 °C
T : berechnete Temperatur in K
beta : Beta-Koeffizient des NTCs
R_{ctn} : Wert des NTCs bei der zu berechnenden Temperatur

Tabelle 1.

R/T No	4901	
T (ºC)	$B_{25/100}$ = 3950 K	
	RT/R25	**α (%/K)**
–30,0	16,915	6,1
–25,0	12,555	5,9
–20,0	9,4143	5,7
–15,0	7,1172	5,5
–10,0	5,4308	5,4
–5,0	4,1505	5,2
0,0	3,2014	5,0
5,0	2,5011	4,9
10,0	1,9691	4,7
15,0	1,5618	4,6
20,0	1,2474	4,5
25,0	1,0000	4,3
30,0	0,808	4,2
35,0	0,6569	4,1
40,0	0,5372	4,0

Ich verwende stattdessen einen Koeffizienten alpha α (%/K), der sich wie vom Hersteller festgelegt mit der Temperatur ändert (**Tabelle 1**), das führt zu folgender Gleichung:

$$R_T = R_{Tx} \cdot e^{\left[\frac{\alpha_x}{100}\cdot(Tx)^2\cdot\left(\frac{1}{T}-\frac{1}{Tx}\right)\right]}$$

R_T : Widerstand bei Temperatur T
R_{Tx} : Widerstand bei Temperatur Tx lt. Tabelle
T_x : Temperatur in K unterhalb des gefundenen Tabellenwerts zur vorliegenden Messung
T : Temperatur zur vorliegenden Messung in K ($T_x < T < T_{x+1}$)
α_x : thermischer Koeffizient bei Temperatur T_x

Ausgehend von dieser Gleichung gestaltet sich die Temperaturberechnung unter Verwendung von α_x wie folgt:

$$T = \frac{1}{\frac{100}{\alpha_x\cdot(Tx)^2}\cdot ln\left(\frac{R_T}{R_{Tx}}\right)+\frac{1}{Tx}}$$

Schließlich für diese letzte Gleichung ein Berechnungsbeispiel mit folgenden Parametern:

R_T = 10506,46 Ω (R_{CTN})
=> 12474 Ω > R_T/T_{25} > 10000 Ω
=> α_x= 4,5
=> T_x = 20 °C = 293,15 K
=> R_{Tx} = 12475 Ω

$$T = \frac{1}{\frac{100}{4{,}5\cdot(293{,}15)^2}\cdot ln\left(\frac{10506{,}46}{12474}\right)+\frac{1}{293{,}15}}$$

$$T = 297{,}01\,K$$

$$T = 297{,}01 - 273{,}15 = 23{,}86\,°C$$

Besitzt R1 eine Toleranz von 0,1 % ergibt sich eine Genauigkeit von 2 % oder besser. Sofern man sich mit 3 % Genauigkeit zufrieden gibt, kann R1 ein gewöhnlicher (und preiswerterer) Widerstand mit 1 % Toleranz sein.

Praktischer Aufbau

Erst nachdem ich ein passendes Gehäuse gefunden hatte, habe ich mich um die Platine (**Bild 3**) gekümmert, um einen Temperatursensor aufzubauen, der wartungsfrei ist für - sagen wir mal - die nächsten zehn Jahre! Die Eagle-Bibliothek für den BL600 habe ich selbst erstellt (verfügbar im Download-Bereich zu diesem Artikel). Das Gehäuse besteht aus ABS, ist sehr robust und wasserdicht, aber der innere Aufbau erfordert einen unregelmäßigen Zuschnitt der Platine.
Ich hatte zunächst Bedenken wegen der Stromaufnahme des neuen Moduls. Deshalb wird es nicht mit einer CR2032-Knopfzelle versorgt, sondern mit einer CR123. Ich habe für den Halter einen Adapter (kleines Bild) entworfen, der einfach nur mit Präzisionskontakten gesteckt wird und somit abnehmbar ist. Die Präzisionskontakte gibt es in Reihen zu kaufen, wir benötigen weiblich und männlich jeweils acht Pins. Die acht

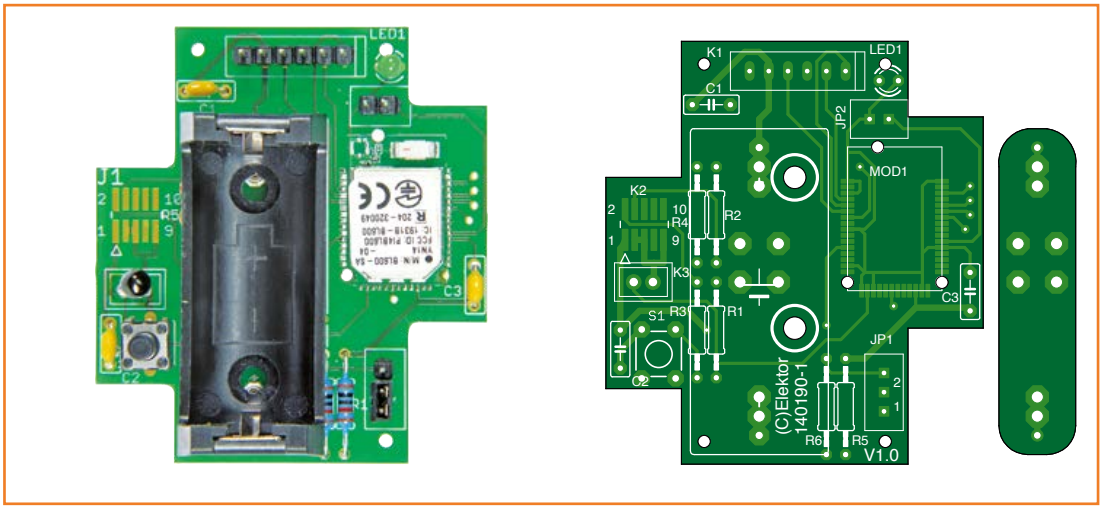
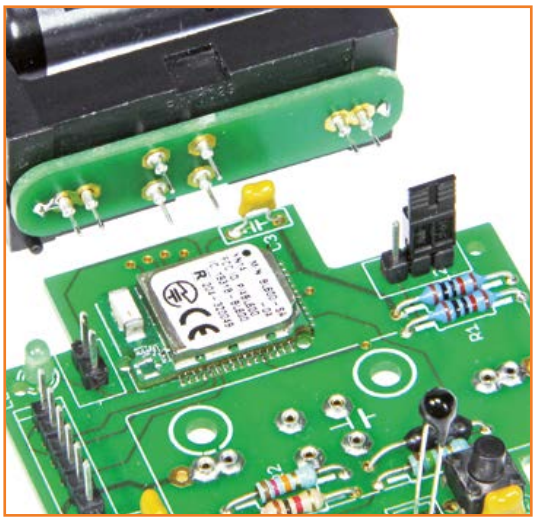


Bild 3.
Für diejenigen, die filigrane Arbeiten nicht so sehr mögen, bietet Elektor eine Platine mit bestücktem BL600-SA an. Es fehlen dann nur noch einige bedrahtete Bauteile.

Anwendungsbeispiel für das Bluetooth-Modul BL600-SA



Stifte löten wir in die längliche Batteriehalter-Platine und danach den CR123-Halter ein (Polarität beachten!).
Das Bluetooth-Thermometer wird von Elektor als teilbestücktes Modul angeboten, der BL600 ist **bereits eingelötet** [3]. Wer es selbst machen möchte fertige sich zunächst eine Matrize zum Auftragen der Lötpaste an. Drei 1,6-mm-Schrauben, eine in jedem der dafür vorgesehenen Löcher, fixieren die Matrize vor dem Auftragen der Paste auf die Platine perfekt. Stecken Sie die Schrauben von unten ein, um das Modul auf diese Weise präzise zu führen. Setzen Sie dann den BL600 auf, der, geführt von den Schlitzen, an den Schrauben entlang gleitet und auf den Zehntelmillimeter genau positioniert wird. Hierzu empfehle ich einen Videoclip [5].
Danach muss die Platine nur noch in den Ofen. Nehmen Sie nun acht weibliche Präzisionskontakte, löten Sie sie in die Platine und schneiden Sie die vorstehenden Spitzen ab, so dass nur die Buchsen verbleiben: diese nehmen die Zusatzplatine mit dem Batteriehalter auf.
Die restlichen Bauteile sind sämtlich bedrahtet und können direkt eingelötet werden. R2 und Verbinder K2 werden nur für Software-Updates benötigt und können entfallen, wenn dies nicht vorgesehen ist.
Je nach gewünschtem Einsatzzweck kann der Batterieadapter wie hier in demselben Gehäuse oder auch abgesetzt in einem Gehäuse nach Wahl montiert werden.

Mit Jumper JP1 wählt man zwischen zwei Modes:

- *autorun* (Position 1) startet das Programm automatisch bei Anlegen der Betriebsspannung oder nach einem Reset.
- *Befehl* (Position 2) gibt die serielle Verbindung frei und akzeptiert AT-Befehle wie z.B. „AT&f 1“, wodurch der Programmspeicher gelöscht, der BL600 neu gestartet wird und ein Programm über die serielle Verbindung geladen werden kann.

Stückliste

Widerstände :
R1 = 10 kΩ 0,1%*
R2 = 12 kΩ*
R3 = 150 Ω
R4 = 1 kΩ
R5, R6 = 10 kΩ
Thermistor NTC B57891S103F8 10 kΩ (2112816)

Kondensatoren :
C1, C2, C3 = 100 nF (RM 5,08 mm)

Halbleiter :
LED1 = LED 3 mm
MOD1 = Modul BL600-SA *Laird Technologies*

Außerdem :
JP1 = Stiftleiste, 3-polig
JP2 = Stiftleiste, 2-polig, z.B. *Molex* (9731148)
2 Jumper
K1 = Stiftleiste, 3-polig
S1 = Mikrotaster, Printmontage (1555985)
Halter für CR123A* (1650670)
16 Präzisionskontakte
Gehäuse *Multicomp* G302 (1094697)
Platine (unbestückt) 140190-1

Bluetooth-Thermometer 4.0 bestehend aus Hauptplatine, teilbestückt mit dem Modul BL600-SA, Elektor-Nr.: 140190-91

* = s. Text
(Farnell-Best.-Nr. in Klammern)

Die Autorin über sich selbst

Mein Großvater, der alles Mögliche gesammelt hat, hat mich für die Elektronik begeistert. Dank ihm hatte ich noch nie Angst, mir die Finger zu verbrennen. Mein Werdegang beginnt im Gymnasium mit dem Heimcomputer TO7 und schließt ab mit dem Fachhochschulabschluss (Bestnote im Fach Elektronik) und 1992 mit dem Ingenieur (im Fach Informatik). Als Ingenieurin für Großrechnersysteme, für den Windows-Support und als Entwicklerin von VBA- und VB.NET-Anwendungen habe ich von allem etwas gemacht - außer Linux und Rechnernetzen.
Etwa im Jahre 2012 bekam ich wieder Sehnsucht nach heißem Lötzinn; so kehrte ich zur Elektronik zurück - als Einzelentwicklerin hauptsächlich für Elektor-Projekte. Ich habe mich auf Bluetooth-Interfaces und Wifi für iOS- und Android-Smartphones spezialisiert.

Jumper JP2 setzt den Debug-Modus, dann werden die Programmmeldungen über den seriellen Port ausgegeben. Im Normalbetrieb bleibt er frei.

Programm des BL600

Dank *smartBASIC* von *Laird Technologies* ist die Programmierung des Moduls ausgesprochen einfach. Es ist Basic – wie damals in der Schule. Mit grundlegenden arithmetischen Funktionen um die Verarbeitung der gewonnenen Daten zu vereinfachen. Es erlaubt Unterprogramme und Funktionen, es verwaltet die Ein- und Ausgänge des BL600 sowie die komplexeren Schnittstellen wie I²C, SPI, CAN und UART. Damit ist es auch für Anfänger geeignet. Das Programm kann mit jedem Texteditor geschrieben werden. Ich empfehle Notepad+, Download von der Webseite von Laird Technologies. Das Programm wird kompiliert und übertragen durch die Gratis-Software *UWTerminal*.

Auf der Elektor-Seite zu diesem Artikel [2] findet sich der Quellcode meines Programms `JATEMP`. Dieses zeigt eine Verbindung zwischen Modul und Smartphone an, die Initialisierung des AD-Konverters und dann - nach etwa einer halben Sekunde - das Senden von Daten an das Smartphone: Versorgungsspannung, Speise- und Abgriffsspannung des Spannungsteilers. Ein Datenpaket wäre z.B.: PW3012V853C433. Die Berechnungen besorgt die App auf dem Smartphone.

Programme für iOS & Android

Laird Technologies empfiehlt den Download des Quellcodes der App `BL600 Serial` für iOS. Da ich jedoch mit iOS Erfahrungen habe, habe ich es vorgezogen, eine eigene App zu schreiben [7], welche zwei Informationen anzeigt: Temperatur und Status der Verbindung. Ich habe aber Teile des Quellcodes des Programms `BL600 Serial` (`UART-Peripherial.c` und `DataClass.c`) integriert. Buttons für Verbinden und Trennen fehlen. Diese werden durch eine Schleife (genauer : einen Thread) ersetzt, die das JATEMP-Thermometer sucht, wenn es nicht verbunden ist und sich dann verbindet (es ist das Thermometer, welches sich trennt).
Da ich keine Erfahrungen mit Java-Programmierung hatte, hat mir das Android-Programm einigen Kummer bereitet. Danke an die französische Seite developpez.com, wo ich Hilfe fand, besonders, wie man eine homogene Benutzeroberfläche für die verschieden großen Android-Smartphones erstellt. Ich habe auch unter Android meine eigene App *BL600 Serial* (Code und Interface) erstellt, aber mit neuem Namen und in einfacherer Form [8], nur zur Verbindung mit JATEMP gedacht. Ein Button *Scan and Connect* erlaubte anfangs den Empfang eines Roh-Datenpakets vom Thermometer; dieser wurde aber wie unter iOS später durch eine Schleife ersetzt.
Unter iOS und Android erfolgt die Temperaturberechnung wie oben beschrieben. Der Bildschirmhintergrund der App wechselt mit der gemessenen Temperatur (**Bild 4**). Um endlose Verbindungen mit dem Thermometer zu vermeiden endet die iOS-App nach 5 Minuten (oder bei Druck auf die Taste *Home).* Unter Android sucht die App nach 5 Minuten (oder bei *Home)* nicht weiter nach dem Thermometer, die App wird aber nicht tatsächlich angehalten.
Um die Verbindung mit dem Smartphone aufzubauen und das Thermometer das erste Mal zu verbinden muss **nichts** getan werden. Alles funktioniert automatisch.

Kühlschrank = Faradayscher Käfig

Sie denken sicherlich genau wie ich, dass ein Kühlschrank einen faradayschen Käfig darstellt, d.h. elektromagnetische Wellen wirksam abschirmt. Während meiner Versuche Anfang September 2014 hatten wir in Paris Spätsom-

mer. Die Temperaturen sanken nicht unter 20 °C. Also habe ich das Thermometer für eine Viertelstunde in den Kühlschrank gelegt, um es beim Herausholen auszulesen. Währenddessen habe ich an der Android-App weitergearbeitet und war sehr überrascht, dass auf dem Android-Smartphone eine rasch sinkende Temperatur angezeigt wurde! Danach habe ich denselben Versuch mit dem Kühlschrank meiner Nachbarin durchgeführt – und siehe da, auch hier fiel die Temperatur-anzeige direkt bis auf 4 °C. Ich folgere daraus, dass die umlaufende Türdichtung für das Bluetooth-Signal durchlässig ist (ein Umstand, auf den die Haushaltsgerätehersteller in ihren technischen Unterlagen hinweisen sollten ;-)).

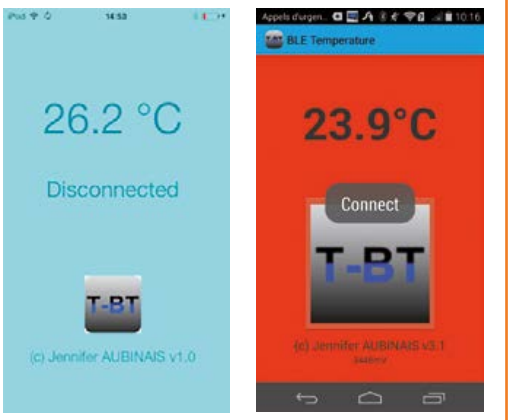


Bild 4.
Die Thermometer-App auf den Displays von iOS und Android.

Bild 5.
Der Prototyp der Autorin mit wasserdichtem Gehäuse und einem soliden, verschraubten Aufhänger.

Zusammenfassung

Ich hoffe, Sie von den Vorzügen des BL600 überzeugt zu haben - vielleicht trauen Sie sich ja auch an den Reflow-Ofen oder löten per Hand, feengleiche Finger vorausgesetzt. Die Anschlüsse eines BL600 sind winzig, dies wird aber bei Weitem aufgewogen durch die Vorteile des einfachen Starts, der wirklich simplen Programmiersprache *smartBASIC* und der Möglichkeit, das Modul über den UART zu programmieren. Und das ist erst der Anfang. Zwischenzeitlich hat *Laird Technologies* den BL620 angekündigt, einen *Master*-BL600: gleiche Hardware, neue Software, die mit anderen BL600-Modulen kommunizieren kann. Dieser eröffnet ganz neue Perspektiven, die Elektor und ich Ihnen gerne bald vorstellen möchten.

(140190)

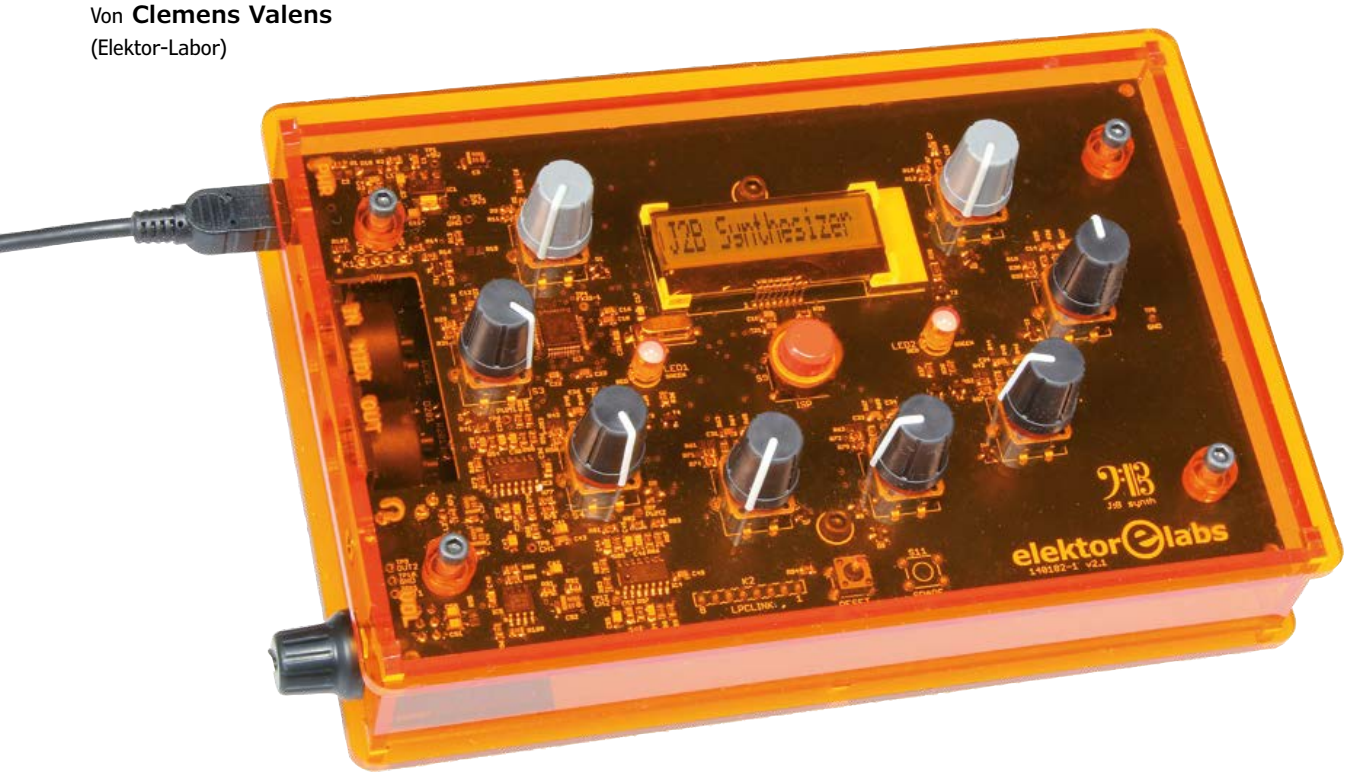
Weblinks

[1] www.lairdtech.com/Products/Embedded-Solutions/Bluetooth-Radio-Modules/BL600-Series/
[2] www.elektor-magazine.de/140190
[3] www.elektor.de/bluetooth-thermometer
[4] https://laird-ews-support.desk.com/?b_id=1945
[5] www.youtube.com/watch?v=0YlKxtYwQiE
[6] www.elektor.de/ft232r-usb-serial-bridge-bob-110553-91
[7] https://appsto.re/fr/XTwnV.i
[8] https://play.google.com/store/apps/details?id=com.JA.bletemperature&hl=fr
[9] www.aubinais.net
[10] CTN EPCOS
www.epcos.com/inf/50/db/ntc_13/NTC_Leaded_disks_S891.pdf
www.epcos.com/blob/531152/download/2/pdf-standardizedrt.pdf
NTC Thermistors / General technical information : www.physics.queensu.ca/~robbie/ENPH354/NTC-Thermistors-Technote.pdf

J²B-Synthesizer

Eine offene digitale Musikplattform

Von **Clemens Valens**
(Elektor-Labor)



Der Musik-Synthesizer Atmegatron von Soulsby Synthesizer [1] ist mit einem ATmega328 8-bit-AVR-Controller von Atmel aufgebaut, dem gleichen Mikrocontroller, der auch den Arduino Uno steuert. Der Atmegatron weist einige Aspekte auf, die mich sehr interessierten und schließlich zu diesem Projekt bewogen.

Bevor Sie in das Projekt eintauchen, lassen Sie mich eine Warnung an alle aussprechen, die hautsächlich an der Klangsynthese eines Synthesizers interessiert sind. Nachfolgend wird beschrieben, wie ich den Atmegatron in meine eigene Hardware-Plattform portiert habe. Die Sound-Engine selbst wird nur kurz und bündig beschrieben. Wenn dies Ihr einziges Interesse ist, werfen Sie

Eigenschaften

- Monophoner 9-bit-Synthesizer
- 32 feste und benutzerdefinierte Wellenformen
- 15 Filtertypen
- 2 Hüllkurven-Generatoren
- LFO mit 16 Wellenformen
- 15-Muster-Arpeggiator
- 16 Patch-Memories
- 6 Live-Steuerungen
- MIDI
- Patch sichern/laden über MIDI
- Mikrocontroller NXP LPC1347 32-bit-ARM-Cortex-M3
- 2 Ausgangskanäle
- Open-Source & Open-Hardware-Design

einen Blick auf Bild 2; danach sollten Sie den Source-Code in [2] studieren.

Atmegatron

Der Mikrocontroller im Atmegatron (**Bild 1**) kann im Prinzip alles, von der Erzeugung von Klängen bis zur Interaktion mit dem Anwender. Er kann sich sogar mit anderen Geräten über seinen MIDI-Anschluss unterhalten. All diese Funktionen passen in den 32 KB großen Programmspeicher, was, wie ich finde, schon eine Leistung ist, zumal die Software als Arduino-Sketch in C/C++ geschrieben wurde. Die Software wird unter einer Open-Source-Lizenz veröffentlicht und kann kostenlos heruntergeladen werden. Leider liegt die Hardware des Atmegatrons nicht offen (obwohl man sie ganz einfach aus dem Studium der Software rekonstruieren kann).

Ein weiteres interessantes Feature ist, dass der Atmegatron für Live-Auftritte konzipiert wurde. Es besitzt sechs Potentiometer zur Steuerung, um zehn Klangparameter im laufenden Betrieb ändern zu können. Zusammen mit einem Funktionswähler und einem Funktionswert-Encoder besitzt der Synthesizer acht Steuerungen. Ein Taster wählt zudem zwischen zwei Betriebsarten. Grün/rote LEDs zeigen den Modus und die Werte an, ein alphanumerisches Display ist nicht vorhanden.

Für alle, die Sound „mit Biss“ süßem und sanftem Gedudel vorziehen, ist es interessant zu wissen, dass Atmegatron verschiedene Algorithmen bietet, um hässliche und entstellte Sounds zu produzieren. Atmegatron sollte daher nicht mit den Synthies der ganz großen Hersteller verglichen werden, Atmegatron hat seinen eigenen, ganz speziellen Sound.

Der Synthesizer verwendet das Pulsweitenmodulationsmodul (PWM) des Controllers, um Töne zu erzeugen. Ein externer Digital/Analog-Wandler (DAC) kann daher auf die Funktion eines Anti-Aliasing-Filters beschränkt sein.

Eine Open-Source-Sound-Engine, die von sechs Potentiometern und zwei Drehgebern gesteuert wird und eine Software, die in einen 32-KB-Programmspeicher passt: das hat mich neugierig gemacht, ob ich so etwas auf das J²B-Controllerboard, das ich vor ein paar Jahren in Elektor vorgestellt hatte, portieren könne [3]. J²B wurde um den 32-bit-ARM-Controller Cortex-M3 LPC1343 von NXP aufgebaut. Wie der ATmega328 besitzt er 32 KB Programmspeicher und alle andere Peripherie, die vom Synthesizer verwendet wird, einschließlich der ausgezeichneten PWM-Funktionen. Außerdem unterstützt das J²B-Board bis zu neun Drehencoder (oder acht und einen Druckknopf). Was allerdings fehlt, ist ein EEPROM, um die Sound-Presets zu speichern, aber es gibt Möglichkeiten, diese Klippe zu umschiffen.

Der Atmegatron verwendet zweifarbige LEDs, um Werte, Positionen und Modi ohne alphanumerisches Display anzuzeigen. Das J²B-Board verfügt aber über ein LCD, man kann sogar unter drei Größen wählen. So habe ich beschlossen, die LEDs durch ein LCD zu ersetzen.

Die Portierung lief auf drei Hauptaufgaben hinaus:

1. Portierung des Arduino-Sketches für den ATmega328 zu einem Eclipse/LPCXpresso-Projekt mit dem LPC1343;
2. Ersetzen der sechs analogen Potis durch sechs digitale Drehencoder;
3. Ersetzen der zweifarbigen LEDs durch ein LCD.

Bild 1.
Der Atmegatron, der Ursprung dieses Projekts.

Aufgabe 1

Die Portierung eines Codes von einem Mikrocontroller zum anderen kann mehr oder weniger entmutigend sein, je nachdem, wie die ursprüngliche Software geschrieben wurde. Ein gut strukturiertes Projekt ist viel einfacher umzusetzen als Spaghetti-Code. Auch die Abstraktion der Hardware spielt eine Rolle. Code, der Funktionen aufruft, um Register und Peripherie zu ändern, ist viel einfacher zu portieren als wenn wir es mit einer direkten Interaktion mit der Hardware zu tun hätten. Darüber hinaus ist eine Datei, die all diese Funktionen gruppiert, einfacher zu handhaben als Code, der diese Funktionen über die gesamte Software verteilt. Zum Glück ist der Atmegatron-Code gut strukturiert, gut dokumentiert und sehr „portierfreundlich“, so dass ich einen großen Teil der Arbeit erledigen konnte, während ich einen Bruce-Willis-Film im Fernsehen verfolgte. Die meiste Arbeit ergab sich bei der Erstellung der Header-Dateien für das Eclipse/LPCXpresso-C-Projekt, Arduino-Sketches kommen nämlich ohne Header-Dateien aus. Typische Fallstricke in dieser Phase sind die inkonsequente Zuordnung der Datentypen in der Software und inkompatible Datentypen zwischen den Compilern (AVR GCC für Arduino und ARM GCC für Eclipse/LPCXpresso). Beide Probleme führen zu Fehlern wie Datenüberlauf (Variablen passen nicht mehr in den für sie reservierten Speicherbereich) und unbeabsichtigte Vorzeichenwechsel. Solche Probleme kann man durch die konsequent richtige Definition der Datentypen (mit deutlicher Angabe der Größe in bits und ob sie „signed“ oder „unsigned“ sind) vermeiden. Verwenden Sie beispielsweise `uint8_t` für einen 8-bit-Ganzzahl-Wert statt `unsigned char`. Verwenden Sie `int16_t` oder `int32_t` statt `int`, da die Größe des `int` meist plattformabhängig ist. Noch besser ist es, wenn Sie Datentypen verwenden, die auch zeigen, um welche Art von Daten es sich handelt. Erstellen Sie beispielsweise einen Datentyp `sample_t` und verwenden Sie ihn nur, um Sample-Werte zu speichern, und dies durchgängig.

Viele schwierige Situationen bei der Portierung entstehen durch die Unterschiede der Hardware. Timer sind relativ einfache Peripherieblöcke, die bei den meisten CPUs mehr oder weniger gleich verwendet werden. Sie müssen also nur wissen, wie der Timer funktionieren soll. Aber was, wenn die Zielplattform nicht über die Funktion verfügt, die die Quellplattform einsetzt? Oder wenn die Peripherie sich ein wenig unterschiedlich verhält? Dies war beispielsweise beim PWM-Modul des LPC1343 der Fall, das seine Arbeit nicht ganz genau so erledigte wie das PWM-Modul des ATmega328, so dass ein verzerrter Klang die Folge war. Lassen Sie es mich erklären!

Beim AVR vergleicht das PWM-Modul ständig einen Zähler mit einem Schwellwert, setzt seinen Ausgang entsprechend und bestimmt so das Tastverhältnis des PWM-Signals. Wenn der Zählerwert gleich oder höher als der Schwellwert ist, ist der PWM-Ausgang Low (oder High, je nach Konfiguration), wenn der Zählerwert dagegen unter dem Schwellwert liegt, ist der PWM-Ausgang High. Wenn man im zweiten Fall aber den Schwellwert unter den Zähler bewegt, wird der PWM-Ausgang unmittelbar Low. Ein C-artiger Pseudo-Code sähe so aus:

```
counter = counter + 1
if (counter == max) then counter = 0
if (counter >= threshold) then PWM = 0
else PWM = 1
```

Beim LPC1343 der Mechanismus ist nahezu identisch. Nahezu. Anstelle der ständigen Kontrolle des Zählerstands ändert dieser Controller den PWM-Ausgang nur dann, wenn Zähler und Schwellwert exakt den gleichen Wert aufweisen (ein „match“). Eine Änderung der Schwelle hat daher keine unmittelbare Wirkung, der Ausgang wechselt nur, wenn der Zähler den neuen Schwellwert erreicht. Auch dies in Pseudocode:

```
counter = counter + 1
if (counter == max) then
  {
    counter = 0
    PWM = 1
  }
if (counter == threshold) then PWM = 0
```

Um die Auswirkungen zu verstehen, müssen wir einen genaueren Blick auf die Sound-Engine des Atmegatron werfen.

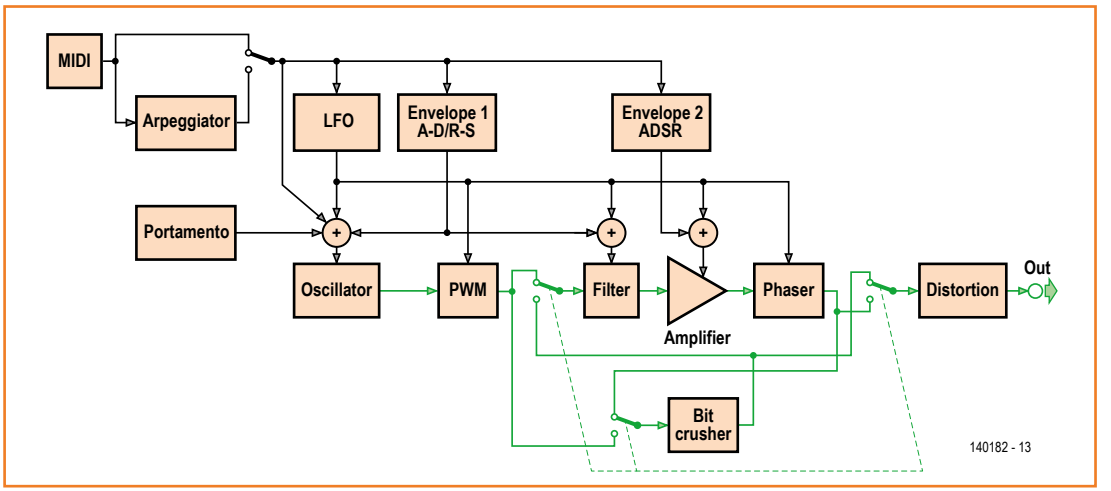


Bild 2.
Die Sound-Engine des Synthesizers, eingefangen in einem Blockdiagramm. Die Blöcke PWM, Bit-Crusher und Distortion sind für die ursprünglichen, rauen Töne verantwortlich.

Über die Sound-Engine

Die Sound-Engine (**Bild 2**) ist für die Ausgabe von Sounds verantwortlich. In unserem Fall berechnet sie den Ausgangssound in Blöcken á 32 Samples. Ein solcher Block entspricht immer einer Periode der Kurvenform am Ausgang (was bedeutet, dass die Abtastrate nicht konstant ist). Die Berechnungen beginnen mit einer Wavetable, die 32 Samples einer Periode einer Kurvenform enthält (es sind 32 Wellenformen vorgefertigt, aber Sie können auch Ihre eigenen definieren). Diese Samples werden gefiltert, auf verschiedene Weise berarbeitet und dann in einem Ausgangspuffer gespeichert. Dieser Puffer wird so schnell, wie es die Hauptschleife des Arduino-Programms erlaubt, aktualisiert, was bedeutet, dass dies eigentlich wie ein Task im Hintergrund läuft, ohne eine wirkliche Priorität. Die Aktualisierung wird halt durchgeführt, wenn das Programm an dieser Stelle ankommt. Sich dynamisch ändernde Parameter wie Lautstärke-Hüllkurven und Modulationssignale werden von einem Millisekunden-Timer synchronisiert, so dass sie von der Ausführungsgeschwindigkeit der langsamen Hauptschleife unabhängig sind.

Die Tonhöhe wird von einem Timer bestimmt, dessen Frequenz 32 Mal höher ist als der Ton selbst, um die Größe der Wavetable mit ihren 32 Samples auszugleichen. Wenn der Timer seinen von der Tonhöhe abhängigen Endwert erreicht hat, wird das nächste Sample vom Ausgangspuffer erfasst und die PWM-Schwelle (also das Tastverhältnis) gemäß der neuen Abtastung neu festgesetzt. So, wie der AVR arbeitet, folgt das Tastverhältnis den Änderungen der Samples nahezu ohne Verzögerung.

Beim LPC1343 wird das Tastverhältnis nur erneuert, wenn der PWM-Timer den neuen Schwellwert erreicht, was bis zu einer PWM-Periode dauern kann (wenn der neue Schwellwert genau unter den aktuellen Zählerwert gesetzt wird). Diese variable Verzögerung produziert hörbare Störungen und weiches rhythmisches Klicken.

Die Lösung des Problems war es, Interrupts beim PWM-Modul des LPC einzusetzen. Obwohl das PWM-Signal eine Frequenz von 140,625 kHz aufweist, kann der Controller dabei problemlos mit Interrupts umgehen. Der Interrupt-Routine des Pitch-Timers geht durch den Sample-Ausgangspuffer wie zuvor, doch anstatt den PWM-Schwellwert direkt dem neuen Abtastwert anzupassen, wird eine Variable als Zwischenspeicher verwendet. Die PWM-Interrupt-Routine liest diese Zwischengröße und benutzt sie, um das PWM-Tastverhältnis zu aktualisieren. Nun wird der PWM-Update ordentlich auf das PWM-Signal synchronisiert und die Klicks sind eliminiert.

Zu wenig Speicher

Als meine Portierung beinahe erledigt war, stellte sich ein anderes Problem ein: zu wenig Speicher! Das war überraschend, da doch beide Controller die gleiche Menge an Programmspeicherplatz (32 KB) zur Verfügung stellen. Zudem hatte ich erwartet, dass der LPC-Code effizienter ist, weil er als so genannter Thumb-Code kompiliert wird, extra um Programmspeicherplatz zu sparen. Ist der Arduino-Compiler denn wirklich so gut? Oder der AVR so codeeffizient? Wie auch immer, schuld am Übergewicht meiner Portierung war zum Teil die Bibliothek für die Hardware-Abstractions von LPCXpresso für den LPC1343 und zum Teil die in

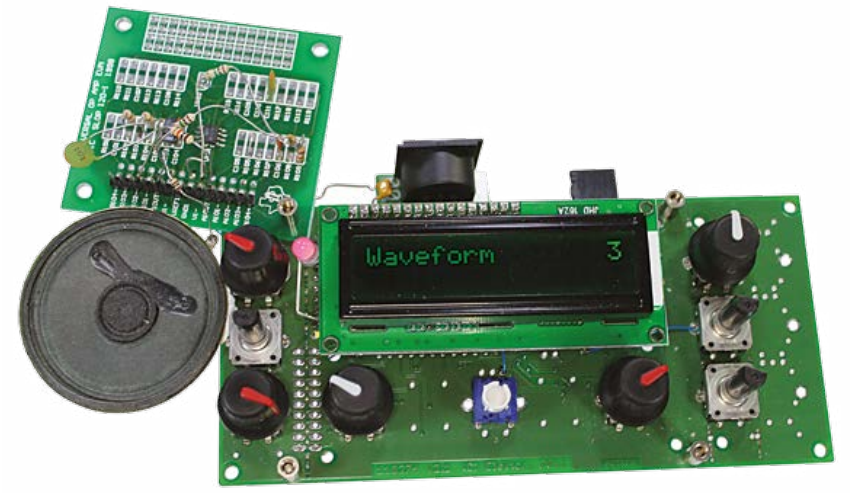


Bild 3.
Der J²B-basierte Prototyp mit seinen acht Drehencodern und dem 2x16-LC-Display.

einem LPCXpresso-Projekt standardmäßig eingebaute newlib-Bibliothek. Abgesehen davon, dass dies die Standard-C-Bibliothek ist, bietet sie auch printf-artige Debug-Unterstützung, auch im Release-Modus. Diese Debug-Unterstützung kann man entfernen, wenn sie nicht nötig ist. Die Option ist schwer zu finden, aber die Mühe lohnt sich: Gehen Sie zu den „Properties“ dieses Projekts und öffnen Sie „C/C++ Build“, gefolgt von „Settings“. Wählen Sie dann in der Liste der „Tool Settings“ das „Managed Linker Script“, um Zugang zur Bibliothek zu erhalten (Sie müssen „MCU Linker“ expandieren). Nach ein bisschen Experimentieren fand ich, dass die Bibliothek „Redlib (none)“ gute Ergebnisse erzielte, genau so gut wie jede andere „(none)“-Bibliothek. Dieser Trick befreit viel Speicherplatz, aber ich musste immer noch die EEPROM-Unterstützung und einen Teil der Benutzerschnittstelle implementieren.

Aufgabe 2

Das Ersetzen von Potentiometern durch Drehencoder mag trivial erscheinen, ist es aber nicht. Ein Poti, das man mit einer schnellen Fingerbewegung von Min nach Max dreht, fühlt sich ganz anders an als ein Drehencoder. Im Gegensatz dazu ermöglichen Drehencoder eine sehr präzise Steuerung der Parameter, aber es sind viele Umdrehungen nötig, um den gesamten Parameterbereich abzudecken. Man muss tricksen, irgendeine Art von Beschleunigungsmechanismus einsetzen, damit man wieder ein Poti „fühlt“, ohne dabei jedoch auf die Präzision des Drehencoders verzichten zu müssen.

Obwohl ich mich eher zu den „Trial-and-error“-Menschen zähle, bin ich an das Problem „wissenschaftlich“ herangegangen. Ich habe den von mir gewählten Drehgeber (24 Takte pro Umdrehung) an mein Oszilloskop angeschlossen und Drehgeschwindigkeiten ausgemessen: Wenn man den Encoder so schnell wie möglich dreht, erzielt man eine Impulsrate von etwa 100 Hz. Mein Ziel war es aber, den gesamten Bereich von 0...255 in *einer* schnellen Umdrehung statt in zehn zu durchlaufen. Da ich nun über praxiserprobte Impulsraten Bescheid wusste, konnte ich einen Algorithmus für die Messung der Taktrate und die Umrechnung in einen Beschleunigungsfaktor entwerfen. Das Resultat war erstaunlich gut!

Aufgabe 3

Die dritte Aufgabe war es, die zweifarbigen LEDs durch ein alphanumerisches LC-Display zu ersetzen. Ich hätte die LEDs erhalten können, aber dies mit einem zusätzlichen Port-Expander am Controller erkauft. Außerdem war es mein Bestreben, so nah wie möglich am J²B-Board zu bleiben. Zudem können auf einem LCD viel mehr Informationen übermittelt werden, was dem Benutzer erspart, dauernd im Handbuch nachschlagen zu müssen. Acht Drehregler am J²B-Board erfordern ein 2x16-LCD, sonst gibt es nicht genug Platz für alle Infos (**Bild 3**). Gute Texte für ein Display zu finden ist schon schwierig, noch komplizierter ist es aber, Werte oder Positionen der sechs Live-Steuerungen sinnvoll und deutlich wiederzugeben. Am einfachsten wäre es, zwei Reihen mit je drei Werten anzuzeigen, dann wäre aber die Position des Wertes (zwischen einer Min- und einer Max-Einstellung) ziemlich undeutlich. Es hat mich allerhand Gehirnschmalz gekostet, aber schließlich habe ich die wohl optimale Lösung in Form kleiner Schieberegler-Symbole gefunden. In einem

Standard-LCD können bis zu acht 5x7-Punkt-Zeichen vom Benutzer definiert werden, und das ist gerade genug, um einen vertikalen Schieberegler mit sieben Positionen zu erstellen. Über zwei Zeilen (zwei Zeichen übereinander) ergibt das 14 und zusammen mit zwei speziellen Zeichen für 0 und Maximum dann 16 Postionen.
Ein besonderer Algorithmus ermittelt, ob der Benutzer eine der sechs Live-Steuerungen oder einen Funktions/Wert-Encoder (zum Beispiel zur Einstellung der Wellenform) nutzt, so dass die Software automatisch zwischen zwei Seiten umschalten und die relevanten Informationen anzeigen kann. Ich habe die Anzeige des roten und grünen Modus des Atmegatron einfach durch eine zweifarbige LED realisiert, obwohl ich noch lieber die Hintergrundbeleuchtung des LCD für diesen Zweck eingesetzt hätte.

Auf der Hardwareseite

Nun hatte ich einen arbeitenden, wenn auch nicht voll funktionsfähigen Prototyp auf mein J²B-Board (Bild 3) portiert und es war an der Zeit, sich um das Hardware-Design des Synthesizers zu kümmern. Ich wollte zwar das J²B-Board als Basis benutzen, aber während der Experimente wurde die unergonomische Positionierung der Drehgeber auf der Platine offenbar. Sie lagen zu dicht beieinander. Da ich ohnehin ein Add-on-Board für das Antialiasing-Filter, die MIDI-Schnittstelle, den Kopfhörerverstärker und ein EEPROM entwerfen musste, entschloss ich mich zu einer großen Hauptplatine mit mehr Platz für die Drehencoder. Dann gäbe es auch genug Platz, um so weit wie möglich Durchsteck-Bauteile aus der Bibliothek *Elektor Labs Preferred Parts* (ELPP) einzusetzen. Und das J²B-Hirn würde sich dann auf einer Tochterplatine befinden.
Der Entwurf der Baugruppe hat nicht lange gedauert. Für das Antialiasing hat mir das freie FilterLab von Microchip ein Tschebyscheff-Filter fünften Ordnung mit einer Grenzfrequenz von 15 kHz (in meinem Alter hört man darüber nichts mehr) berechnet. Das hat etwa 30 Sekunden gedauert. Der „Rest“ dauerte etwas länger, aber nach zehn Tagen lagen die geätzte Platine und alle Bauteile auf meinem Labortisch. Ich konnte mit der Montage des neuen Prototyps beginnen.
Leider dämmerte es mir jetzt (viel zu spät), dass mein Hardware-Ansatz eine einzige Katastrophe war. Es gab einfach zu viele Verbindungen zwischen J²B- und Hauptplatine. Und um es noch schlimmer zu machen, waren die meisten der Verbindungen fast unzugänglich. Nach einigen vergeblichen (dummen) Bemühungen, das Problem zu lösen, habe ich dann aufgegeben.
Ich hatte jetzt eine unbrauchbare mechanische Konstruktion mit einem immer noch vorhandenen leichten Speicherproblem. Das hat mich nachdenklich gemacht. Um die Dinge richtig zu machen, sollte ich das J²B-Board erst einmal vergessen und das Design komplett neu beginnen. Aber, wenn ich schon alles neu machen muss, warum dann an dem LPC1343 kleben? Seit diese Controllerfamilie vor einigen Jahren erschien, sind neue Bauteile wie der LPC1347 aufgetaucht, der nicht nur die doppelte Speichergröße (64 KB) besitzt, sondern auch 4 KB internes EEPROM. Er hatte auch den ausgezeichneten USB-Bootloader, also warum nicht zum neuen 1347 wechseln?

Mk.II

Mein neues Design (V2 oder Mk.II, wie Sie wollen) basiert daher auf dem LPC1347 und ist durch-

Anmerkung zur Klangqualität

Die Atmegatron produziert keine qualitativ hochwertige Musik, sondern ist ein LoFi-8-bit-Synthesizer und klingt auch so. Er ist dank seiner speziellen Verzerrungsalgorithmen in der Lage, aggressive und hässliche Geräusche und alle Arten von „Robotersounds“ zu erzeugen. Es klingt ein bisschen wie ein Casio-Keyboard der 80er Jahre auf Speed. Wenn dies berücksichtigt wird, kann man einige hervorragende Sounds erzeugen und damit jede Menge Spaß haben. Der Arpeggiator ist ein nettes zusätzliches Feature. Ich habe die Klangqualität des J²B-Ports ein „bitchen“ verbessert: Die PWM des LPC1347 ist 16 bit tief, während Atmegatron nur 8 bit erlaubt (der 16-bit-PWM-Modus des AVR ist zu langsam für diese Anwendung). Ich habe die PWM-Tiefe um ein Bit erhöht, so dass es jetzt ein 9-bit-Synthesizer ist. Ich habe auch die PWM-Frequenz mehr als verdoppelt, so dass das Anti-Aliasing-Filter (5. Ordnung anstelle 3. Ordnung beim Atmegatron) bessere Ergebnisse liefert, was zur weiteren Verbesserung der Klangqualität führt.
Es gibt noch viele weitere Möglichkeiten zur Klangverbesserung. Die Sound-Engine-Algorithmen neigen dazu, ihre Ausgaben bei acht Bit abzuschneiden. Bei einem 32-bit-Controller ist dies ein wenig schade. Die Wavetables könnten länger sein. Die Filterung verwendet jetzt eine Gleitkomma-Arithmetik, was zwar ok ist, aber auch aber zeit- und speicheraufwändig. Ein Übergang zu Festkomma-Arithmetik würde kostenlos eine Menge Rechenleistung für andere Sound-Algorithmen frei machen.
Eine weitere interessante Erweiterung wäre, virtuelle Synthesizer-Funktionen (zum Beispiel zur Steuerung eines Software-Synthesizers auf dem PC) über den USB-Port hinzufügen und/oder MIDI über USB.
Der J²B-Synthesizer ist eine preiswerte Plattform, um mit computerbasierter Klangsynthese direkt am Gerät zu experimentieren. Das Stichwort ist Prog’n’Play!

Bild 4.
Schaltung der Hauptplatine des J²B-Synthesizers.

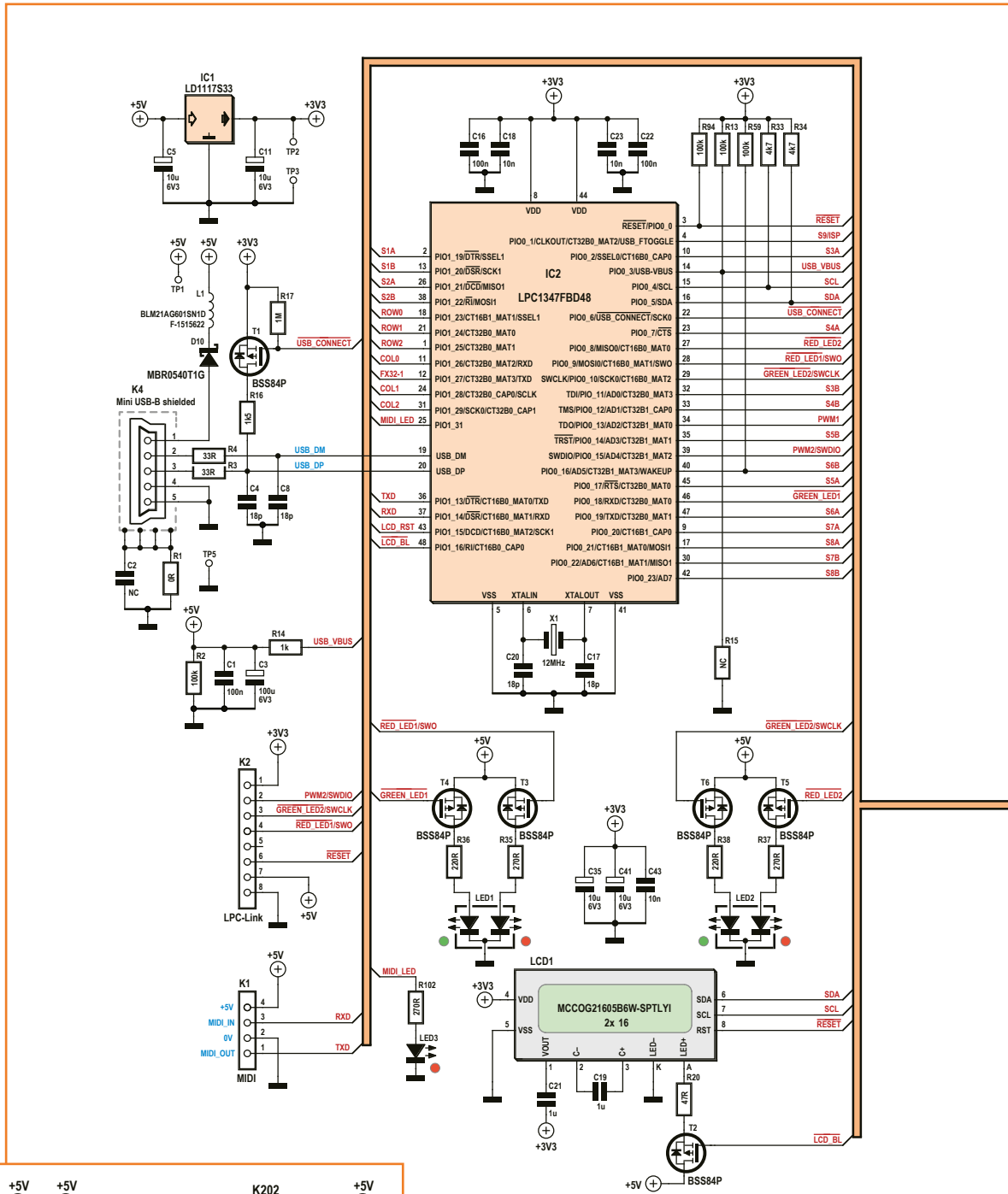
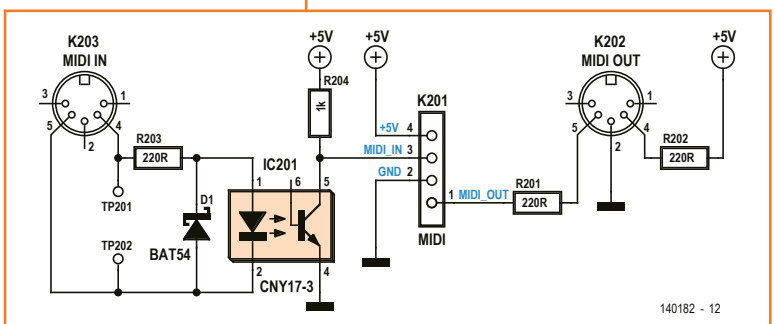


Bild 5.
Schaltung des MIDI-Interfaces. Es ist wegen der Größe der 5-poligen DIN-Buchse auf einer eigenen Platine untergebracht. Damit ist das Design mechanisch gesehen sehr flexibel.



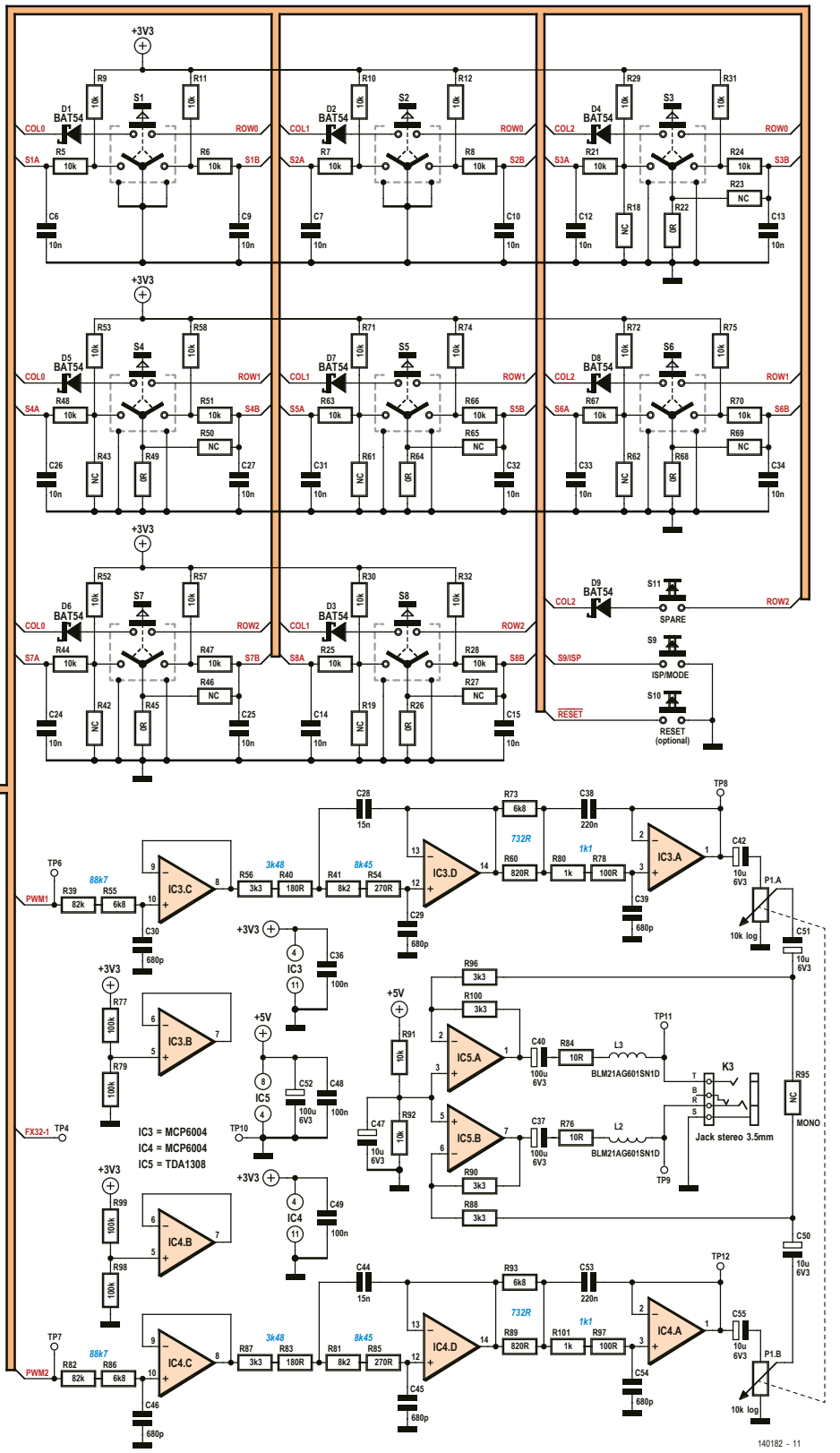

+3V3
S1
S2
S3
S4
S5
S6
S7
S8
S11
SPARE
S9
S9/ISP
ISP/MODE
S10
RESET
RESET (optional)
D1 BAT54
D2 BAT54
D3 BAT54
D4 BAT54
D5 BAT54
D6 BAT54
D7 BAT54
D8 BAT54
D9 BAT54
COL0
COL1
COL2
ROW0
ROW1
ROW2
IC3.A
IC3.B
IC3.C
IC3.D
IC4.A
IC4.B
IC4.C
IC4.D
IC5.A
IC5.B
IC3 = MCP6004
IC4 = MCP6004
IC5 = TDA1308
Jack stereo 3.5mm
MONO
P1.A
P1.B
10k log
PWM1
PWM2
FX32-1
BLM21AG601SN1D
140182 - 11

Bild 6.
Der J²B-Synthesizer in einem vom Laser geschnittenen Acrylgehäuse.

gängig mit SMDs bestückt (mit Ausnahme der Buchsen, Encoder und LEDs). Nicht aus Gründen der Symmetrie habe ich eine zweite zweifarbige LED hinzugefügt. In der Bedienungsanleitung des Atmegatron hatte ich übersehen und erst bei der Portierung gemerkt, dass eine LED eine MIDI-Aktivität anzeigt. Da der LPC1347 wie der LPC1343 über mehrere Kanäle pro PWM-Timer verfügt, habe ich beschlossen, ein zweites Antialiasing-Filter hinzuzufügen, um einen Zweikanal-Betrieb zu ermöglichen. Schließlich verwenden wir einen 32-bit-Cortex-M3, der mit 72 MHz getaktet wird. Da dürfte wohl ausreichend Rechenleistung vorhanden sein...

Die Schaltung

Beim Blick auf die Schaltung des Synthesizers (**Bild 4**) werden Sie keine Überraschungen erleben. Die acht Drehencoder belegen alle verfügbaren I/O-Ports, zwei pro Encoder. Die Widerstände bei den sechs Live-Steuerungen dienen der Flexibilität: Man kann durch die richtigen Widerstandswerte die Drehgeber durch Potentiometer ersetzen, da eine Seite der Encoder mit je einem Eingang des Analog/Digital-Wandlers (ADC) des Controllers verbunden ist. Nehmen Sie Encoder S5: Normalerweise wären R63, R64, R66, R71, R74, C31 und C32 vorhanden, um aber ein Poti

Stückliste

Hauptplatine

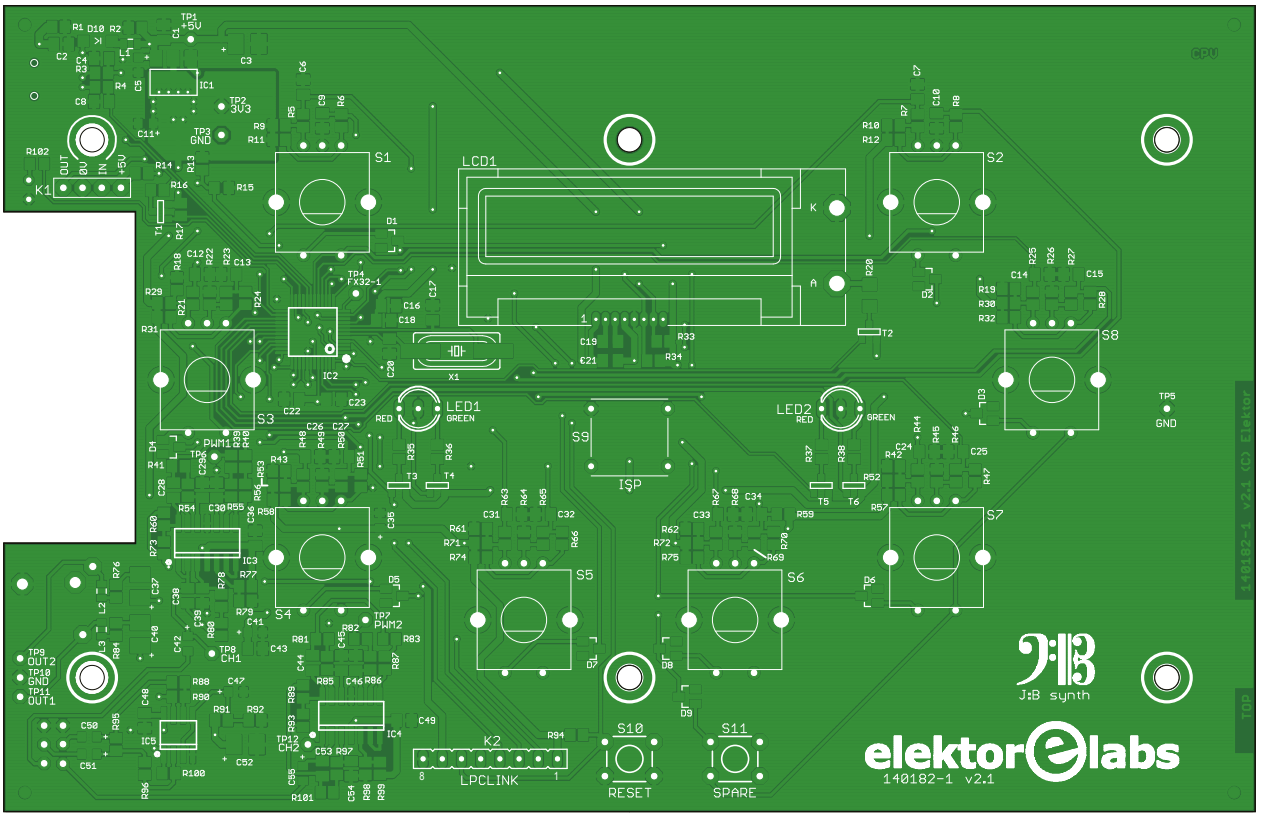
Widerstände:
alle SMD 0805, 5%, 0,1 W
R1,R22,R26,R45,R49,R64,R68 = 0 Ω
R2,R13,R59,R77,R79,R94,R98,R99 = 100 k
R3,R4 = 33 Ω
R5-R12,R21,R24,R25,R28..R32,R44,R47,R48,R51,R52,R53,R57,R58,R63,R66,R67,R70..R75,R91,R92 = 10 k
R14,R80,R101 = 1 k
R16 = 1k5
R17 = 1 M
R20 = 47 Ω, 1206, 0,25 W
R33,R34 = 4k7
R35,R37,R54,R85,R102 = 270 Ω
R36,R38 = 220 Ω
R39,R82 = 82 k
R40,R83 = 180 Ω
R41,R81 = 8k2
R55,R73,R86,R93 = 6k8
R56,R87,R88,R90,R96,R100 = 3k3
R60,R89 = 820 Ω
R78,R97 = 100 Ω
R76,R84 = 10 Ω
P1 = 10 k-Potentiometer, Stereo, logarithmisch
R15,R18,R19,R23,R27,R42,R43,R46,R50,R61,R62,R65,R69,R95 = nicht verwendet

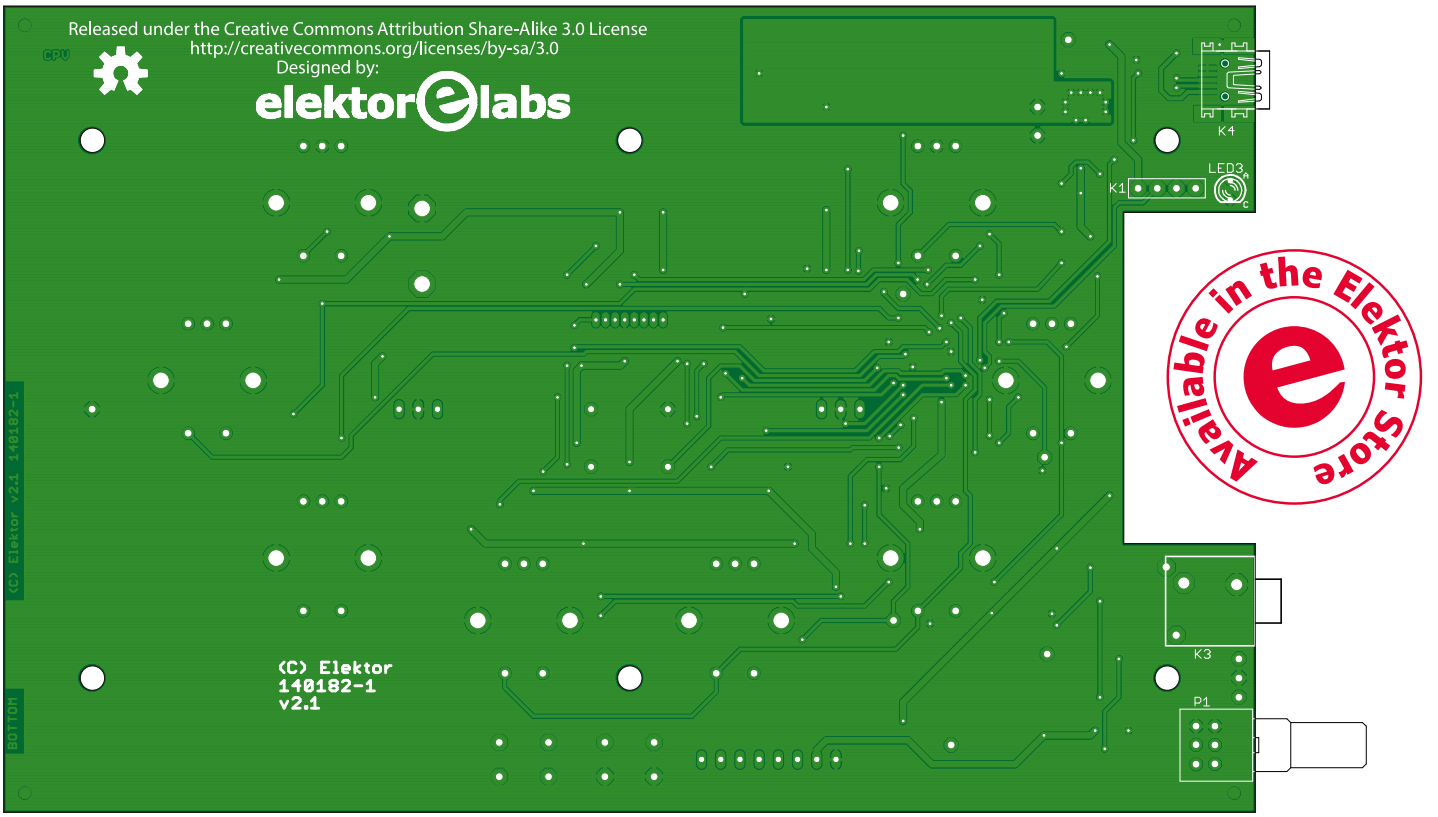
Kondensatoren:
alle SMD 0805
C1,C16,C22,C36,C48,C49 = 100 n
C3,C37,C40,C52 = 100 µ, 6V3 Tantal, Größe B
C4,C8,C17,C20 = 18 p
C5,C11,C35,C41,C42,C47,C50,C51,C55 = 10 µ, 6V3 Tantal
C6,C7,C9,C10,C12-C15,C18,C23-C27,C31-C34,C43 = 10 n
C19,C21 = 1 µ
C28,C44 = 15 n
C29,C30,C39,C45,C46,C54 = 680 p
C38,C53 = 220 n
C2 = nicht montiert

Induktivitäten:
L1,L2,L3 = Ferritperle, 0Ω21, 0,6 A, SMD 0805

Halbleiter:
D1...D9 = BAT54C (SOT-23)
D10 = MBR0540T1G
IC2 = LPC1347FBD48
IC1 = LD1117S33CTR
IC3,IC4 = MCP6004-I/SL
IC5 = TDA1308T/N2
LED1,LED2 = LED, zweifarbig rot/grün, CC, 5 mm
LED3 = LED, rot, 3 mm
T1...T6 = BSS84P (SOT-23)

Außerdem:
K1 = 4-polige Stiftleiste, Raster 0,1”
K2 = 8-polige Stiftleiste, Raster 0,1”
K3 = Klinkenbuchse 3,5 mm, Stereo
K4 = Mini-USB-B-Buchse, abgeschirmt
S1...S8 = Drehencoder
S9 = Drucktaster Multimec 3FTL6
LCD1 = LCD 2x16, I²C, z.B. Midas MCCOG21605B6W-SPTLYI
X1 = Quarz 12 MHz
BOX1 = Hammond 1597DGY oder lasergeschnitten.
Hauptplatine 140182-1 (Elektor-Shop)


LCD1
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
LED1
LED2
ISP
K1
K2
LPCLINK
RESET
SPARE
J:B synth
elektorlabs
140182-1 v2.1


Released under the Creative Commons Attribution Share-Alike 3.0 License
http://creativecommons.org/licenses/by-sa/3.0
Designed by:
elektorlabs
(C) Elektor
140182-1
v2.1
K1
K3
K4
P1
LED3
Available in the Elektor Store

einzusetzen, setzen Sie statt dessen R61, R65 und R74 (0 Ω) und vielleicht C32 ein. Signal S5B (der Schleifer) ist dann mit dem ADC-Eingang AD3 verbunden.

Die Drehencoder haben integrierte Drucktasten, die miteinander in Form einer 3x3-Matrix verbunden sind. Eine Stelle der Matrix ist (zurzeit) ohne Funktion. Ein spezieller Reset-Taster ist ebenso vorhanden wie ein eigener Modus-Taster (für die Modi Rot und Grün). Diese Taste wird auch verwendet, um den Controller in den Firmware-Update-Modus zu versetzen. Die rot/grünen LEDs werden von MOSFETs angesteuert, so dass eine Menge LED-Strom fließen kann, ohne dass der Controller überfordert wäre. Das LCD besitzt eine I²C-Schnittstelle. Es ist großartig, es passt in seiner Größe und Höhe zu den Drehencodern, kostet weniger als ein typisches LCD-Modul und belegt weniger Controlleranschlüsse!

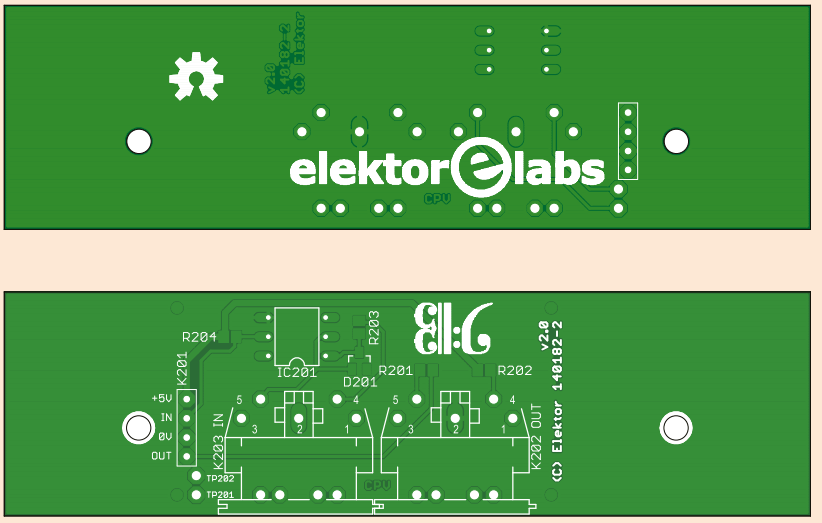
Die Anti-Aliasing-Filter bestehen jeweils aus drei Operationsverstärkern und einigen Widerständen und Kondensatoren. Die Widerstände sind in je zwei Werte geteilt, so dass die E12-Reihe verwendet werden kann. Je ein Operationsverstärker pro Kanal bleibt ungenutzt. Die Filter-Ausgänge, die über die Lautstärke-Potis geführt werden, sind mit zwei Kondensatoren von der Gleichspannung entkoppelt, so dass keine knisternden Geräusche auftreten können. Hi-End-Audio-Freaks mögen die Stirn runzeln, aber dies reicht für eine solche Anwendung aus. Ein Stereo-Kopfhörerverstärker stellt genügend Leistung für viele Anwendungen zur Verfügung.

Die MIDI-Schnittstelle (**Bild 5**) ist auf einer separaten Platine untergebracht, da die DIN-Anschlüsse zu hoch für das von mir gewählte Gehäuse sind. So können sie aber niedriger montiert werden und passen perfekt. Das Hardwaredesign erwies sich nun ebenfalls als perfekt, ich habe für Sie eine Bohrschablone [2] gezeichnet. Ich habe auch einen Versuch mit Laserschneiden gemacht (**Bild 6**). Auch diese Vorlagen stehen Ihnen unter [2] zur Verfügung.

Nochmal: Portierung

Ein neues Hardware-Design mit einem neuen Mikrocontroller erfordert unweigerlich eine zweite Runde der Code-Portierung. Wenn Sie nun denken, der neue Controller kommt ja aus der gleichen Familie wie der alte und die Portierung wäre ein Kinderspiel von einigen Minuten (das war es, was ich erwartet hatte), weit gefehlt!

In Wahrheit war es kompliziert, weil NXP beschlossen hatte, die Bibliothek für den Chip zu ändern, der eine Architektur aufweist, die näher an der Familie LPC11Cxx liegt als an der Familie LPC134x. Kurz gesagt, ist der LPC1347 weder pin- noch 100%-ig codekompatibel mit dem LPC1343. So musste ich in den sauren Apfel



Stückliste

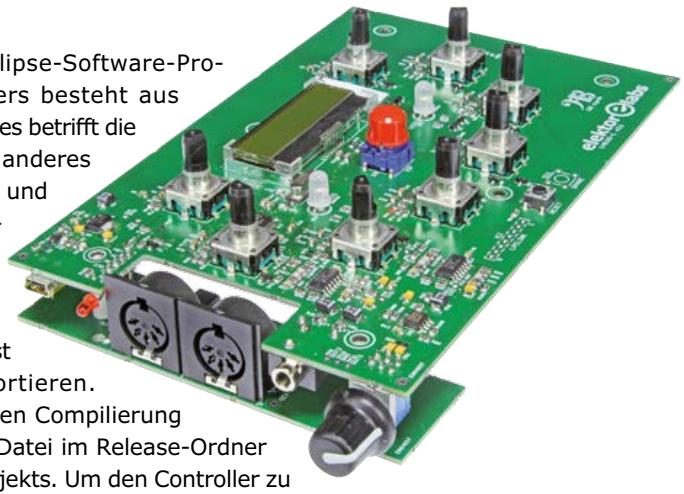
MIDI Board

Widerstände:
alle SMD 0805, 5%, 0,1 W
R201,R202,R203 = 220 Ω
R204 = 1 k

Halbleiter:
D201 = BAT54C (SOT-23)
IC201 = CNY17-3 (DIP-6)

Außerdem:
K201 = 4-polige Buchsenleiste, Raster 0,1“, vertikal
K202,K203 = 5-polige DIN-Fassung für Platinenmontage, 180°
MIDI-Platine 140182-2 (Elektor-Shop)

beißen und den Code durchackern, um die Kompatibilitätsprobleme zu beheben.



Do it yourself

Das LPCXpresso/Eclipse-Software-Projekt des Synthesizers besteht aus drei Teilprojekten. Eines betrifft die Chip-Bibliothek, ein anderes die Board-Bibliothek und ein drittes die Synthesizer-Anwendung selbst. Sie können das Paket als Zip-Datei (ohne zuerst auszupacken) importieren. Nach der erfolgreichen Compilierung finden Sie eine BIN-Datei im Release-Ordner des Synthesizers-Projekts. Um den Controller zu programmieren, brauchen Sie bloß den Synthesizer mit einem freien USB-Anschluss Ihres Windows-PCs zu verbinden, während Sie die Rot/Grün-Taste gedrückt halten. Der Synthesizer muss dabei ausgeschaltet sein, die Energie liefert nur der PC. Wenn alles gut geht, erkennt Windows einen USB-Speicherstick von 64 KB mit einer Datei im Inhalt. Löschen Sie diese Datei und kopieren Sie die BIN-Datei darauf. Drücken Sie die Reset-Taste oder unterbrechen Sie kurzzeitig die Stromversorgung des Synthesizers. Das Display sollte Sie mit der Meldung „J2B Synthesizer“ in großer Schrift begrüßen. Wenn die Anzeige verschwindet, ist der Synthesizer einsatzbereit. Schließen Sie ein MIDI-Keyboard und Kopfhörer an und das Spiel kann beginnen.

Alles ist offen

Dieses Projekt ist zu 100 % Open Source, inklusive der Hardware. Alle Design-Dateien, Hardware, Software und mechanische Unterlagen sind kostenlos erhältlich bei [2]. Ich lade Sie ein, einen Blick darauf zu werfen und zu versuchen, den Synthesizer weiter zu verbessern. Dazu gibt es sicher viele Möglichkeiten. Wenn Sie Ihren eigenen Synthesizer bauen, schicken Sie mir doch bitte ein Foto oder fügen einen Beitrag an die Projekt-Seite [2] an.

User Manual

Eine gute Sache bei der Portierung des Atmegatron (anstelle einer Neuentwicklung) ist, dass wir von der Bedienungsanleitung und dem Librian-Utility des Atmegatron profitieren können. Dies hat mir einen Haufen Schreibarbeit erspart. Laden Sie die Anleitung von [1] herunter und lesen Sie sie sorgfältig, sie ist tatsächlich umfassend. Der J²B-Synthesizer besitzt alle originalen Steuerungen des Atmegatron mit Ausnahme der Klangregelung. Die Drucktasten der sechs Live-Steuerungen, die ein schnelles Zurücksetzen der Werte eines Parameters emöglichen, sind Zugabe: Sie gibt es am Atmegatron nicht.

(140182)

Weblinks

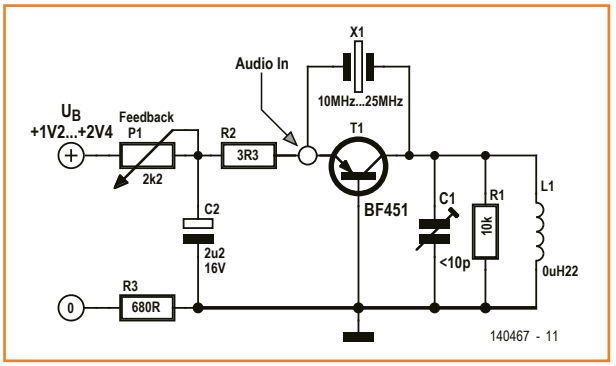
[1] Atmegatron: http://soulsbysynths.com/

[2] Projekt-Downloads: www.elektor-magazine.de/140182

[3] J2B: Vielseitiges HMI-Modul mit ARM Cortex-M3. Elektor September 2011, www.elektor-magazine.de/110274; www.elektor-labs.com/node/3832

FM-Synchro-Sender

Kapazitiver Dreipunkt-Oszillator



Von
Hans-Norbert Gerbig
(D)

Kleine UKW-Sender niedrigster Leistung mit Frequenzmodulation zum Testen von UKW-Empfängern lassen sich ja relativ simpel aufbauen. Dieser hier ist preiswert, einfach und trotzdem stabil, da seine Frequenz quarzstabilisiert ist. Der in **Bild 1** präsentierte sogenannte KDO (kapazitiver Dreipunkt-Oszillator) ist in Basisschaltung konzipiert und schwingt auf seiner per Quarz stabilisierten Eigenfrequenz. Er lässt sich in einem bestimmten Mitnahmebereich von NF-Signalen synchronisieren. In diesem Bereich macht er alle Frequenzänderungen der Steuersignale mit. Der Mitnahmebereich kann verbreitert werden, wenn man parallel zum Schwingkreis einen Widerstand schaltet.

Der Oszillator schwingt mit seiner Eigenfrequenz und der gegebenen Amplitude konstant weiter. Der Mitnahmebereich entspricht genau der steuernden NF, es wird aber die links und rechts vom Mitnahmebereich störende Amplitudenmodulation quasi abgeschnitten. Das Resultat ist eine Frequenzmodulation!

Die modulierende Niederfrequenz wird in der Basis-Schaltung extrem niederohmig (3,3...4,7 Ω) über dem Emitterwiderstand eingekoppelt. Der Wert des Widerstands ist von Bedeutung für die „Emphase“ bei der Modulation des UKW-Senders. Hohe Frequenzen werden überbetont. Bei der Demodulation in normalen UKW-Empfängern wird das höherfrequente Audiospektrum abgesenkt und somit korrigiert, sodass das Audiospektrum bis etwa 15 kHz ausgewogen klingt.

Der Oszillator hat natürlich nur eine sehr geringe HF-Leistung; sie liegt irgendwo im pW-Bereich. Die Basisschaltung schwingt recht stabil und ist daher wenig „handempfindlich“. Auf eine Abschirmung kann man also verzichten. Ein in der unmittelbaren Nachbarschaft stehender UKW-Empfänger kann die schwache Abstrahlung der frequenzmodulierten Oberwellen der Quarzfrequenz dann problemlos empfangen.

Übrigens kann man in der Schaltung auch ohne weiteres einen NPN-HF-Transistor statt des PNP-Exemplars einsetzen. Man muss hierzu lediglich C2 und die Versorgungsspannung U_B umpolen. Für eine stabile Schwingung muss neben P1 auch C1 passend zu L1 und X1 eingestellt werden.

(140467)

FM-Synchro-Empfänger

Mit kapazitivem Dreipunkt-Oszillator

Von
Hans-Norbert Gerbig
(D)

Ein kleiner UKW-Empfänger braucht nicht unbedingt eine komplexe FM-Demodulation und muss auch kein Doppelsuperhet sein. Auf der Basis eines kapazitiven Dreipunkt-Oszillators in Basisschaltung als Synchro-Demodulator geht das viel einfacher.

Bei diesem „UKW-Radio“ mit Synchro-FM-Demodulator wird das gleiche Prinzip wie beim FM-Synchro-Sender verwendet, der an anderer Stelle in dieser Ausgabe beschrieben ist. Das Ganze hört sich komplizierter an, als es ist. Im Grunde ist es nichts anderes als ein gesteuerter

bzw. mitgezogener Oszillator. Dieser ist in der linken Hälfte der Schaltung von **Bild 1** rund um T1 aufgebaut. Auf der rechten Seite ist lediglich die NF-Verstärkung mit Entzerrung zu finden.

Synchro-Demodulator

Ein Sinusoszillator lässt sich in einem Mitnahmebereich (spektral rechts und links von der Eigenfrequenz) von einer extern zugeführten Frequenz synchronisieren. Er macht in diesem Gebiet alle Frequenzänderungen (NF und HF) des Steuersignals mit.

Der Oszillator ist in Basisschaltung mit dem PNP-Transistor T1 ausgeführt. Hier bewirkt C2 die HF-Rückkopplung. Die Schwingfrequenz ergibt sich aus der Induktivität von L1 und dem über P1 einstellbaren Wert der Kapazitätsdiode D1. Der Mitnahmebereich ist umso breiter und damit die Trennschärfe umso geringer, je stärker der Schwingkreis gedämpft ist. Die Dämpfung wird durch den parallel zum Schwingkreis geschalteten Widerstand R2 erreicht. Ohne R2 ist der Empfänger also sehr selektiv und mit dem Widerstand wird der Mitnahmebereich breiter.

Eigenschaften

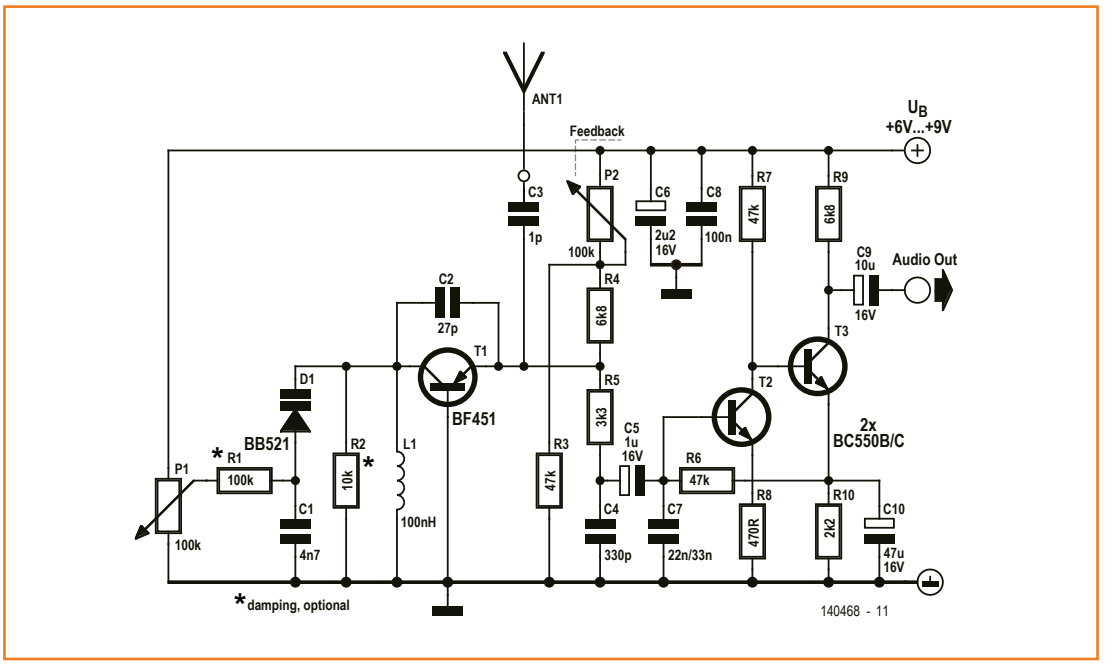
Bei der vorliegenden Schaltung kommen die Eigenschaften eines mitgezogenen Oszillators voll zur Geltung: Seine Ausgangsspannung ist sehr konstant und überlagerte Störungen werden unterdrückt. Außerdem wird der Oszillator ausschließlich von der Frequenz des stärksten Senders synchronisiert. Andere Empfangsfrequenzen mit kleinerer Amplitude, selbst wenn sie im gleichen Kanal auftreten, können den Oszillator nicht aus dem Tritt bringen und haben daher keine Wirkung auf den Demodulator. Schwächere Sender werden daher vollständig unterdrückt, selbst wenn ihre Feldstärke bis zu 70 % der Feldstärke des gewünschten Senders erreicht. Dies ergibt eine ungewöhnlich gute Trennschärfe.

Die proportionalen Audioschwingungen werden am Emitter des HF-Transistors T1 abgenommen. Der Widerstand R5 und die Kondensatoren C4 und C7 sieben als Tiefpass noch vorhandene HF-Anteile aus. C7 hat noch eine weitere Aufgabe: Die typische Emphase des Hochtonspektrums der Audiosignale eines FM-Senders wird rückgängig gemacht.

Die Basisschaltung ist wie beim KDO-Sender recht rückwirkungsarm und dadurch kaum handempfindlich. Auf eine Abschirmung des Oszillators kann man also verzichten.

Mit T2 und T3 wurde eine simple Verstärkung der Audiosignale realisiert. Man kann am Ausgang eine kleine integrierte Endstufe oder sogar direkt einen hochohmigen Kopf- oder Ohrhörer anschließen.

(140468)



LED-Treiber für Glühlämpchen

LEDs in kleinen Taschenlampen nachrüsten

Von **Peter Krüger** (D)

Kleine Taschenlampen haben häufig eine kleine Glühbirne eingebaut, die von zwei 1,5-V-Batterien versorgt wird. Manche Exemplare arbeiten sogar mit nur einer Zelle. Das ist zu wenig Spannung für eine weiße LED. Dazu braucht es einen Aufwärtswandler. Um genau so eine Schaltung geht es hier: einfach, selbstgebaut und trotzdem winzig.

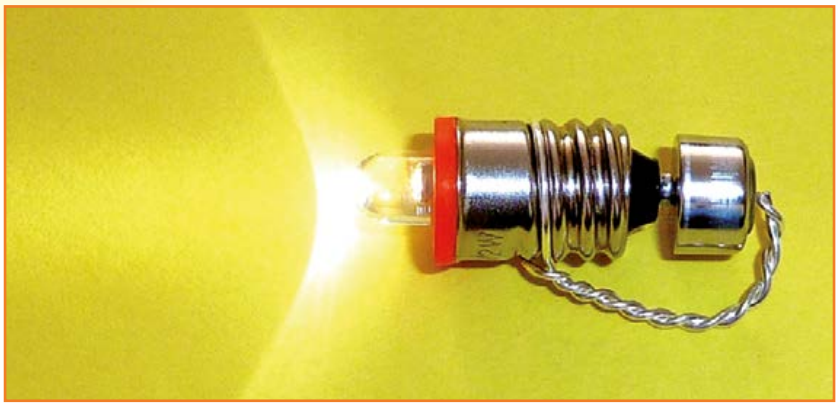


Bild 1.
Step-up-Konverter samt LED, integriert in das E-10-Schraubgehäuse eines Glühlämpchens. Dahinter ist eine kleine Knopfzelle vom Typ AG5 (1,5 V mit 60 mAh) zu sehen, die für über eine Stunde Dauerlicht sorgt.

Der Autor stand vor dem Problem, eine kleine Taschenlampe „auf LED“ umzurüsten. Der große Vorteil ist, dass LEDs gegenüber Glühlampen einen mehrfach besseren Wirkungsgrad haben. Bei gleicher Helligkeit halten daher die Batterien deutlich länger. Hierfür muss aber eine weiße LED dauerhaft mit Spannungen über 3 V und dem passenden Strom versorgt werden.

Dafür könnte man zwar eine fertige integrierte Lösung nehmen, doch gibt es dabei schnell Platzprobleme. Von daher reifte der Entschluss, die Sache gleich komplett selbst und richtig zu machen. Für einen selbstschwingenden Step-up-Konverter braucht es nämlich nicht mehr als einen simplen Transistor. Solch eine Minimalschaltung hat den Vorteil, so klein zu sein, dass die ganze Elektronik samt LED in das Edisongewinde eines normalen Glühlämpchens passt (siehe **Bild 1**). Der Glühlampenersatz ist damit bequem und einfach möglich.

Step-up-Konverter

Der Aufwärtswandler in **Bild 2** ist wirklich sehr einfach: Neben dem schon erwähnten Transistor sind lediglich noch ein Widerstand, zwei Kondensatoren und die obligatorische Spule notwendig. Die Spule selbst hat eine Anzapfung, wodurch sich eine Wicklung zur Mitkopplung (AB) und eine zweite zur Energiespeicherung (BC) ergibt. Da die Spannungsversorgung an Punkt B anliegt, sind die Signale der beiden Wicklungen um 180° phasenverschoben. Zusammen mit den 180° des Transistors ergibt sich eine Mitkopplung, welche die Schaltung zum Schwingen bringt.

Das funktioniert folgendermaßen: Beim Einschalten fließt Strom durch den Spulenteil AB über R1 und den parallelen Kondensator C2 in die Basis von T1, der daraufhin durchschaltet. In der Folge fließt auch Strom von Punkt B nach C in den Kollektor von T1. Dies wiederum lässt die Spannung an Punkt A über $+U_B$ steigen und über C2 wird T1 richtig in die Sättigung gesteuert. Der Strom durch die Punkte BC steigt nun solange an, bis der Kern in die Sättigung geht, was in der vorlie-

genden Dimensionierung etwa 2...3 µs benötigt. Ab da steigt der Strom nicht mehr weiter, was die Spannung an Punkt A einbrechen lässt. In wenigen Nanosekunden schaltet T1 ab und die im Kern der Spule gespeicherte Energie entlädt sich in knapp 2 µs über die weiße LED. Anschließend beginnt der Zyklus von neuem.

In der angegebenen Dimensionierung schwingt die Schaltung bei etwa 200...250 kHz. Sie schwingt solange, bis $+U_B$ unter die Schwellenspannung der BE-Strecke von T1 = 0,55 V fällt. Die einfache Schaltung sorgt dafür, dass der LED-Strom und damit die Helligkeit mit sinkender Batteriespannung abnimmt - ganz wie bei einer richtigen Glühbirne.

Realisierung

Da der LED-Treiber maximal nur etwa 75 mW zu übertragen hat, kann der Kern der Spule recht klein ausfallen. Eine kleine Ferritperle ist völlig ausreichend. Zwecks Miniaturisierung wurden für T1, R1, C1 und C2 ausschließlich SMD-Ausführungen gewählt. Damit passt die ganze Elektronik prima in das E10-Schraubengewinde einer für Taschenlampen typischen Glühbirne. Das erfordert allerdings Fingerspitzengefühl und den Einsatz einer Pinzette beim Zusammenbau, denn diese Schaltung kommt ohne Platine aus und wird wie in **Bild 3** zu sehen komplett freiluftverdrahtet.

Damit das Ganze am Ende so aussieht, gibt es ein paar Dinge zu beachten: Das zentrale Element der Schaltung ist L1. Man kann so eine Spule leider nicht fertig kaufen, sondern muss sie selbst wickeln. Für die Wicklung BC sind 14 Windungen notwendig. Anschließend kommen nochmals 5 Windungen für die Wicklung AB hinzu. Da die Ferritperle klein, bzw. ihre Bohrung nur 1,3 mm groß ist und man mit nur 0,15 mm „dünnem“ Kupferlackdraht zu tun hat, kommt man um Geduld und eine feine Pinzette nicht herum. **Bild 4** zeigt einen Wickelplan. Die Punkte entsprechen denen der Schaltung von Bild 2. Zuerst wird die Wicklung mit den 14 Windungen aufgebracht und dann darauf die mit den 5 Windungen. Die beiden Anschlüsse für die Anzapfung (Punkt B in Bild 2) kann man verdrillen und verlöten. **Bild 5** gibt einen Eindruck von der Kleinheit der fertigen Spule.
Damit die Sache nachher so aussieht wie in Bild 3, kann man sich beim freien Aufbau am Verdrah-

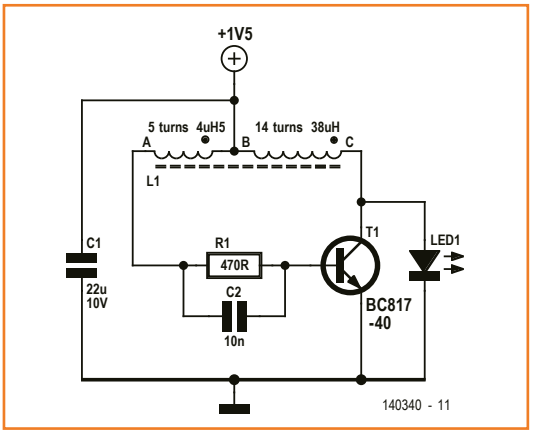


Bild 2.
Schaltung des simplen Low-Voltage-Step-up-Konverters mit nur einem Transistor, einer Spule, einem Widerstand und zwei Kondensatoren.

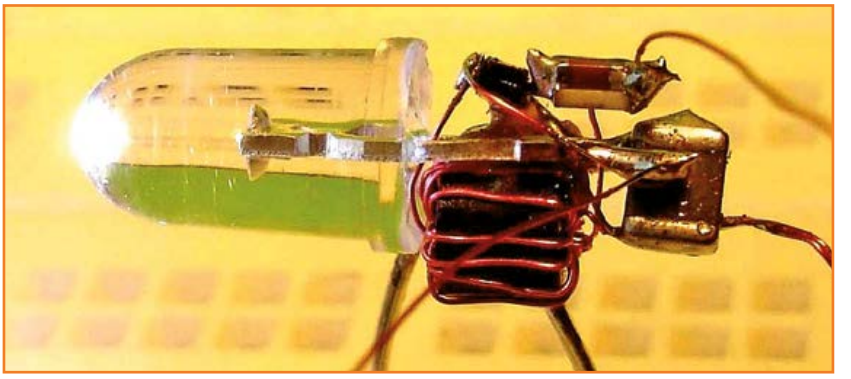


Bild 3.
Freier Aufbau der Schaltung samt LED ohne Platine.

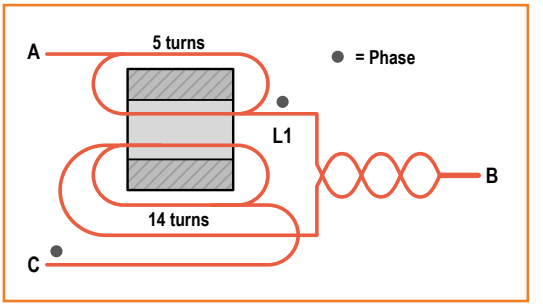
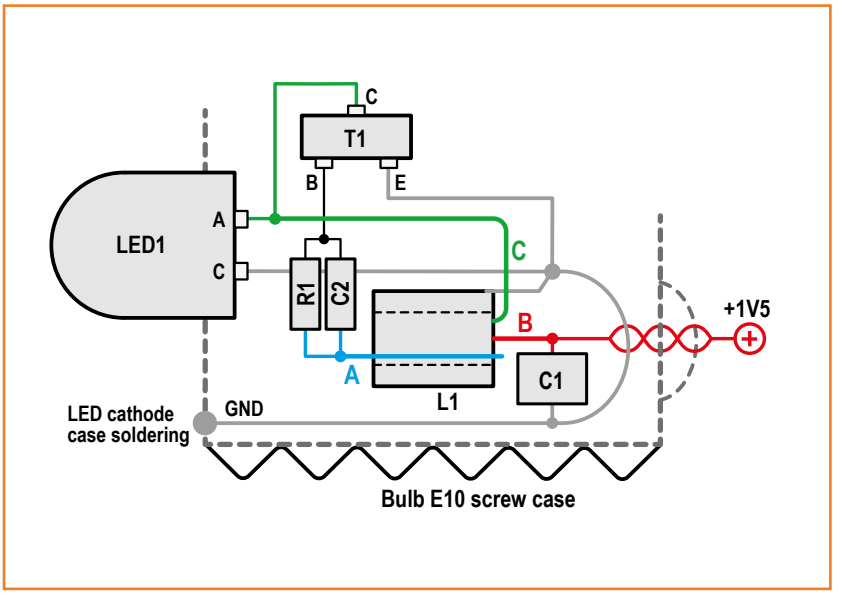


Bild 4.
Wickelplan der Spule L1. Die Punkte entsprechen denen von Bild 2.

Bild 5.
Foto der fertig gewickelten Spule des Autors. Sie ist wirklich winzig.



Stückliste

Widerstände:
R1 = 470 Ω, 0805

Kondensatoren:
C1 = 22 µ / 10 V, Vielschicht, 1206
C2 = 10 n / 25 V, Vielschicht, 0805

Spulen:
L1 = Ferritperle, beschichtet, Länge 3 mm,
ø 3,5 mm, Bohrung 1,3 mm,
z.B. Farnell 2643001501

Halbleiter:
LED1 = LED, weiß, 5 mm,
z.B. Nichia NSPW500DS b2W
T1 = BC817-40, SOT23

Außerdem:
Kupferlackdraht 0,15 mm
Glühbirne mit E10-Gewinde zum Ausschlachten

Bild 6.
Verdrahtungsschema von LED und Treiberelektronik.

tungsschema von **Bild 6** orientieren. Die LED wurde dann beim Autor in eine rote Kunststoff-U-Scheibe mit 5-mm-Bohrung gesteckt und das Ganze mit dem freipräparierten E10-Lampengewinde verlötet und verklebt.

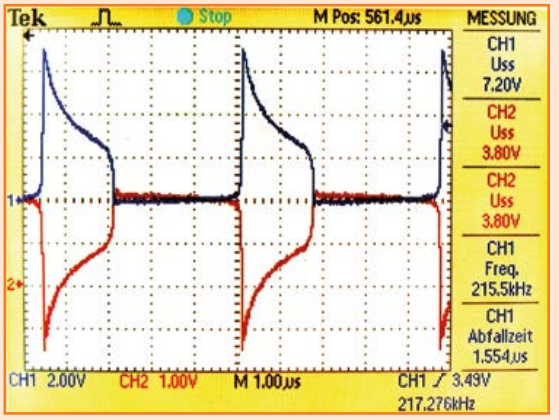
Außerdem

Auch wenn die LED mit einer kleinen 1,5-V-Knopfzelle schon über eine Stunde leuchtet, so dürften kleine, schlanke Taschenlampen ein typisches Einsatzgebiet sein, die mit ein oder zwei AA- oder AA-Primärzellen oder Akkus betrieben werden. Mit einer normalen AA-Zelle mit 2.000 mAh kommt man dann auf einen Dauerbetrieb von immerhin gut 40 Stunden. Hat die Taschenlampe zwei Zellen, so muss man eine davon durch eine Dummy-Zelle ersetzen, da der Treiber maximal mit 1,5 V = 1 Zelle versorgt werden darf! Hierzu schneidet man ein Stück Heißklebestift passend und durchbohrt ihn der Länge nach. Ein Stück Kupferdraht durch die Bohrung verlötet mit zwei Metallplättchen an den Enden, und schon ist so eine Pseudobatterie fertig.

(140340)

Oszillogramm

Der nebenstehende Screenshot zeigt die Spannungsverläufe an T1. Die Spannung am Kollektor (CH1, blau) steigt im Moment des Abschaltens auf bis zu 7,2 V. An der Basis (CH2, rot) erreicht sie einen negativen Spitzenwert von etwa -3,3 V (= U_{SS} – U_{BE}). Diese Spannung führt zum schnellen Abschalten und bleibt innerhalb der Spezifikationen für den gewählten Transistor.
Die hohe Kollektorspannung überrascht zunächst. Sie wird aber verständlich, wenn man bedenkt, dass noch kleine Induktivitäten der LED-Zuleitungen vorhanden sind (die Spannung am LED-Chip fällt kleiner aus) und der Impulsstrom deutlich über dem zulässigen Dauerstrom von 25 mA liegt. Für die LED ist das aber nicht gefährlich, da bei der hohen Schaltfrequenz die Belastung durch thermische Integration immer im grünen Bereich liegt.



Signalverstärker
für USB-Oszilloskop

Von **Christian Wendt** (D)

Mein PicoScope USB-Oszilloskop hat einen (fantastischen) Signalgenerator mit 2 V_{pp} maximaler Ausgangsspannung. Das reicht oft aus, aber halt nicht immer. Zudem regle ich meine Signalstärke sowie den Nullpunkt gerne manuell über ein Poti. Auf der Suche nach einer fertigen Lösung fand ich nicht wirklich etwas, also Lötkolben raus und los.

Die von mir entwickelte Schaltung besteht aus drei Gruppen. Der Teil um IC1A dient der Nullpunkteinstellung; mit Schalter S1 kann der Offset zugeschaltet werden. Die Offset-Spannung wird eingestellt mit P1.

Der zweite Teil rund um IC1B dreht das Ausgangssignal des Funktionsgenerators um 180° in der Phase, wobei mit P2 die Signalamplitude eingestellt wird. Der dritte Teil rund um IC2 dient als Endstufe, sie ist als Addierer beschaltet. Die Offset-Spannung wird hier etwa dreifach verstärkt (R8/R6), das durch IC1B gelieferte Signal des Funktionsgenerators wird etwa acht Mal verstärkt (R8/R7). Insgesamt kann man sowohl den Offset als auch die Signalamplitude des Ausgangssignals zwischen -12 V und +12 V einstellen. Der Frequenzbereich läuft von DC bis etwa 1 MHz. Es gibt zwei parallelgeschaltete Ausgänge (BNC-Stecker), so dass man gleichzeitig einen Oszilloskop-Eingang und die zu prüfende Schaltung anschließen kann.

Die Spannungsversorgung erfolgt über ein Netzteilmodul von Traco, das ±15 V liefert. Das Modul lag noch bei mir herum; es ist zwar teuer, aber dafür bequem einzubauen. Natürlich kann man auch selbst ein kleines Netzteil bauen mit Netztrafo, Brückengleichrichter, Pufferelko und zwei Spannungsreglern. Die Stromaufnahme liegt bei unter 100 mA.

(140346)

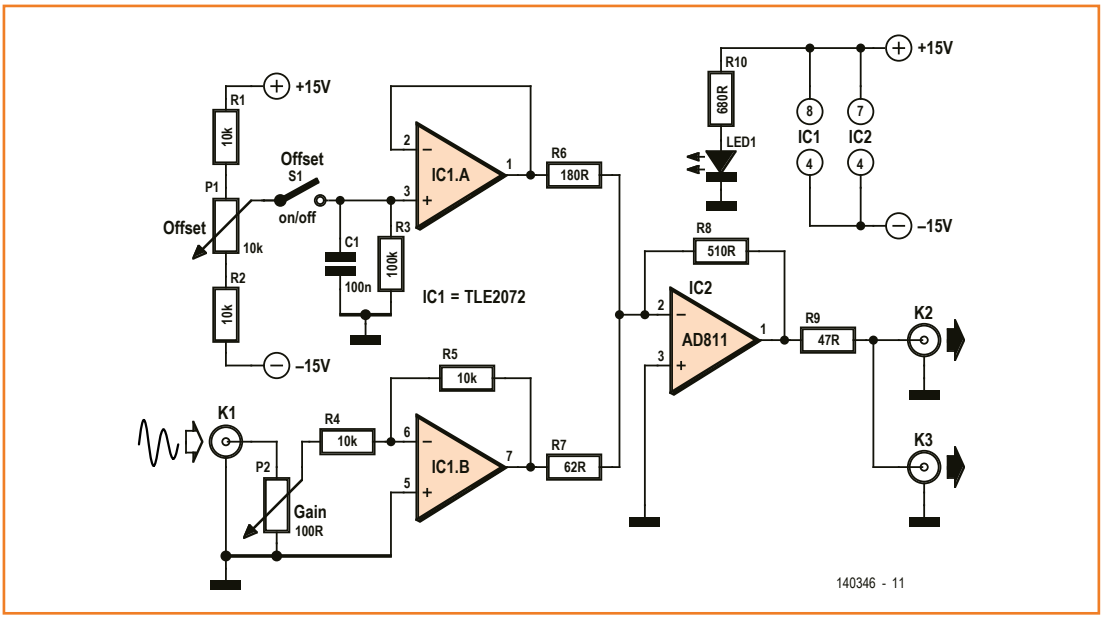


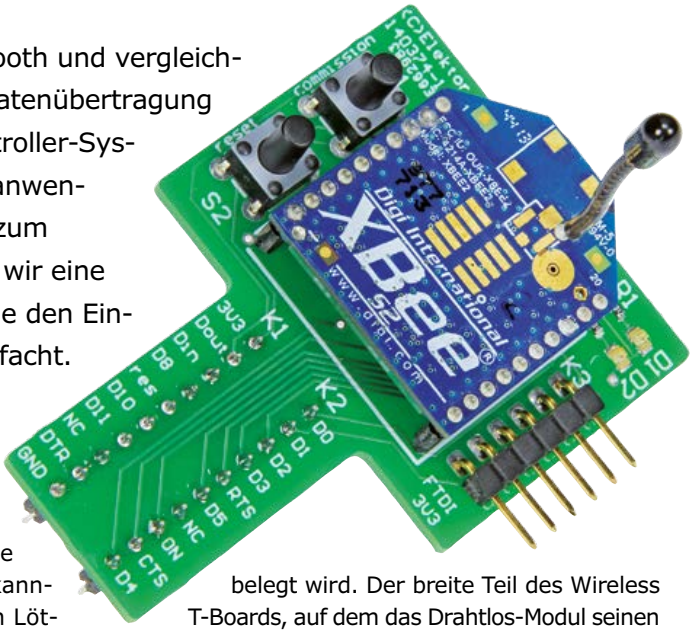
Bild 1.
Schaltplan eines einfachen Signalverstärkers mit Offset-Einstellung.

T-Board Wireless

Für 20-polige XBee-, Bluetooth- und WiFi-Module

Von **Luc Lemmens**
(Elektor-Labor)

Seit Erscheinen von WLAN, Bluetooth und vergleichbarer Technik hat die drahtlose Datenübertragung auch auf der Ebene der Mikrocontroller-Systeme Einzug gehalten. Für Einzelanwendungen und Versuchsaufbauten, zum Beispiel auf Steckplatinen, haben wir eine Modul-Trägerplatine entworfen, die den Einsatz von Drahtlos-Modulen vereinfacht.



Das Wireless T-Board ist in erster Linie dazu gedacht, das Anschlussraster bekannter Drahtlos-Module an das Raster von Lötpunkt- oder Steckplatinen anzupassen. Während hier die Verbindungspunkte im Raster 0,1 Inch = 2,54 mm angeordnet sind, ist bei den Drahtlos-Modulen das Raster 2 mm gebräuchlich. Dort sind 20 Anschluss-Pins auf zwei 10-polige parallele Reihen verteilt. Auf dem Wireless T-Board führen Leitungen von diesen Anschlüssen zu zwei ebenfalls 10-poligen parallelen Stiftkontakt-Reihen im Raster 2,54 mm (100 mil), diese Reihen haben den Abstand 7,62 mm (300 mil).

Die äußeren Platinenumrisse sind so gestaltet, dass auf einer Steckplatine möglichst wenig Grundfläche belegt wird. Der breite Teil des Wireless T-Boards, auf dem das Drahtlos-Modul seinen Platz hat, kann über eine Kante der Steckplatine hinausragen.

Ursprünglich wurde das T-Board Wireless für ein XBee-Modul von Digi International in einem anderen Elektor-Projekt konzipiert. Die Anschlussbezeichnungen in der Schaltung und auf der Platine beziehen sich auf dieses Modul. Der „XBee form factor“ stimmt jedoch mit diversen anderen Drahtlos-Modulen überein. Dazu gehören unter anderem die Typen RN-171-XV, RN41x und RN42x von Microchip sowie die XRF-Module von Ciseco [1]. **Tabelle 1** stellt die Anschlussbelegungen einiger viel verwendeter Drahtlos-Module unterschiedlicher Techniken einander gegenüber. Die dort genannten Module, die nur beispielhaft genannt sind, können auf dem Wireless T-Board ohne Modifikationen platziert werden. Die betriebswichtigen Anschlüsse wie Betriebsspannung und Masse, UART-Leitungen und Reset sind bei den genannten Produkten von Microchip, Ciseco und Digi International pinkompatibel. Beim Einsatz anderer Module ist zu prüfen, ob das T-Board Wireless ohne Modifikationen passt. Das gilt insbesondere für die Anschlüsse der Betriebsspannung, hier sind Abweichungen nicht selten.

Bild 1.
Drahtlos-Module von Digi, Ciseco und Microchip. Diese Module sind eine Auswahl der zum T-Board Wireless passenden Typen.

Schaltung

Das Drahtlos-Modul ist, wie **Bild 2** zeigt, auf dem T-Board von einigen weiteren Komponenten umgeben. Die Kontaktleisten MOD1, K1 und K2 passen das Anschlussraster 2 mm des Drahtlos-Moduls an das Anschlussraster 2,54 mm an.

Auf dem T-Board befinden sich ferner eine Steckleiste für ein USB/TTL-Schnittstellen-Kabel (K3), ein Spannungsregler 3,3 V (IC1), zwei Drucktaster (S1 und S2) und zwei LEDs (D1 und D2). Aus dem Schaltbild ist ersichtlich, dass auch die beim XBee-Modul mit „NC“ (Not Connected) bezeichneten Anschlüsse mit den zugehörigen Kontakten auf K1 und K2 verbunden sind. Bei einigen Produkten anderer Hersteller sind diese Anschlüsse mit Funktionen belegt.

Die Drucktaster S1 und S2 (Reset und Commissioning) sind für den regulären Betrieb entbehrlich. Reset wird gelegentlich beim Flashen von Firmware und beim Entwickeln von Applikationen benötigt. Commissioning dient beim XBee-Modul zur Diagnose und Konfiguration von Netzwerken, was ebenfalls nicht täglich vorkommen dürfte. Weiterreichende Informationen zu diesen Themen enthalten die Datenblätter und Application Notes des Herstellers Digi International [2].

Die LED D1 (RSSI) kann für das letzte empfangene Daten-Paket die Signalstärke anzeigen. In den Modulen von Digi International ist diese Option implementiert und aktiviert. Bei Bedarf lässt sich die Anzeige über die Software XCTU deaktivieren. LED D2 wird bei Digi als „Associate LED“ bezeichnet, sie zeigt den Netzwerk-Status und beim Drücken des Tasters „Commissioning“ das Ergebnis der Diagnose an. Beim XRF-Modul von Ciseco blinkt D1 im regulären Betrieb einmal in der Sekunde und D2 signalisiert das Übertragen der Datenpakete.

In **Bild 3** ist das Layout des T-Board Wireless wiedergegeben. Der Spannungsregler, die Widerstände und Kondensatoren wie auch die LEDs sind Ausführungen in SMD-Bauform. Alle SMD-Bauelemente können noch von Hand gelötet werden. Das T-Board Wireless ist unter der Bestellnummer 140374-91 auch aufgebaut und getestet im Elektor-Shop erhältlich. Wenn der Spannungsregler, die Drucktaster und die LEDs weggelassen werden, kann das T-Board Wireless auch als einfacher Adapter für Steckplatinen dienen.

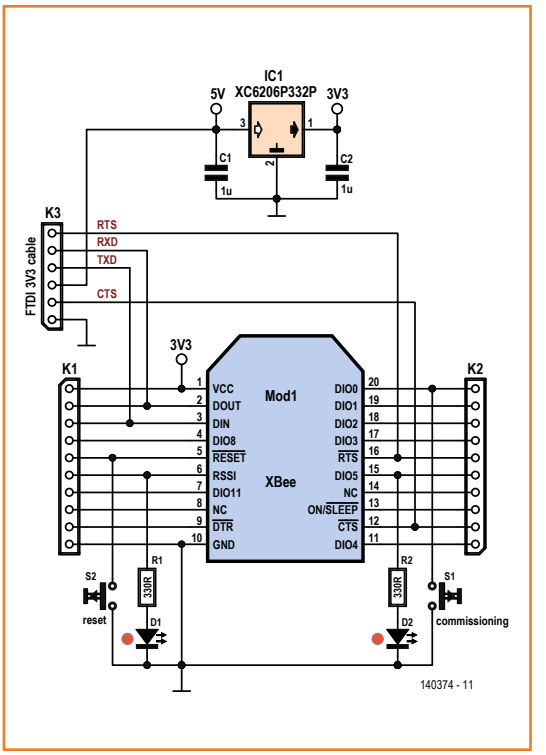


Bild 2.
Auf dem T-Board Wireless sind außer dem Drahtlos-Modul ein Spannungsregler und einige weitere Bauelemente vorhanden.

Tabelle 1. Anschlussbelegungen einiger Drahtlos-Module.

Modul-Typ	Digi XBee XBee	Ciseco XRF ISM (868 MHz)	RN171XV WiFi	RN42XV Bluetooth
1	3V3	3V3	3V3	3V3
2	Dout	Dout	Dout	Dout
3	Din	Din	Din	Din
4	DIO8	RTS	GPIO8	GPIO7
5	#reset	#reset	#reset	#reset
6	RSSI	P1_7	GPIO5	GPIO6
7	DIO11	P1_6	GPIO7	GPIO9
8	NC	P1_5	GPIO9	GPIO4
9	DTR	P1_4	GPIO1	GPIO11
10	GND	GND	GND	GND
11	DIO4	P0_1	GPIO14	GPIO8
12	CTS	CTS	RTS	RTS
13	on/#sleep	on/#sleep	GPIO4	GPIO2
14	NC	NC	NC	NC
15	DIO5	P0_7	GPIO6	GPIO5
16	RTS	P2_0	CTS	CTS
17	DIO3	P2_1	Sensor5	GPIO3
18	DIO2	P2_2	GPIO3	GPIO7
19	DIO1	P2_3	Sensor3	AIO0
20	DIO0	P0_5	Sensor2	AIO1

Korrekte Signalpegel

Auf dem T-Board Wireless befindet sich eine 6-polige Kontaktleiste (K3) für ein USB/TTL-Schnittstellen-Kabel, das mit 3,3 V (!) arbeitet (Elektor-Shop 080213-72). Ein solches Kabel stellt die Verbindung zwischen dem UART des Drahtlos-Moduls und einem PC her, so dass das Modul über den PC konfiguriert werden kann. Das T-Board Wireless wird dann über das Kabel mit Betriebsspannung versorgt. Beim Schnittstellen-Kabel in der Version 3,3 V liegen trotzdem 5 V an der Leitung V_{DD}, für die Versorgung des Drahtlos-Moduls muss ein Spannungsregler vorgeschaltet werden.

Die Typenvielfalt der XBee- und sonstigen Drahtlos-Module ist groß. Auf Internet-Foren wird häufig diskutiert, ob die Eingänge und Ausgänge eines bestimmten Typs tolerant für Signale mit dem Pegel 5 V sind. Tatsächlich gibt es solche Typen, oder es hat sie in der Vergangenheit gegeben. Auf der sicheren Seite ist der Anwender nur, wenn die Signalpegel bei Kopplung mit 5-V-Systemen auf 3,3 V transformiert werden. Und das bedeutet auch, dass das USB/TTL-Schnittstellen-Kabel, angeschlossen an K3, vom Typ 3,3 V sein muss. Ein funktionsgleiches Kabel für den Signalpegel 5 V ist **nicht** geeignet!

Weitere Möglichkeiten

Das Wireless T-Board ist auch eine gute Wahl, wenn XBee-Module konfiguriert oder geupdatet werden müssen. Das USB/TTL-Schnittstellen-Kabel, Spannung 3,3 V, stellt die Verbindung mit dem PC her, auf dem die Konfigurier- und Programmier-Software XCTU von Digi International installiert ist. Mit XCTU lassen sich XBee-Module unkompliziert an die vorgesehenen Applikationen anpassen. Diverse Einstellungen sind auch mit AT-Kommandos über ein beliebiges Terminal-Programm zugänglich, bei XBee-Modulen ist jedoch XCTU einfacher in der Handhabung.

Viele Anwendungen der genannten Drahtlos-Module nutzen ausschließlich den UART, sie senden und empfangen drahtlos Daten in seriellem Format. In dieser Betriebsart genügt es, das Drahtlos-Modul über die Kontaktleiste K3 mit dem Mikrocontroller-System oder mit dem PC zu verbinden. Die übrigen digitalen Leitungen und die A/D-Ports haben dann keine Funktion. Wenn es darauf ankommt, Platz zu sparen, kann der Platinenteil abgetrennt werden, auf dem sich K1 und K2 befinden.

(140374)gd

Weblinks

[1] www.elektor.de/xrf-wireless-module-140403-91

[2] www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/

[3] www.elektor-magazine.de/140374

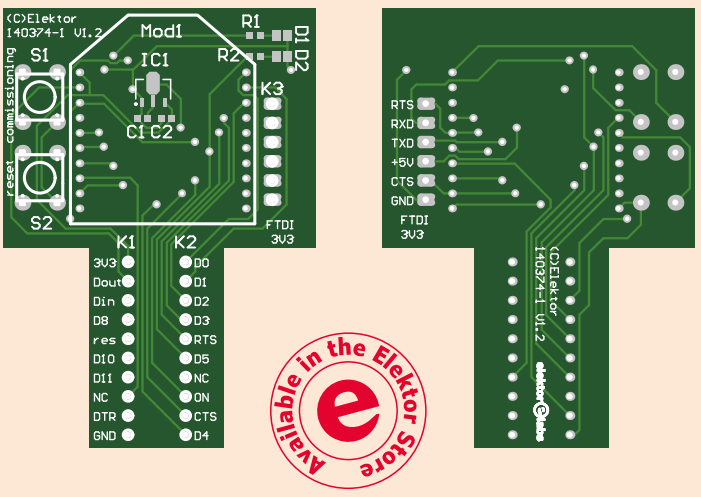


Bild 3.
Platinenlayout des T-Board Wireless. Die Platine ist auch bestückt und getestet (ohne Drahtlos-Modul) im Elektor-Shop erhältlich.

Stückliste

Widerstände:
R1,R2 = 330 Ω 5 % 100 mW, SMD 0603

Kondensatoren:
C1,C2 = 1 µ/10 V 10 %, SMD 0603

Halbleiter:
D1,D2 = LED rot 20 mA, SMD 0805
IC1 = XC6206P332PR, Spannungsregler 3,3 V, SMD SOT-89-3

Außerdem:
K1,K2 = Stiftkontaktleiste 10-polig, Raster 2,54 mm
K3 = Stiftkontaktleiste 6-polig, Raster 2,54 mm
Mod1 = 2 Kontaktleisten für Modul 10-polig, Raster 2 mm
S1,S2 = Drucktaster für Platinenmontage, 6 · 6 mm
Drahtlos-Modul, z. B. XRF-Modul von Ciseco [1] oder XBee ZB-Modul von Digi
Leere Platine 140374-1 oder:
Aufgebaute Platine 140374-91 (ohne Drahtlos-Modul)

Beep

Der Logiktester mit Pfiff

Von **Clemens Valens**
(Elektor-Labor)

Selbst wenn Sie einen ausgebufften Logikanalysator besitzen und damit völlig zufrieden sind, ist ein solches Gerät völlig überdimensioniert, wenn man einfache Schaltungen wie zum Beispiel eine Add-on-Karte für Arduino oder Raspberry Pi testen möchte. Hier kommt Beep ins Spiel!

Sicherlich kann man zur Prüfung einfacher Digitalschaltungen ein gemeines Digitalmultimeter verwenden, aber es ist doch mühsam, immer die Anzeige im Auge zu behalten, während man die Messspitze in „gefährlicher“ Umgebung an einen kleinen IC-Pin halten muss. Und bedeutet 0 Volt im Display nun, dass das geprüfte Signal wirklich Low ist, oder ist die Messspitze einfach nur vom Pin abgerutscht? Diese und andere Unsicherheiten haben uns veranlasst, einen einfachen Logiktester zu entwerfen.

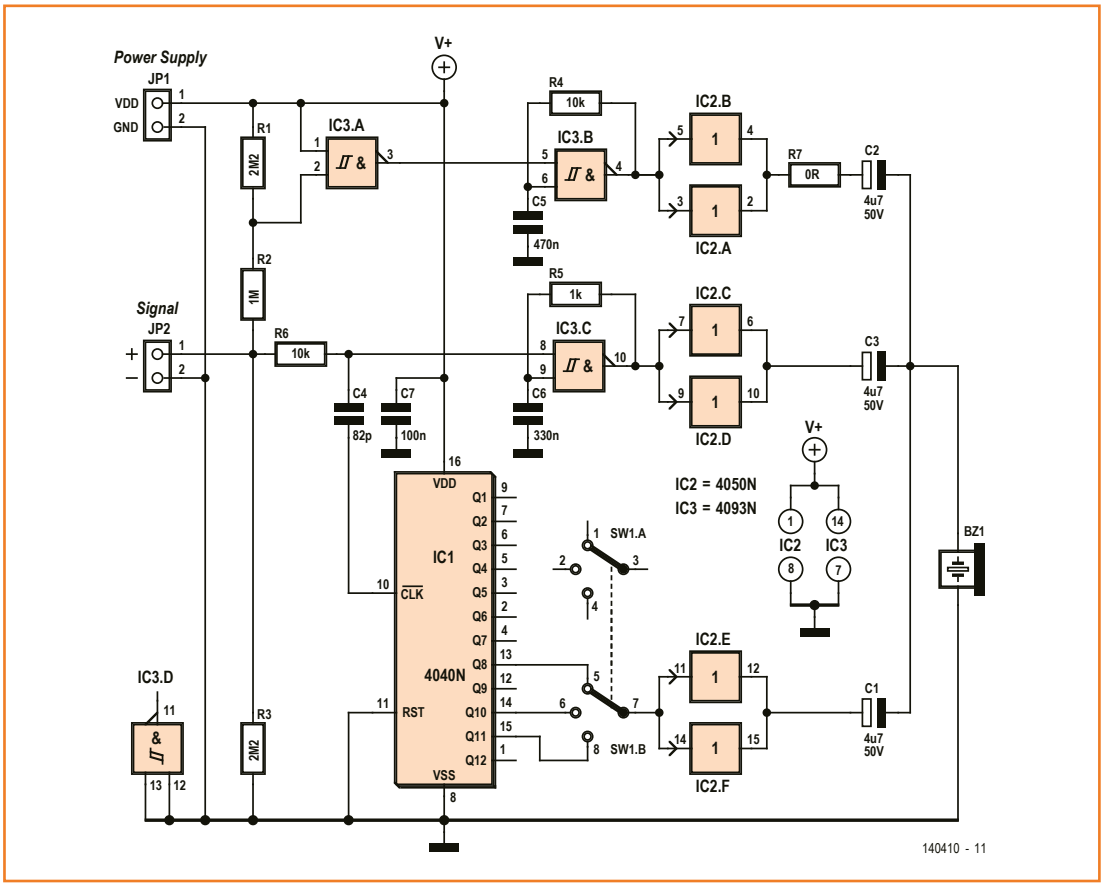


Bild 1.
Die Beep-Schaltung. R7 ist eine Drahtbrücke.

Der Klang der Hertzen

Beep, wie wir unseren Logiktester nennen wollen, verwendet Töne, um Signalpegel anzuzeigen, so dass Sie sich voll auf die Messarbeit konzentrieren können. Beep erzeugt einen niederfrequenten Ton, wenn der Signalpegel Low ist und einen hohen, wenn auch das Signal High ist. Ist der Signalpegel nicht definiert, weil der Ausgang hochohmig ist, es sich um einen unbeschalteten Eingang handelt oder wir es mit einem Open-Collector-Ausgang mit fehlendem Pull-up-Widerstand zu tun haben, schweigt Beep. Viele Signale wechseln ihren Pegel so schnell, dass ein entsprechendes Audiosignal viel zu hoch wäre, um es hören zu können. Auch hierfür hält Beep eine Lösung bereit!

Oszillieren und dividieren

Wie schafft Beep dies alles nur ohne Mikrocontroller? Wie **Bild 1** zeigt, geht das problemlos mit zwei Oszillatoren und einem Frequenzteiler! Es gibt einen Oszillator (IC3.B, R4, C5) für den tiefen und einen zweiten (IC3.C, R5, C6) für den hohen Ton. Die Frequenzen werden durch R4/C5 (Low) und R5/C6 (High) bestimmt. Da das zu untersuchende Signal ja nicht gleichzeitig Low oder High sein kann, ist jeweils nur einer der Oszillatoren aktiv. Deren Ausgänge können deshalb problemlos parallel geschaltet werden und den Summer BZ1 ansteuern.
Welcher Oszillator aktiv ist, hängt vom Pegel an Pin 1 von JP2 ab. Wenn dort und auch an Pin 8 des NAND-Gatters IC3.C der Pegel Low ist, liegt das Gatter still, der Ausgang ist, wie die Wahrheitstafel zeigt, immer High, so dass der Oszillator um IC3.C nicht schwingt (siehe Kasten). Am Eingang von IC3.A liegt eine Spannung, nicht ganz 0 Volt, aber niedrig genug, damit der Ausgang des Gatters (Pin 3) High wird. Der Oszillator um IC3.B ist durch den High-Pegel an Pin 5 freigegeben und oszilliert. Ein niedriger Ton ist zu hören. Das Gegenteil geschieht, wenn das Signal an JP2 einen High-Pegel aufweist. Nun sind die beiden Eingänge von IC3.A High, wodurch sein Ausgang auf Low geht und den Niederfrequenzoszillator IC3.B blockiert. Pin 8 von IC3.C ist dagegen High, so dass nun dieser Oszillator freigegeben ist. Es ertönt ein hoher Ton.

Wenn Beep einem nicht angeschlossenen Eingang, einem Tri-State-Ausgang (mit hoher Impedanz) oder einem Open-Collector-Ausgang ohne Pull-up-Widerstand begegnet, hat die Testschaltung nicht die Kraft, den hochohmigen Eingang (beachte die hohen Werte von R1, R2 und R3) auf ein gut definiertes High oder Low zu bringen. Der Spannungsteiler sorgt dafür, dass sowohl an Pin 5 (Pin 1 und 2 High) als auch an Pin 8 von IC3 ein Low-Pegel anliegt und so beide Oszillatoren gesperrt sind. Der Summer bleibt stumm!

Ist der Pegel an JP2 nicht konstant, sondern wechselt beständig, wird ein „zweifarbiger" Ton zu hören sein. Mit steigender Frequenz des Eingangs-

Oszillatorbetrieb

Nehmen Sie an, Pin 5 von IC3.B sei gleichbleibend High. Beim Einschalten ist C5 entladen und hält deshalb Pin 6 auf Low. Da wir es mit einem NAND-Gatter zu tun haben, wird sein Ausgang Pin 4 dann High, so dass C5 über R4 geladen wird. Wenn die steigende Spannung an C5 einen bestimmten Schwellwert überschreitet, nimmt IC3.B an, dass der Eingang Pin 6 logisch High ist. Jetzt sind beide Eingänge High, so dass der Ausgang auf Low kippt. C5 entlädt sich nun über R4, bis die Spannung an Pin 6 so gering ist, dass das Gatter sie als Low interpretiert und der Ausgang wieder auf High zurückkippt. Wir sind wieder da, wo wir angefangen haben, der Oszillator schwingt! Die Frequenz des Oszillators hängt von den Werten des Widerstands und des Kondensators als auch von den unteren und oberen Grenzwerten des Schmitt-Triggers ab. Aufgrund der NAND-Funktion kann der Ausgang niemals Low werden, solange ein stabiles Low am Eingang Pin 5 anliegt. Der Oszillator ist dann wirksam deaktiviert.

Tabelle 1. Wahrheitstafel eines NAND-Gatters (not-AND; invertiertes AND)

Eingang A (Pin 5)	Eingang B (Pin 6)	Ausgang (Pin 4)
0	0	1
0	1	1
1	0	1
1	1	0

Weblinks

[1] Beep-Projektseite: www.elektor.de/140410

[2] ELPP: https://github.com/ElektorLabs/PreferredParts

signals wechselt auch der Ton immer schneller, bis der Punkt kommt, an dem die Oszillatoren nicht mehr folgen können und der Ton ganz verstummt. Jetzt kommt der Frequenzteiler IC1 ins Spiel. Er teilt die Frequenz des Eingangssignals je nach Stellung des Schalters SW1.B durch 256, 1024 oder 2048. Damit können Sie Frequenzen bis etwa 4 MHz abhören, vorausgesetzt, Ihr Gehör schafft die magische Grenze von 20 kHz.

Die Ausgänge der Oszillatoren sind zu schwach, um den Summer direkt ansteuern. Deshalb sind sie mit je einer Ausgangsstufe ausgestattet, bestehend aus zwei parallelen Puffern, die genügend Schwung für den Summer bereitstellen. Auch der Frequenzteiler besitzt eine solche Ausgangsstufe, um sicherzustellen, dass der Ton immer die gleiche Lautstärke besitzt.

Beep auf der Platine und im Betrieb

Beep wird bequemerweise von der zu prüfenden Schaltung über den Anschluss JP1 versorgt. Leere Batterien ausgeschlossen! Wenn Sie CMOS-ICs verwenden, darf die Versorgungsspannung bis zu 12 V betragen, im Fall von TTL-ICs (wie dem 74HCT4040) darf die Versorgungsspannung 5 V nicht überschreiten. Deshalb sind TTL-ICs nicht empfehlenswert.

Der Aufbau von Beep ist nicht übermäßig kompliziert. Wir haben eine Platine entworfen, die Sie im Elektor-Shop bestellen können. Die Platine passt prima in ein Hammond-Gehäuse 1593D.

Bild 2 zeigt den Bestückungsplan und die Stückliste. Es werden nur Durchsteck-Bauteile verwendet, die meisten aus der Bibliothek **Elektor Labs Preferred Parts** (ELPP) [2], so dass ihre Footprints und Größen gut definiert und sie überdies einfach zu beschaffen sind. Die ICs sind Standard-Logikbausteine. Beim Schiebeschalter kann die Beschaffung ein wenig schwerer werden, aber Sie können ihn auch durch eine Stiftleiste und einen Jumper ersetzen.

(140410)

Bild 2.
Auf dieser Platine ist Beep zu Hause. Sie passt genau in das vorgeschlagene Hammond-Gehäuse.

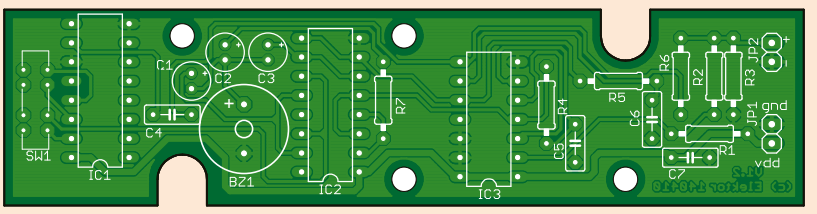
Stückliste

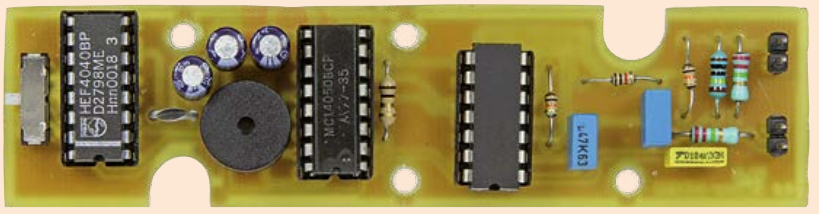
Widerstände:
alle 5%, 0,25 W
R1,R3 = 2,2 M
R2 = 1 M
R4,R6 = 10 k
R5 = 1 k
R7 = 0 Ω (oder Drahtbrücke)

Kondensatoren:
Raster 5 mm
C1,C2,C3 = 4µ7, 50V (2 mm Raster)
C4 = 82 p
C5 = 470 n
C6 = 330 n
C7 = 100 n

Halbleiter:
IC1 = HEF4040BP (oder Äquivalent)
IC2 = CD4050BE (oder Äquivalent)
IC3 = HEF4093BP (oder Äquivalent)

Außerdem:
SW1 = Schiebeschalter, 2-polig, 3 Positionen (RS 702-3568 oder ähnlich)
BZ1 = Summer 12 mm Durchmesser
JP1,JP2 = 2-polige Stiftleiste
2 DIP-16 IC-Fassung (IC1 und IC2)
1 DIP-14 IC-Fassung (IC3)
Gehäuse Hammond 1593D
Platine 140410-1 im Elektor-Shop





Tristate-Level-Shifter

Von **Nicolas Boullis** (F)

Mit dem SRAM 23K256 von Microchip, ein statisches RAM mit SPI-Schnittstelle, ist der Speicher eines Arduino-Systems schnell aufgerüstet. Allerdings tritt das Problem auf, dass die Signalpegel von Arduino und SRAM nicht übereinstimmen. Während das Arduino-System mit 5 V arbeitet, beträgt die nominale Betriebsspannung des SRAM 3,3 V, sie darf 3,6 V nicht übersteigen.

Im Web kursieren diverse Lösungsvorschläge, die meisten erscheinen wenig professionell. In diese Kategorie gehört der Vorschlag, das SRAM einfach unbesehen an +5 V zu legen. Da die Spannung +5 V weit oberhalb des im Datenblatt propagierten zulässigen Arbeitsbereichs liegt, dürfte die Lebenserwartung des 23K256 drastisch sinken.

Arduino-Systeme stellen eine stabilisierte Gleichspannung 3,3 V bereit, die für den Betrieb des 23K256 genutzt werden kann. Die SPI-Signale CS, SCK und MOSI lassen sich unkompliziert über Spannungsteiler anpassen, denn ihr Weg verläuft unidirektional vom Arduino-System zum 23K256. Die Leitung MISO (Serial Output) erfordert weiter gehende Maßnahmen, denn sie kann auch den hochohmigen Zustand annehmen. I²C-Leitungen werden häufig mit N-Kanal-MOSFETs und Pullup-Widerständen angepasst, doch damit sind die Zustände „Logisch 1“ und „Hochohmig“ nicht mehr unterscheidbar. Außerdem werden die Signalflanken durch die unvermeidlichen parasitären Kapazitäten verschliffen. Eine deutlich bessere Lösung stellt der integrierte „Voltage Level Translator“ TBX0104 von Texas Instruments dar. Leider ist ein definitionskonformer hochohmiger Zustand auch mit ihm nicht herstellbar.

Die hier angegebene Schaltung, aufgebaut mit gängigen diskreten Standard-Bauelementen, löst das Problem kompromisslos. Wenn Ausgang SO des seriellen RAMs hochohmig ist, leiten T1 und T2 gerade. Infolge des Spannungsteilers R1/R7/R8/R2 liegt die Basis-Emitter-Spannung von T1 und T2 bei 0,65 V. Der Strom durch T1 beträgt ebenso wie der Strom durch T2 etwa 0,45 mA (ungefähr 0,15 V/330 Ω). An R5 und R6 fallen Spannungen nahe 0,45 V ab, so dass T3 und T4 sperren: Ausgang MI ist jetzt hochohmig. Geht das Signal am Schaltungseingang auf logisch 1, steigt die Spannung an der Basis von T2, während T1 vollständig sperrt. Der Strom, der durch T2 fließt, beträgt mehr als 2 mA, so dass T3 infolge des Spannungsabfalls an R5 leitet und Ausgang MI an +5 V (logisch 1) legt. Bei logisch 0 am Eingang verhält sich die Schaltung komplementär. Jetzt leiten T1 und folglich T4, Ausgang MI wird von T4 nach Masse (logisch 0) gezogen.

Die Widerstände R7 und R8 am Eingang begrenzen die Basisströme von T1 und T2, C1 und C2 verbessern das Schaltverhalten. Schottky-Diode D1 verhindert ebenso wie Schottky-Diode D2, dass der zugehörige Transistor in die Sättigung gerät.

Beim Versuchsaufbau auf einer Steckplatine hat der Tristate-Level-Shifter zuverlässig Signale mit Frequenzen bis über 3 MHz transformiert.

(140224)gd

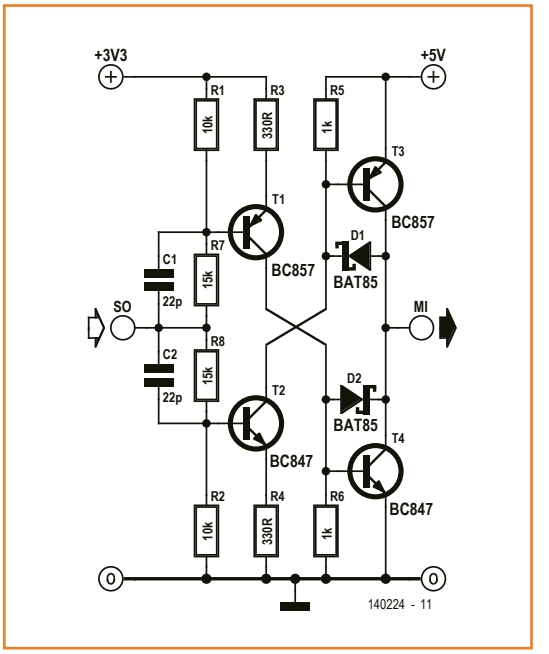


Bild 1.
Der Signalpegel 3,3 V wird auf 5 V transformiert, auch der hochohmige Zustand wird korrekt übertragen.

Microchip/Hillstar-Entwicklungskit – ein Überblick

Touch-Control und Gestensteuerung in einem Chip

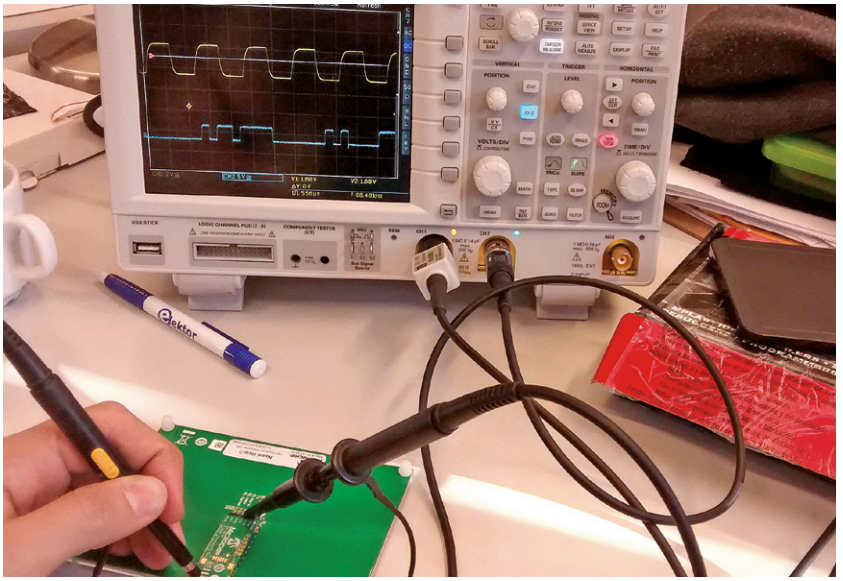


Bild 1.
Auf dem Scope: oben das Trägersignal in Tx, unten der SDA-Datenstrom. Links die Hand des „Dirigenten“, rechts, auf dem Karton das 3D-Touchpad aus dem Elektor/Microchip-Produktpaket.

Von **Jaime González-Arintero Berciano**
(Elektor-Labor)

Sie wollten immer schon ein Orchester(halb)leiter sein? Wild gestikulierend, um das Orchester im Takt zu halten? Ein Orchester können wir zwar nicht bieten, aber mit Microchips MGC3130 GestIC-Controller und dem betriebsbereiten Entwicklungskit kommen Sie dem Ziel sehr nahe: ein Controller wie aus dem ScienceFiction-Film, touchfree und touch in einem und bereit zum Einbetten in eine Applikation.

Hände hoch!

Das Entwicklungs-Kit von Microchip/Hillstar ist ein ausgezeichneter Ansatz, um eine berührungslose Gestensteuerung für Mikrocontroller-Systeme und PCs zu entwickeln. Meine Freunde in der Redaktion hatten bereits in der November- und Dezemberausgabe [1] und [2] darüber berichtet. Jan hat mir das Kit für einen Überblicks-Artikel überlassen. Das Steuerpad ist eine vierlagige Platine mit den Elektroden, einem Sender und vier Empfängern. Eine kleine Platine mit dem MGC3130-Chip ist direkt auf die 7-polige Stiftleiste (sechs Elektroden und Masse) gesteckt. Dieser Chip kümmert sich um all die magischen Dinge, die Erzeugung eines Trägersignals für die Senderelektrode, sowie die Konditionierung, Digitalisierung und Verarbeitung der Signale, die von den fünf Empfängern aufgefangen werden. Die Informationen stehen dann als I²C-Datenstrom zur weiteren Nutzung bereit. Für die Kommunikation mit einem PC bietet das Entwicklungs-Kit auch eine I²C-zu-USB-Brücke, die gleichzeitig als 5/3,3-V-Konverter fungiert, da dies die Betriebsspannung des MGC3130-Chips ist. Das Kit enthält außerdem vier Styroporblöcke und ein Stück Kupferfolie für erweiterte Kalibrierungszwecke. Nach besonderer Vereinbarung mit Microchip bieten wir im Elektor-Store das Dev Kit von Hillstar zusammen mit einem fertig montierten Touchpad als Paket [3] an; und dies preiswerter als bei Microchip direkt.

Zuerst Lesen, dann loslegen? Na klar...

Ich gestehe, „Erst lesen und dann“ ist nicht gerade mein Fall. Doch enden meine Fehler meist glücklich und ich habe etwas gelernt. Ist es nicht genau das, was Elektronik bedeutet?
Da ich mich mit dieser Art von Designs schon ein wenig auskannte, beschloss ich, mit dem Spielen sofort zu beginnen. Ich wollte dem Kit ohne Treiber und Software, aber mit dem Oszilloskop zu Leibe rücken. Zuerst wollte ich die Vorgänge im MGC3130-Modul überprüfen. Ich nahm an, Tx stehe für Sender. ¡Olé! (ich bin Spanier), sofort hatte ich ein schönes 88-kHz-Signal. Es sollte im

Bereich von 100 kHz liegen, also war alles in Ordnung hier. Ich holte einen Kaffee und kam zum Labortisch zurück, da merkte ich, dass das Signal irgendwie auf 103 kHz sprang... was? Es stieg bis 115 kHz und ging dann zurück auf 88 kHz. Ich raufte meine Haare und beschloss, doch einmal einen Blick in die Dokumentation zu werfen. Und ich erfuhr, dass der MGC3130 automatisch die Sendefrequenz in Abhängigkeit von den „Außengeräuschen“ ändert! Hexenwerk!

Was passiert, bevor die Signale verarbeitet werden, ist aber kein Hexenwerk. Die Senderelektrode produziert ein elektrisches Feld. Wird dieses Feld von der Hand oder einem Gegenstand gestört, werden die Variationen von den Empfangselektroden erkannt. Wir haben zehn Kapazitäten zu berücksichtigen: je fünf Rx-Elektroden zu Masse und zum Sender (Tx). Diese Informationen werden vom Chip verwendet, um die Position des Fremdkörpers zu bestimmen und diese Informationen in einem I²C-Datenstrom zu verpacken, der wunderschön auf dem Scope angesehen werden kann.

Danach habe ich (endlich) die Dokumentation richtig gelesen und die mitgelieferte Aurea-Software installiert, die eine umfassende und ordentliche Test&-Design-Suite darstellt. Man kann dann das gesamte System von Grund auf gestalten, alle Parameter ändern, neu kalibrieren und alle Einstellungen in eine Bibliothek packen, die dann auf den Chip übertragen wird. Ich fand bei Aurea auch kristallklare Beweise für den Tx-Frequenzsprung. Diese Software zusammen mit allen Unterlagen und Referenzen finden Sie bei [4]. Die Informationen zum MGC3130-GestIC sind weitreichend und gut angeordnet.

Touch me oder nicht

Mit mehreren Konfigurationsassistenten ist das Herumspielen mit dem Aurea-GUI angenehm und intuitiv. Einige Aspekte sind erwähnenswert, vor allem für diejenigen, die nicht der Gewohnheit des Handbücher-Lesens verfallen sind.

Das Pad ist sehr empfindlich, so ist sorgfältig zu prüfen, ob es keine Störquellen in der Nähe gibt, da dies den Betrieb beeinträchtigen würde. Auch wenn das Pad an der Unterseite abgeschirmt ist: Wenn Sie es flach auf dem Schreibtisch legen, wird das Feld von allem, was sich darunter bewegt, gestört, zum Beispiel von den Beinen oder der Katze.

Abgesehen von dem berührungslosen GestIC-HID bietet das Pad auch eine gewöhnliche Touchsteuerung. Es arbeitet als Multi-Touch-Trackpad und kann bis zu fünf (!) Finger gleichzeitig erkennen. Das Aurea-GUI enthält einen Assistenten zur Parametrierung für eigene Zwecke. Hier kommen die faszinierenden Hartschaumblöcke ins Spiel. Sie müssen den großen mit der Kupferfolie aus dem Kit umwickeln und ein dünnes Kabel anlöten, um es mit Masse zu verbinden. Oder Sie halten das Kabel fest, solange Ihre Füße den Boden berühren. Das gesamte Verfahren wird im Handbuch erklärt, und es macht Spaß, da es um Bruchteile von Picofarad geht. Dieser „Kupferziegelstein“ und die anderen Blöcke sind für die Elektroden-„Gewichtung“ und die Linearisierung des elektrischen Feldes zuständig (was ich jetzt bei meinen Versuchen gelernt habe).

Das Hillstar-Pad besteht aus einer nackten Platine mit einer transparenten Kunststoffdeckschicht und ist mit all den Software-Tools an Embedded-Entwickler gerichtet. Das Elektor-Angebot [3] enthält auch ein fix und fertiges, in sich geschlossenes 3-D-Pad in einem kommerziellen Gehäuse für alle, die eine Anwendung für einen PC oder ein Tablet entwickeln wollen.

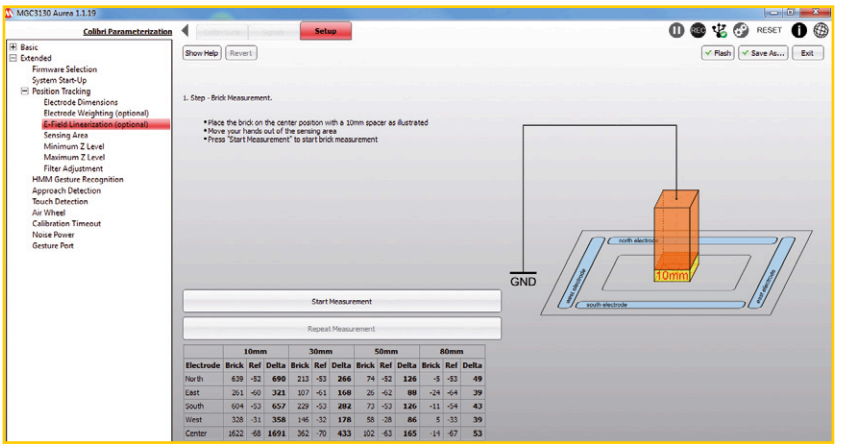


Bild 2. Präzisionskalibrierung und Parametrierung des GestIC-Systems mit den Styroporblöcken, die ebenfalls im Entwicklungskit von Hillstar enthalten sind.

Die Dokumentation empfiehlt, leitfähigen Kunststoff zu vermeiden und vermeldet, dass schwarzer Kunststoff leitfähigen Kohlenstoff enthalten kann. Aber das Touchpad ist ... schwarz. Ist es vielleicht frei von Kohlenstoff? Ich bin weiter neugierig.

(140434)

Weblinks

[1] 3D-Steuerung für µC und PC: E lektor November 2014

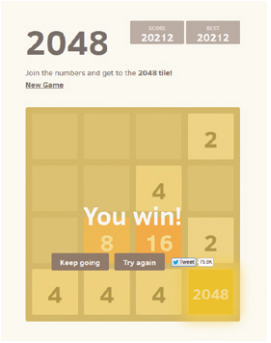
[2] GestIC & 3D-TouchPad-Kurs (1): Elektor Dezember 2014

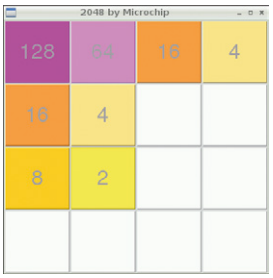
[3] www.elektor.de/microchip-dm160218-hillstar-development-kit-and-dm160225-3d-touchpad

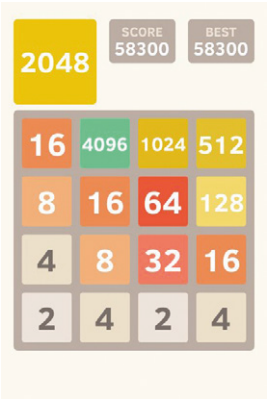
[4] http://j.mp/MGC3130

GestIC & 3D TouchPad Kurs (2)

Gestikulieren und gewinnen







Von Thomas Lindner und **Hung Nguyen**
(Microchip GestIC®-Team, Deutschland)

Diesen Monat geht es weiter mit dem Anschluss des MGC3130-ICs an den Raspberry Pi. Und wir spielen das extrem trendige Spiel 2048 (ja, das ist 2^{11}). Wenn Sie neu in diesem Kurs sind, sollten Sie beachten:

1. Dies ist ein Arbeitsblatt, wir halten es kurz und prägnant;
2. Die Hardware ist exklusiv über den Elektor-Shop (siehe [1]) erhältlich;
3. Sie sollten die früheren Artikel [2] und [3] kennen.

Zutaten

Beginnen Sie mit der Überprüfung der Hardware-Verbindungen, so wie unter [3] beschrieben. **Tabelle 1** fasst die I/O-Verbindungen zusammen.

Sie benötigen:

1. die neueste Raspbian-Version (Raspbian Debian Wheezy), Version: September 2014, Release date: 2014-09-09.
2. Python-Programm *2048_with_Hillstar_Gesture_Port.py* (mit dem Bugfix für das Port-Mapping). Bei [4] erhältlich.
3. MGC3130 FW und Parametrierung: *Hillstar Gesture Port V1.2.4 to Raspberry Pi Demo 2048.enz* (Version 1.2.4). Bei [4] erhältlich.

Python-Vorbereitungen

Sie können sofort mit der IDLE Integrated Development Environment (IDE) beginnen, die das Tkinter GUI Toolkit enthält.
Die Raspbian-Installation enthält IDLE for Python 2.7 und Python 3. Unsere GestIC-Demo basiert auf Python 2.7. Starten Sie die IDE als Root im Linux-Terminal, um schnellen Zugriff auf die GPIO-Hardware zu erhalten (siehe **Bild 1**).

```
# sudo idle
```

Das erste Python-Programm kann direkt aus der Shell ausgeführt werden:

```
>>> print “Hello Python”
```

Der Zugriff auf die GPIOs des RPis ist genauso einfach. Laden Sie das vorinstallierte GPIO-Python-Modul, initialisieren Sie die Pins und lesen Sie die Signale in ihr Programm (**Listing 1**) ein.

Listing 1

```
>>> import RPi.GPIO as GPIO
>>> GPIO.setmode(GPIO.BCM)
>>>
>>> GPIO.setup(17, GPIO.IN, pull_up_down
   = GPIO.PUD_DOWN)
>>> GPIO.setup(18, GPIO.IN, pull_up_down
   = GPIO.PUD_DOWN)
>>>
>>> while true:
>>>     if GPIO.input(17)==1:
>>>         print “GPIO 17 is HIGH”
>>>     elif GPIO.input(18)==1:
>>>         print “GPIO 18 is HIGH”
```

Wenn die Hardware korrekt eingerichtet wurde, erscheint (wie in **Bild 2** zu sehen) bei jeder Geste eine neue Zeile.

2048 für RPi

2048 ist ein Spiel, das im März 2014 vom 19-jährigen italienischen Webentwickler Gabriele Cirulli erstellt wurde. Das Ziel ist es, vornummerierte Kacheln auf einem 4x4-Raster in einer Weise zu verschieben, dass eine Kachel mit der Nummer 2048 erscheint. Gabrieles Spiel hat in weniger als einer Woche vier Millionen Besucher angezogen. Sie erfahren die Regeln unter [5]. 2048 ist nicht das Maximum, gewiefte Elektor-Leser sollten in der Lage sein, die höchstmögliche Kachel 131.072 oder 2^{17} zu erreichen.
Das Python-Programm, das wir für die Umsetzung auf dem RPi verwenden, stammt aus Hrvojes

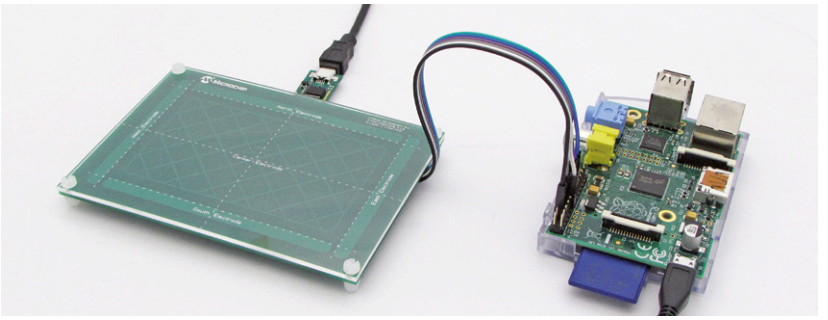
Tabelle 1.		
MGC3130	**RPi**	**Geste**
EIO1	GPIO17	← von Ost nach West
EIO2	GPIO18	↑ von Süd nach Nord
EIO3	GPIO25	→ von West nach Ost
EIO6	GPIO23	↓ von Nord nach Süd

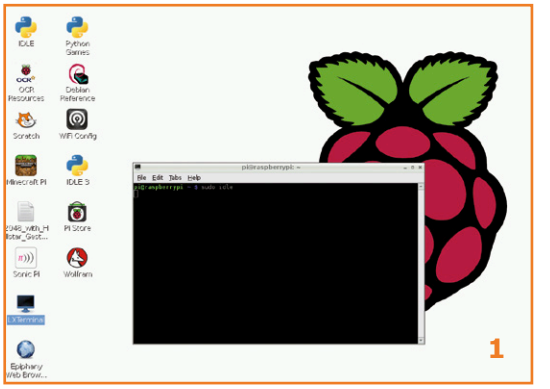
Blog [6]. Wir kombinieren das Programm mit den Gesture-Eingaben des MGC3130-Sensors.
Lassen Sie uns nun alle Teile zusammenstellen. Der vollständige Code kann von der Elektor-Webseite dieses Kursteils heruntergeladen werden [4]. Kopieren Sie die Datei auf den RPi und öffnen Sie sie aus der IDLE-Python-Shell (**Bild 3**).
Die Datei öffnet in einem Editor-Fenster, Sie können das Programm durch „Run“ im Menü oder einen Druck auf F5 starten. Spielen Sie das Spiel mit Gesten in den vier Himmelsrichtungen über der Hillstar-Elektrode (**Bild 4**). Es macht Spaß!

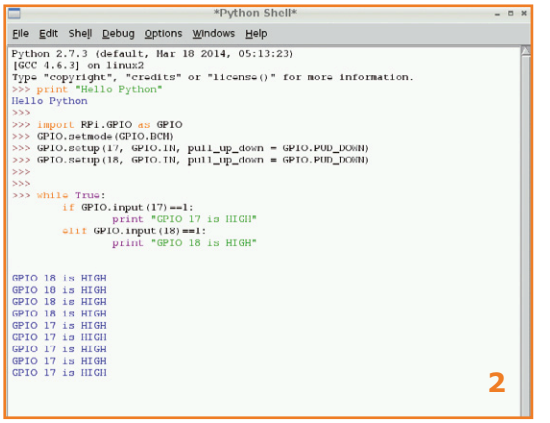
Im Wartestand

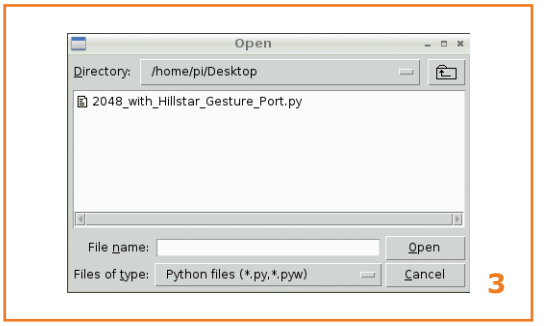
Das nächste Mal werden wir 2048 mit dem vorgefertigten 3D-Touchpad spielen, das ebenfalls im Elektor/Microchip-Bundle enthalten ist [1]. Wir werden uns auch mit der USB-Schnittstelle und dem 3D-Touchpad Software Development Kit (SDK) beschäftigen und das 3D-Touchpad am USB des RPi anschließen.

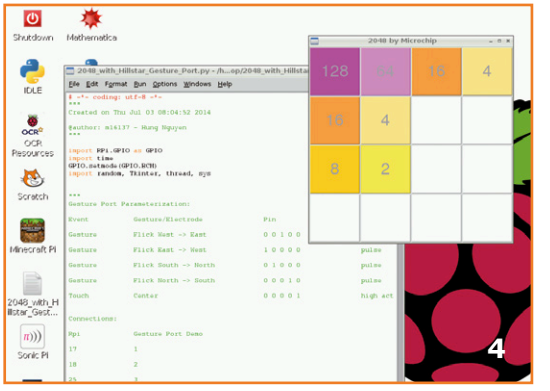
(140513)











Weblinks

[1] Microchip Hillstar Development Kit und 3D-Touchpad Bundle: www.elektor.de/microchip-dm160218-hillstar-development-kit-and-dm160225-3d-touchpad

[2] 3D-Steuerung für µC und PC: Elektor November 2014

[3] GestIC & 3D TouchPad Kurs (1): Elektor Dezember 2014

[4] 2048 Spiel auf Rpi, gepatched für MGC3130: www.elektor-magazine.de/140513

[5] 2048 Spielregeln: http://de.wikipedia.org/wiki/2048_%28Computerspiel%29

[6] Hrvoje’s Blog: http://blog.hrvoje.org/blog/2014/09/20/a-simple-2048-clone-in-python

DesignSpark Tipps & Tricks

Multiple Gates & Gehäuse

Von **Neil Gruending**
(Kanada)

In der letzten Folge ging es um die Erstellung einfacher Bauelemente. Jetzt dreht sich alles um Bauelemente mit komplexem Innenleben wie Logik-ICs.

Bislang ging es lediglich um Bauelemente mit einem einzigen Symbol und einem einzigen Gehäuse, was für die meisten, aber nicht alle Bauelemente zutrifft. Nun ist der komplexere Rest mit mehr als einem Gate und mehr als einem Gehäuse an der Reihe.

Bauelemente mit multiplen Gates

In DesignSpark benötigt jedes Bauteil mindestens ein Symbol, das es im Schaltplan repräsentiert. In den meisten Fällen reicht das völlig, aber z.B. Logik-ICs oder Mehrfach-Opamps beinhalten mehrere Teile. DesignSpark muss hierfür also mehr als ein Symbol pro Bauelement verknüpfen. Üblicherweise wird hier wie bei Opamps das gleiche Grundsymbol mehrfach verwendet – Hauptsache alle im Layout definierten Pins werden angeschlossen.

Bild 1.
Component Wizard.

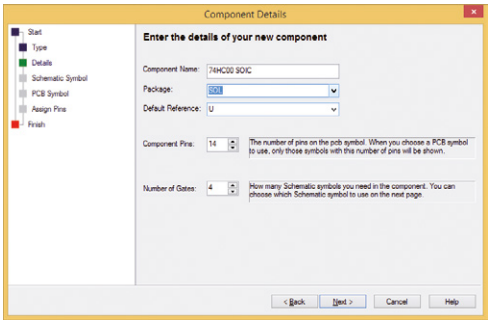
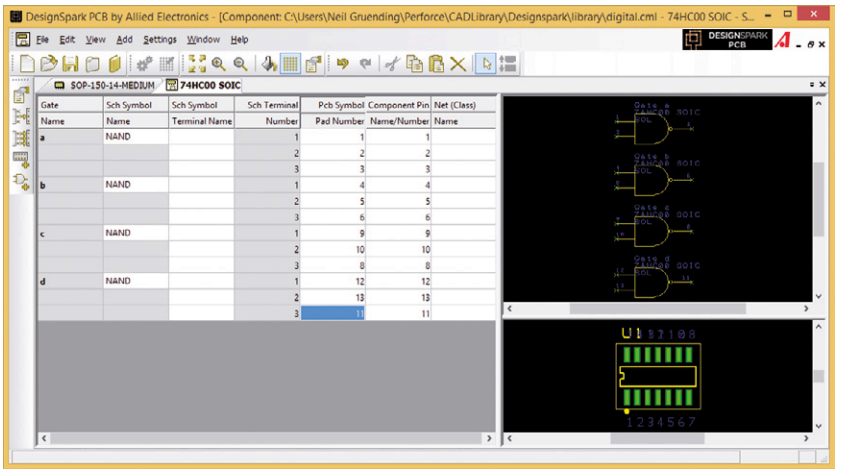


Bild 2.
Component Editor.



Das gilt auch für das Beispiel 74HC00: Dieses Vierfach-NAND-IC nutzt das Symbol für die Gates eben vierfach. Mir hat das NAND-Symbol der DesignSpark-Libraries nicht gefallen, weswegen ich ein eigenes erstellt habe. Als nächster Schritt war das Layout für das 14-polige SO-Gehäuse an der Reihe, und schon war alles für den *Component Wizard* bereit. Nach dessen Start gibt man im Dialog *Component Details* eine „4“ bei der Anzahl an Gates ein (siehe **Bild 1**). Der Wizard weiß nun, dass er es mit vier NAND-Gates zu tun hat. Die Anschlüsse für die Stromversorgung kommen später an die Reihe. Jetzt muss man zuerst einmal das gewünschte Schaltplan-Symbol und das zugehörige Gehäuse auswählen. Dann muss man die Pins des Symbols mit denen des Layouts verknüpfen. Für mich ist es einfacher, zunächst den Knopf *Assign 1-zu-1* zu betätigen und anschließend die Pin-Belegung mit dem *Component Editor* zu ändern.

Nach dem Schließen des Wizards kann man das Bauteil editieren (**Bild 2**). Die linke Bildschirmseite enthält die Gates in einer Tabelle, in der man die Pin-Nummern ändern kann. Jedes Gate erhält einen Kleinbuchstaben, beginnend mit „a“. Jeder Pin hat seine eigene Reihe in der Tabelle. Die Pin-Nummern des Layouts werden in eigenen Spalten aufgelistet. Mit liegt der Editor-Dialog mehr als der *Component Wizard*, da hier alle Gates plus Layout bzw. Gehäuse angezeigt werden und man die Änderungen sofort überprüfen kann.

Wenn das erledigt ist, kann man das 74HC00 wie andere Bauteile auch sofort im Schaltplan verwenden. **Bild 3** zeigt, wie dies aussieht. Die schwarzen NAND-Gates wurden bereits in der Schaltung platziert, aber die roten noch nicht. Letztere sind neben dem Mauszeiger dargestellt und zeigen so an, wie viele Gates dieses Bauteils noch platziert werden können. DesignSpark führt Buch über die bereits verwendeten Gates, sodass man sich darum nicht zu kümmern braucht. Wenn man sich also nach dem Platzieren eines Gates um Widerstände kümmert, erinnert sich DesignSpark

immer noch daran, dass bisher lediglich Gate „a“ verwendet wurde. Bei der nächsten Verwendung des 74HC00 möchte DesignSpark automatisch das Gate „b“ platzieren und es wird angezeigt, dass noch drei Gates zur Verfügung stehen.
DesignSpark sorgt dafür, dass man nicht aus Versehen das gleiche Gate mehrfach verwendet. Das Beispiel in Bild 3 zeigt, dass schon alle vier Gates von U1 (U1a...U1d) platziert wurden. Wenn man jetzt einzelne Gates für ein einfacheres Routing der Platine austauschen möchte, nachdem sie platziert und schon „verdrahtet“ wurden, dann reicht ein Doppelklick auf ein Gate und man kann einfach den Buchstaben dieses Gates in den gewünschten ändern. Wurde der gewünschte Buchstabe schon verwendet, dann wird man von DesignSpark gewarnt und es wird angeboten, diese Gates auszutauschen. Möchte man beispielsweise das Gate „a“ mit „c“ austauschen, kann man einfach im Dialog *Gate Properties* manuell das „a“ durch ein „c“ ersetzen. DesignSpark wird dann anbieten, Gate „c“ in „a“ umzubenennen.

Spannungsversorgung

Das erstellte IC 74HC00 hat zwar jetzt die richtige Anzahl an Gates, aber es funktioniert noch nicht, da die Anschlüsse für die Spannungsversorgung noch nicht verknüpft sind. Hierfür gibt es nur drei prinzipielle Möglichkeiten: versteckte Power-Pins, jedes Gate mit eigenen Power-Pins oder ein separates Symbol für die Power-Pins. DesignSpark bietet die letzten beiden Optionen. Eigene Power-Pins pro Gate können sinnvoll sein, wenn diese auch tatsächlich nur für dieses Gate genutzt werden. Gleichwohl kann DesignSpark einem Gehäuse-Pin auch mehrere Symbol-Pins zuordnen. Beim 74HC00 kann z.B. jedes Gate Power-Pins haben, die dann alle mit den Power-Pins des Layouts verknüpft werden. Ästhetischer sind allerdings von den Gates unabhängige Power-Pins. Daher verwende ich ein separates Schaltplan-Symbol für die Power-Pins.
Der *Component Wizard* verweigert bei der Erstellung des Gesamtbauteils die Verwendung unterschiedlicher Symbole. Man muss sie also nachträglich mit dem *Component Editor* via Menü *Add → Gate* hinzufügen. Es öffnet sich dann der Dialog *Add Gate* und man kann das gewünschte Symbol auswählen. **Bild 4** zeigt, wie das Power-Symbol nach Anpassung der Pin-Belegung aussieht.

Bauteile mit multiplen Gehäusen

Dass Bauteile mehr als nur ein Gehäuse haben können, ist ein nützliches Feature. DesignSpark verlangt nicht einmal, dass alle Gehäuse eines Bauteils die gleiche Anzahl an Pins haben. Bei gleicher Pin-Anzahl ist es aber einfacher, da so die gleiche Pin-Belegung geteilt werden kann. Das Menü *Add → Package* öffnet den Dialog *Add Package*, mit dem man weitere Gehäuse hinzufügen kann. Man wählt hier das gewünschte zusätzliche Gehäuse aus und vergibt einen Gehäusenamen. Es existieren schon etliche vordefinierte Bezeichnungen, aber man kann auch einen eigenen Namen vergeben. Die Gehäusenamen werden dann beim Platzieren des Bauteils zur Auswahl angezeigt.

Bild 3.
Platzieren eines 74HC00.

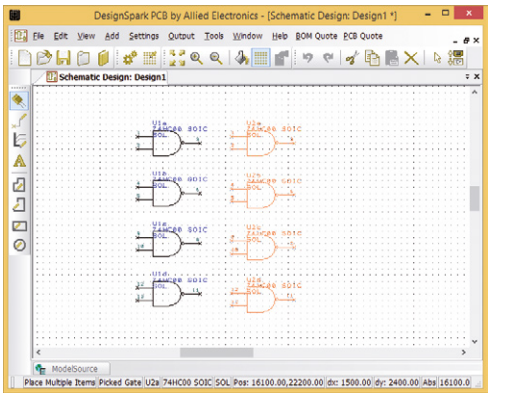


Bild 4.
Alle 74HC00-Symbole.

Bild 5 zeigt den Dialog *Add Component* beim Platzieren des 74HC00 mit dem ausgewählten Gehäuse. Hier wurde *SOL* als „narrow SOIC“ ausgewählt. Die Option *SOIC WIDE* erklärt sich selbst. Das ausgewählte Gehäuse wird dann wie gewohnt platziert. Wenn man das Gehäuse eines schon im Schaltplan eingefügten Bauteils nachträglich ändern möchte, öffnet man den Properties-Dialog des Bauteils und klickt auf *Change*. Nun zeigt sich der Dialog *Change Component*, in dem man das Gehäuse ändern kann.

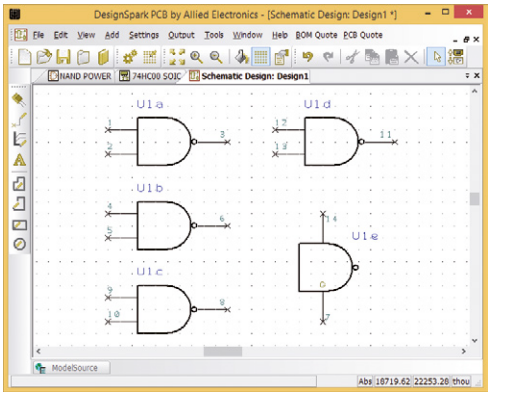
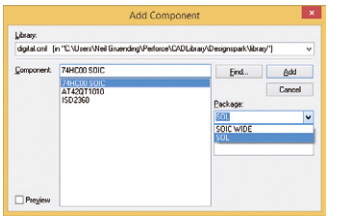


Bild 5.
Platzierung eines Bauteils mit mehreren Gehäuseformen.



Ausblick

In diesem Beitrag ging es um Bauteile mit mehreren Symbolen und mehreren Gehäuseformen. In der nächsten Folge werden die Layout-Tools von DesignSpark näher beleuchtet.

(140419)

Tetroden-Transistor

Seltsame Bauteile (12)

Von **Neil Gruending**
(Kanada)

Wir alle verwenden npn- und pnp-Transistoren mit drei Anschlüssen wie selbstverständlich, aber ich habe noch nie zuvor von einem Tetrodentransistor gehört. Solche Tetrodentransistoren besitzen vier Anschlüsse und tauchen in verschiedenen Variationen auf. Bemerkenswerterweise weisen sie zwei Steuerelemente anstelle von einem auf: zwei Basis-Anschlüsse (b) bei Bipolar-Transistoren und zwei Gate-Anschlüsse (G) bei MOSFETs, wie die Schaltsymbole in **Bild 1** zeigen. Werfen wir einen genaueren Blick darauf und erforschen, was die Transistoren so besonders macht!

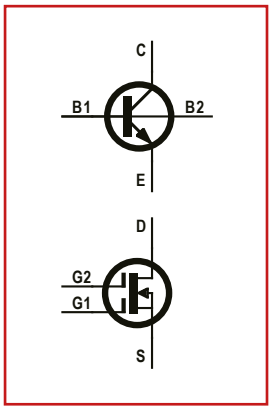


Bild 1. Schaltsymbol des Tetrodentransistors.

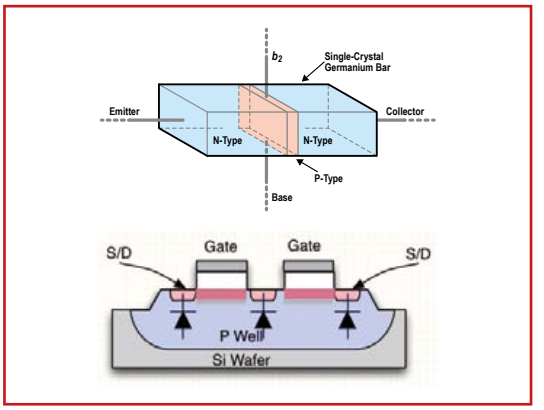


Bild 2.
Aufbau des Tetrodentransistors.

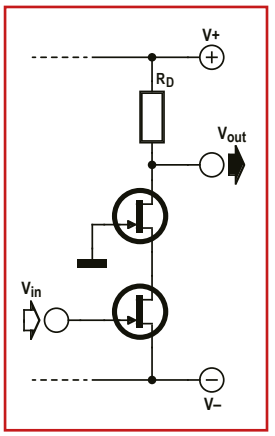


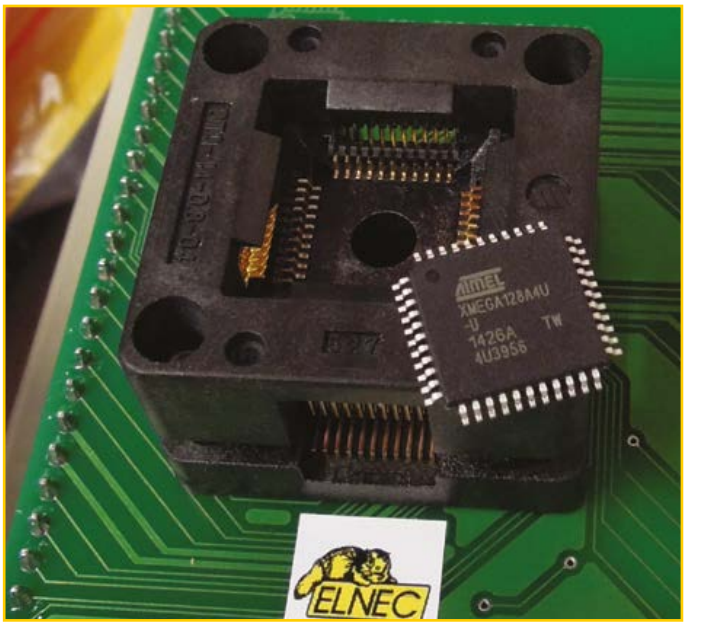
Bild 3.
Kaskodenverstärker.

Eine Tetrode bezeichnet in der Elektronik prinzipiell ein Bauteil mit vier aktiven Elektroden. In der Regel bezieht sich der Begriff auf eine Vakuumröhre, die zwei Gitter anstelle eines hat. Das zusätzliche Gitter, Schirmgitter genannt, senkt die Kapazität zwischen Gitter und Anode im Vergleich zu einer herkömmlichen Triodenröhre, was den Frequenzbereich erweitert. Tetrodentransistoren sind zwar kein 1:1-Ersatz für Vakuumröhren-Tetroden, aber auch gedacht, um die parasitäre Kapazität zu reduzieren und den Arbeitsfrequenzbereich zu erweitern. Ein bipolarer Tetrodentransistor besitzt eine zusätzliche Basis-Verbindung auf der entgegengesetzten Seite des Siliziums (obere Seite von **Bild 2**). Die untere Abbildung zeigt, wie ein Dual-Gate-MOSFET aufgebaut ist.

Ich habe ziemlich viel von Kapazität gesprochen. Warum ist das so wichtig? Nun, die Konstruktion von Verstärkern mit diskreten Transistoren ist immer ein ständiger Kampf um genügend Bandbreite, Verstärkung und Trennung von Ein- und Ausgang. Einer der Gründe ist der Miller-Effekt und sein Einfluss auf den Verstärker. Der Miller-Effekt bezeichnet die effektive Vergrößerung der Eingangskapazität eines Verstärkers aufgrund der Verstärkung der Kapazität zwischen Ein- und Ausgang. Bei einem Transistorverstärker ist es normalerweise die parasitäre Kapazität des Transistors, die stört, wenn Sie die Verstärkerbandbreite maximieren wollen.
Eine Möglichkeit, den Miller-Effekt zu minimieren, ist es, zwei Transistoren zu verwenden, um einen Kaskodenverstärker wie in **Bild 3** zu schaffen. Ein Transistor wird dabei als Transkonduktanzverstärker verwendet, der zweite stellt einen Strompuffer dar. Die beiden Verstärkerstufen helfen, den Miller-Effekt durch Isolation der Eingangskapazität des unteren Transistors von der Pufferstufe oben zu minimieren. Es stellt sich heraus, dass Tetrodentransistoren ausgezeichnet für Kleinsignal-Kaskodenverstärker geeignet sind, wenn sie eine hohe Bandbreite aufweisen müssen.
Der häufigste Typ des Tetrodentransistors in vielen interessanten HF-Schaltungen ist der Dual-Gate-MOSFET. Berühmte HF-Bauteile aus den 1980er Jahren sind die europäischen **BF96x/BF98x**-Typen und in den USA die **40.673**- und **3N211**-Transistoren. Bei einem einfachen FET-Mischer sind der Ausgang eines lokalen Mischers und der HF-Ausgang mit dem Gate eines FETs verbunden; dabei sind einige Bauteile nötig, um den Mischer von der HF zu trennen. Ein Dual-Gate-MOSFET mit seinen schon voneinander isolierten Gates kann diese Bauteile überflüssig machen. Dual-Gate-MOSFETs sind auch gut für eine automatische Verstärkungsregelung (AGC) geeignet, da man ein Gate vorspannen und das Signal zum anderen Gate führen kann. Dann kann die Vorspannung zur Steuerung der gesamten Transistorverstärkung verwendet werden.

(140418)

Der Programmer im Elektor-Labor



Wir bei Elektor sind in vielen (und verschiedenen) Fällen mit Problemen konfrontiert, die bei der Realisierung von in der Zeitschrift beschriebenen Projekten entstehen können. Selbstverständlich sind wir immer bemüht, Ihnen zu helfen, Ihr Projekt erfolgreich abzuschließen. Zum Beispiel mit unserem Chip-Programmierungs-Service. Die berüchtigten Fuse-Bits und Boot-Reset-Vektoren müssen für jedes Projekt anders und dennoch absolut richtig gesetzt werden. Um Ihnen diese (fehlerträchtige) Arbeit zu ersparen, können Sie die meisten Mikrocontroller unserer DIY-Projekte vorprogrammiert in unserem Online-Shop erwerben. Für diese Hilfestellung ist das Personal im Elektor-Labor zuständig.
Um Firmware richtig in einen Mikrocontroller zu „brennen“, setzen wir im Elektor-Labor seit einiger Zeit unseren treuen BeeProg+-Programmer ein, mit freundlicher Unterstützung des Herstellers Elnec. Dieser universelle Programmer nimmt Mikrocontroller mit bis zu 48 Pins in einer komfortablen Zero-insertion-force-Fassung (ZIF) auf. Das passt für viele Mikrocontroller, aber nicht für alle. Wenn die ZIF-Fassung nicht passt, wird ein Adapter benötigt, wie beim Mikrocontroller ATXmega128A4U-AU aus dem VariLab 402-Projekt (11/12-2014), der in einem TQFP44A-Gehäuse steckt. Im Entwicklungsstadium wurde der Mikrocontroller ICP-programmiert. Wenn wir aber für den Einzelhandel und den Versand eine größere Stückzahl benötigen, scheint es sehr umständlich, die Controller erst auf eine Platine zu löten, ihn zu ICP-programmieren, wieder auszulöten und schließlich zu verschicken. Also brauchen wir einen Adapter für unseren BeeProg+-Programmer!

Glücklicherweise konnte unser Freund Vladimír von Elnec helfen. Als wir ihn im November letzten Jahres auf der electronica 2014 in München trafen, überraschte er uns mit einem Adapter für TQFP44A-Bauteile. Wir verdanken solchen Jungs wie Vladimír (und der völligen Flexibilität der BeeProg+-Programmierer), dass wir die Preise von programmierten Mikrocontrollern im Zaum halten können, obwohl wir ja nicht in Massenstückzahlen produzieren. Sicher, ich sollte ein kurzes Stückchen darüber schreiben, aber ich nutze diese Gelegenheit auch, um Ihnen einen Blick hinter die Kulissen zu gewähren. Dies sind die Deals, die ich liebe!

(140417)

Von **Thijs Beckers**
(Elektor-Labor)

Abschaltverzögerung für Mikrocontroller

Geordnetes Herunterfahren

Von **Jack Jouas** (F)

Wenn man ein Mikrocontroller-System während des Programmablaufs ausschaltet, können unvorhersehbare Komplikationen auftreten. Mit der Abschaltverzögerung ist das System hiergegen gewappnet.

Die Abschaltverzögerung ist eine analoge Ergänzung für Mikrocontroller-Systeme, sie dient vorrangig der Betriebssicherheit. Wenn der Anwender die Schaltung manuell ausschaltet, durchläuft der Mikrocontroller zuerst eine im Programm implementierte Notroutine, bevor die Schaltung stromlos gemacht wird.
Wenn ein Mikrocontroller-System übergangslos abgeschaltet wird, sei es absichtlich oder versehentlich, sind die Auswirkungen oft unberechenbar.

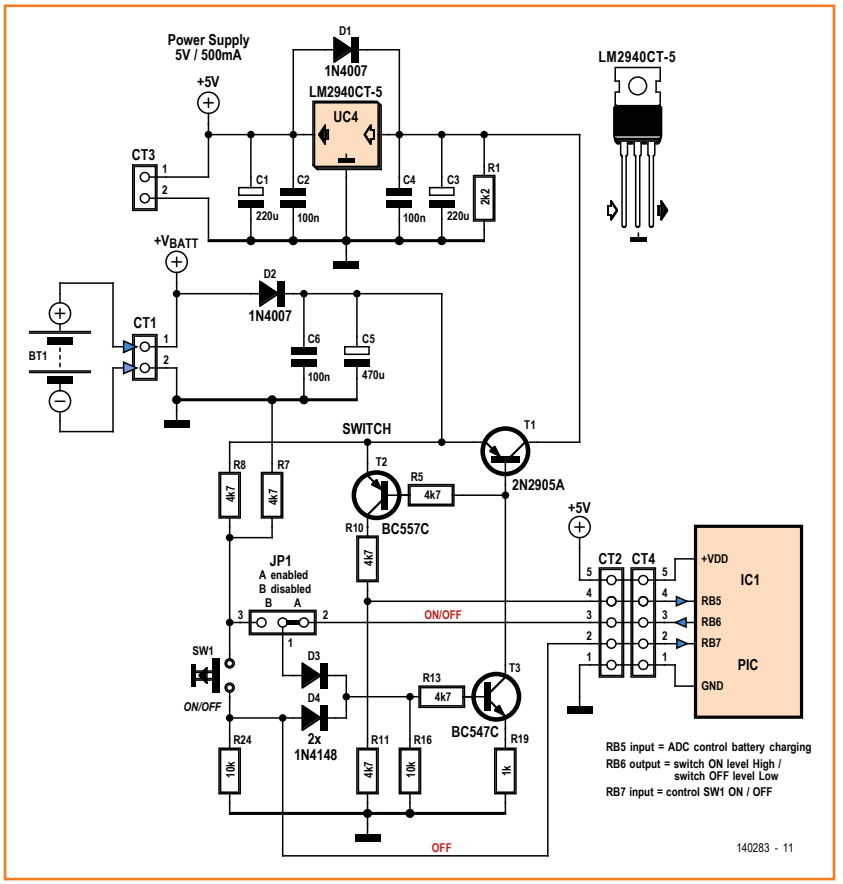


Bild 1.
Die wichtigen Bauteile sind drei Transistoren und ein Spannungsregler.

Die System-Parameter zum Zeitpunkt des Ausfalls sind unbekannt. Das gilt auch für periphere Subsysteme, die an das Mikrocontroller-System angebunden sind. Die Betriebssicherheit erfordert, dass der Mikrocontroller das Abarbeiten des Programms nicht willkürlich abbricht, sondern zuvor essentielle Daten nichtflüchtig speichert, beispielsweise in einem EEPROM. Die Abschaltverzögerung hält die Betriebsspannung noch so lange aufrecht, bis der Mikrocontroller eine Notroutine durchlaufen hat, die diese Aufgaben erledigt.
Das gleiche Ziel lässt sich zwar auch mit einem kapazitätsstarken Kondensator erreichen, der die Betriebsspannung stützt. Die im Kondensator gespeicherte Energie muss so lange ausreichen, bis die Abschaltprozedur vollständig durchlaufen ist. Leider ist diese Lösung nicht unbedingt zuverlässig. Welches Kriterium garantiert, dass die Abschaltprozedur tatsächlich vollständig durchlaufen wurde? Falls der Mikrocontroller seinen Betrieb erst beenden darf, nachdem ein peripheres Subsystem in den Ruhezustand versetzt wurde, steigt die Komplexität der Abschaltprozedur unverhältnismäßig an. Es bleibt das Risiko, dass die im Kondensator gespeicherte Energie erschöpft ist, bevor das System vollständig in einen geordneten Ruhezustand überführt werden konnte.

Selbstbestimmt

Mit der hier beschriebenen Abschaltverzögerung kann im Mikrocontroller-Programm entschieden werden, wann das System abschaltet, nachdem die programmierten Abschlussprozeduren ausgeführt sind. Solche Prozeduren können beispielsweise das Anfahren neutraler Stellungen bei Servo-Steuerungen sein. Zuvor können die aktuellen Servo-Stellungen wie Momentaufnahmen gespeichert werden, damit der Betrieb nach Wiederkehr der Betriebsspannung nahtlos fort-

gesetzt werden kann.
Die Abschaltverzögerung ist ausschließlich mit diskreten Bauelementen aufgebaut, so dass sie sich bei Bedarf unkompliziert anpassen lässt. Zur Schaltung gehören keine Relais, da der Strombedarf eines Relais wahrscheinlich den Strombedarf des Mikrocontrollers übersteigen würde, was natürlich nicht sinnvoll wäre.
In der Schaltung (**Bild 1**) liegt an der Basis von T3 ein Signal, das diesen Transistor in den Leitzustand (High) oder Sperrzustand (Low) steuert. Wird der Start-Stopp-Taster SW1 kurzzeitig oder auch lange gedrückt, gelangt über den Spannungsteiler R7/R8 und die Diode D4 ungefähr die Hälfte der an der Spannungsquelle BT1 liegenden Spannung zur Basis von T3. Widerstand R13 begrenzt den Basisstrom. Der Kollektor von T3 steuert die Basis von T1, ein PNP-Transistor, der die Spannung der Spannungsquelle BT1 schaltet. Symbolisch für andere Typen ist in Bild 1 ein PIC-Mikrocontroller gezeichnet. T1 ist ein 2N2905A, über diesen Transistor können Ströme bis 500 mA fließen, ohne dass er gekühlt werden muss. Wenn das zu versorgende Mikrocontroller-System einen höheren Strombedarf hat, kann der 2N2905A durch einen leistungsstärkeren Typ ersetzt werden, beispielsweise durch einen p-Kanal-Power-MOSFET.

Interaktion

Mit SW1 wird die Schaltung an- und ausgeschaltet. Während des Betriebs muss der Mikrocontroller (im Beispiel mit dem PIC) den Ausgang RB6 auf High halten, so dass die Basis von T3 über die Diode D3 Strom erhält (auch wenn der Drucktaster SW1 offen ist). Die Schaltung wird jetzt über den Spannungsregler UC4 mit Strom versorgt.
Wenn Taster SW1 gedrückt wird, um die Schaltung wie beschrieben zu starten, reagiert der Mikrocontroller noch nicht auf das Signal High am Eingang RB7. Der Mikrocontroller ist dann mit dem Initialisieren und Steuern seiner Subsysteme beschäftigt. Nach Loslassen von SW1 wird Eingang RB7 von Widerstand R24 nach Low gezogen. Eine Aktion findet nicht statt.
Jetzt soll der Fall betrachtet werden, dass die Betriebsspannung verzögert abgeschaltet wird. Angenommen das System arbeitet bereits im regulären Betrieb. Taster SW1 wird gedrückt, so dass Eingang RB7 des Mikrocontrollers auf High geht. Für den Mikrocontroller muss dies das Signal sein, dass unverzüglich die Notroutine abzuarbeiten ist: Speichern systemrelevanter Daten sowie weitere Prozeduren, die vor dem Ausstieg erledigt sein müssen. Beim Autor steuert das Mikrocontroller-System diverse Servos, die vor dem Abschalten unbedingt in Neutralstellung gefahren werden müssen. Ist die Prozedur abgeschlossen, setzt der Mikrocontroller Ausgang RB6 auf Low, so dass T3 und T2 sperren. Die Betriebsspannung über Spannungsregler UC4 ist unterbrochen, das System ist spannungslos.

Spannung messen und anzeigen

Die Teilungsfaktoren für die Anzeige der Batterie- oder Akkuspannung lassen sich wie folgt berechnen: Angenommen die Spannungsquelle ist ein Li-Ion-Akku mit der nominalen Spannung 7,4 V. Geladen (100 %) beträgt die Spannung 8,3 V beim Strom 50 mA. Wenn die Spannung auf 6,5 V gesunken ist, kann der Akku als entladen (0 %) betrachtet werden. Bei einem 8-bit-A/D-Wandler ist dem Spannungsbereich 0...5 V der Wertebereich 0...255 zugeordnet, so dass die Auflösung 5 V / 256 = 0,01953 V beträgt. Mikrocontroller verarbeiten normalerweise keine Fließkommazahlen, sondern nur ganze Zahlen vom Typ Byte oder Word. Um den Dezimalpunkt zu verschieben, werden die ADC-Werte vor der Rechenoperation um einen Faktor 10 oder 100 oder höher multipliziert. Das Ergebnis muss bei der Länge 16 bit unter 65535 (und 256 bei 8 bit) bleiben. Feste Faktoren (Verhältniswerte) können auf anderem Weg berechnet werden und als Konstanten in das Programm eingehen.

[1] Mit $V_{Bat}/2$: 830 / 2 = 415 (100 %) und 650 / 2 = 325 (0 %) [$8.3 * 10^3$ und $6.5 * 10^3$]
Auflösung = 1953 [$0,01953 * 10^5$]
[2] Verhältnis Min/Max 212 – 166 = 46 100 / 46 = 2,174 2174 [$2,174 * 10^3$]
(415 * 1000 / 1953) = 212
(325 * 1000 / 1953) = 166
Der vom A/D-Wandler ausgegebene Wert liegt zwischen 212 für 100 % und 166 für 0 %.
[3] Spannung des Akkus in %:
100 – [((212 – ADC) 2174) / 1000]
In der Anzeige:
(ADC) = 180 >>> 100 – [((212–180) 2174) / 1000] = 30 % >>> 7,04 V

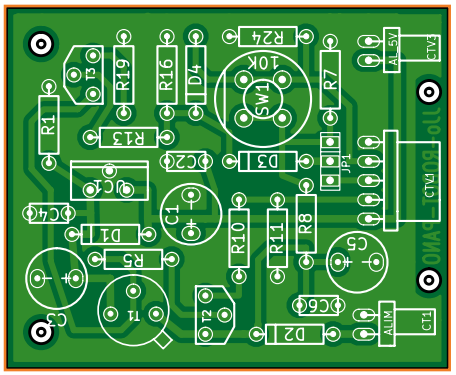


Bild 2.
Das vom Autor für die Abschaltverzögerung entworfene Platinenlayout.

Noch nicht alles

Jumper JP1 leistet Hilfe beim Entwickeln der Notroutine für den Mikrocontroller. Wenn Ausgang RB6 zu einem nicht definierten Zeitpunkt auf Low geht, würde sich die Schaltung selbst außer Betrieb setzen. Diese Situation kann beispielsweise bei einem fabrikneuen Mikrocontroller auftreten, dessen Portleitungen vom Hersteller auf Low gesetzt sind. Mit Jumper JP1 in Stellung B bleibt dies ohne Auswirkungen. Sobald das Programm in den Mikrocontroller geladen ist, kann Jumper JP1 in Stellung A zurückgesetzt werden.

Eine Nebenfunktion der Abschaltverzögerung ist das Überwachen der Batterie- oder Akku-Spannung, sie kann vom Mikrocontroller auf einem Display angezeigt werden. Transistor T2 ist ein PNP-Transistor BC557, sein Emitter liegt hinter Diode D2 am Pluspol der Spannungsquelle BT1. Sowohl T2 als auch T1 leiten, wenn die Betriebsspannung anliegt. Die Spannung von BT1 gelangt dann über T2 und Spannungsteiler R10/R11 zum Eingang RB5 des Mikrocontrollers, der als Eingang eines internen AD-Wandlers konfiguriert ist. An den Eingängen des Mikrocontrollers dürfen keine Spannungen liegen, solange der Mikrocontroller ohne Betriebsspannung ist. Der Spannungsteiler vor Eingang RB5 ist notwendig, weil die anliegende Spannung die nominale Betriebsspannung des Mikrocontrollers nicht übersteigen darf. Nach Teilung durch den Faktor 2 liegt die Spannung immer unter +5 V, auch wenn BT1 neu oder voll geladen ist. Diode D2 schützt die Abschaltverzögerung gegen die Verpolung von BT1. D1 und R1 sorgen dafür, dass C1 und C3 entladen werden, wenn die Schaltung außer Betrieb ist.

Mikrocontroller-Programm

Der Autor hat die Abschaltverzögerung zusammen mit einer komplexen Steuerung eingesetzt, in der Servos die Aktionen eines Roboters steuern. Das Roboter-Programm des verwendeten PIC-Mikrocontrollers ist umfangreich, an dieser Stelle ist es sicher nicht relevant. Hier kann nur in Kurzform gezeigt werden, wie die Ausstiegsprozedur gestaltet werden kann.

Als Vorbild soll das Beispiel in **Listing 1** dienen, es kann lediglich Richtschnur für den Einsatz in anderen Mikrocontroller-Systemen sein. Das Prinzip bleibt gleich, unabhängig von der Mikrocontroller-Familie und der Programmiersprache. Wenn der Mikrocontroller vorwiegend mit dem Abarbeiten eventgesteuerter Routinen beschäftigt ist, kann es sinnvoll sein, für das Signal an Eingang RB7 eine Interrupt-Routine zu implementieren.

(140283)gd

Listing 1

```
Initialization:
Port B5 and B7 as inputs (IN) and B6 as output (OUT)
Initialize port B5 as ADC 8 bits
Set Port B6 to high.
Rest of initialization . . .
. . . .
//-------------------------------------------------
main :  (microcontroller main program loop)
do
. . . . . (various actions)
. . . . .
    //-------------------------------------
    read  port B7
          If high
  execute routine for shutdown (backup data, servos to neutral . . . .)
  set port B6 to low (power shutdown)
    //-------------------------------------
    read port B5 (ADC)
       execute routine to update battery charge level
       display value
loop
//-------------------------------------------------
```

Entwickeln mit dem IoT-Kit WiSmart

Von **Jan Buiting**

1

Viele IoT-Produkte, die gegenwärtig auf den Markt kommen, sind BlackBoxes und so für eigene Experimente und Entwicklungen völlig ungeeignet. Sie können sich aber zuverlässigen, kostenlosen Software-Tools bedienen, um auf Chip-Ebene IoT-Applikationen zu entwickeln. Die Firma *Econais*, die in Griechenland und dem Silicon Valley beheimatet ist, bietet mit dem EC19D-WiSmart ein winziges Modul mit WLAN-Fähigkeiten an. Mit diesem Modul können Ingenieure wie Enthusiasten raffinierte IoT-Produkte in Kleinserien wie für die Serienproduktion entwickeln, die clever, umweltfreundlich und „WiFi" sind. Um die Entwicklungsarbeit zu unterstützen, bietet Econais ein Entwicklungskit mit der Bezeichnung **EC19D01DK** an. Damit können Sie eine Menge lernen!

Mit Standard-Tools geht es

Econais unterscheidet sich von der Konkurrenz schon allein durch die Tatsache, dass Sie keine 0,5 GB große proprietäre Tools-Suite herunterladen müssen. Abgesehen von einem kleinen Python-Programm namens *SimpleCom.py* basiert der gesamte Prozess von Entwicklung, Debugging und Implementierung eines WLAN-IoT-Geräts auf Hard- und Software, die bekannt, vertraut und größtenteils kostenlos ist: Linux, Windows, SPI, UART, JTAG, AT-Befehlssatz, FTDI USB-TTL, SD-Karte, Open Hardware, Gerber, Python ...
Nach der einfachen Anmeldung auf der Econais Website [1] können Sie auf eine Unmenge von Dokumentation zum EC19D-Chip und zum damit verbundenen EC19D01DK-Entwicklungskit (seit August 2014 verfügbar) zugreifen. Das Elektor-Labor war einer der ersten Empfänger des neuen Kits. Das Kit ersetzt/verbessert eine frühere Version und enthält zum nahezu gleichen Preis nun auch ein Debugger-Board.

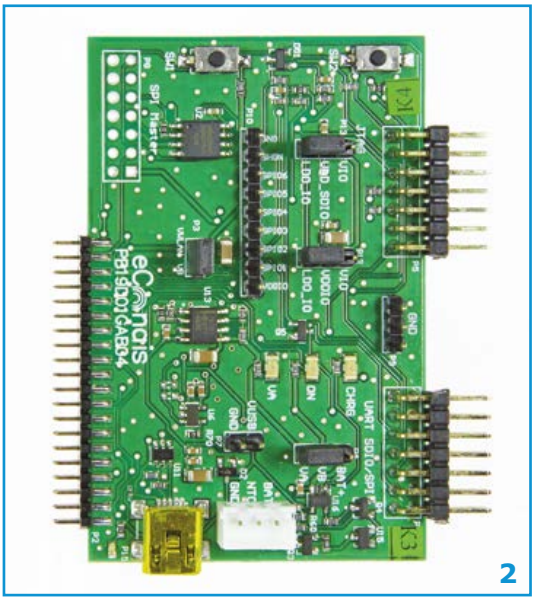
Von TTL-TxD/RxD zum WLAN-IoT

In der Box befinden sich erwartungsgemäß Kabel, eine Multi-Standard-Stromversorgung, das EC19D00SD-WLAN-Board **(1)**, die EC19D01EX-Erweiterungskarte **(2)** und der EC19D01DBG-JTAG/Debugger **(3)**. Die obligatorische „Schnelleinstieg"-Karte ist auch dabei,

mit so knappen Informationen, dass sie zunächst eher als eigene Herausforderung anzusehen ist. Doch obwohl die Dokumentation ein wenig ungehobelt daherkommt, ist das Kit gut zur Ausbildung geeignet. Wie ich erfahren musste, ist die erste Hürde aber recht hoch: Ich musste das EX-Board dazu bringen, mit mir zu kommunizieren. Je nachdem, was man (via EX-Board) vom EC19D-Chip hören möchte - Maschinen- oder menschliche Sprache - gibt es zwei Möglichkeiten: die erste verwendet ausschließlich Befehlskürzel und Zahlen, die zweite so etwas wie AT-Kommandos. Beide Methoden nutzen einen virtuellen FTDI USB-TTL-COM-Port (RxD-TXD-CTS-RTS) und ein Terminal-Programm wie RealTerm unter Windows (oder Cuteterm unter Linux) oder aber den Python-Weg mit SimpleCom. Für SimpleCom muss allerdings Python installiert sein, was aber im Hinblick auf das IoT-Potential eines RPi ohnehin angebracht erscheint.
Sobald die Kommunikation Ihres ARM, RPi, AVR oder Arduino über die SPI/UART-Schnittstelle des EX-Boards funktioniert, ist der Weg zum WLAN-IoT auch nicht mehr weit. Der gesamte Datenverkehr über den Port ist „lesbar“, er besteht aus Befehlen, Datenworten und Parametern. In einem wirklich anspruchsvollen Setup werden -SD, -EX und -DGB-Platine wie auf dem Foto miteinander verbunden und mit einem USB-Kabel an den PC angeschlossen. Dieses Setup wird derzeit aber nur unter Linux unterstützt. Das DBG-Board fungiert als fortschrittlicher Flash-Programmer für den EC19D-Chip. Da es sich um ein relativ neues Produkt handelt, wären allerdings erst einmal ein paar Anwendungsbeispiele erforderlich. Wenn Sie in Sachen IoT-Programmierung ambitioniert sind, dann ist dieses Kit etwas für Sie. Zu den aufregendsten Dingen gehören OTA (Over the air-Upgrading), „smarte“ Messtechnik, Kanaldatenerfassung und Protokollierung mit *Wireshark* und Aufwecken einer beliebigen Anzahl von versteckten (!), schlafenden (!) WLAN-Geräten mit der *ProbMe*-App im WLAN-Direct-Modus Ihres Android-Smartphones.

Und jetzt?

Das EC19D01-Development-Kit erlaubt die Entwicklung einer vollständigen DIY-IoT-Anwendung: Programmierung und Löten, Linux und Windows, USB und Mikrocontroller-I/O sind dabei. Alles im Kit vermittelt das Gefühl, Pionierarbeit zu leisten, nicht zuletzt wegen der tonnenweise zur Verfügung gestellten Linux-Programmierbeispiele.



2

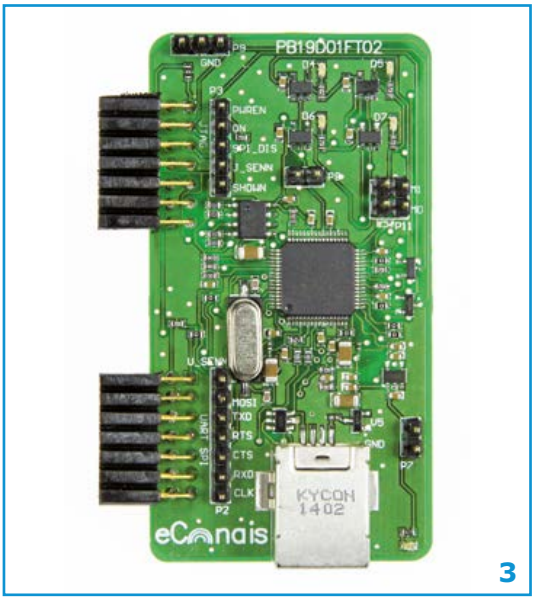
Die Leute bei Econais haben mit Elektor vereinbart, ihr neuestes Produkt, den EC19D01DK zu einem günstigen Sonderpreis exklusiv im Elektor-Shop [2] anzubieten. Ich brenne darauf, die erste von Ihnen entwickelte tolle IoT-Anwendung mit dem EC19D in Elektor zu veröffentlichen!

(140338)

Weblinks

[1] www.econais.com (Support -> Dev Kit Resources folgen)

[2] www.elektor.de/search?cat=0&q=wismart



3

VariLab 402 (3)
Software und Aufbau

Von **Clemens Valens** (Elektor-Labor)

Im dritten und letzten Teil geht es zuerst um die Software. Wie wurde sie geschrieben und wie ist sie strukturiert? Wichtige Themen sind die Benutzeroberfläche und die Kalibrierung. Am Schluss stehen der elektrische und mechanische Aufbau.

Mikrocontroller

Die Wahl des Mikrocontrollers für ein projektiertes Gerät oder System ist fast immer eine schwierige Entscheidung. Für das VariLab 402 haben wir nach einem Typ mit integrierten A/D- und D/A-Wandlern Ausschau gehalten, damit die Regelungsschleife möglichst verzögerungsarm arbeitet. Ein separater D/A-Wandler neben dem Mikrocontroller würde eine Alternative darstellen, zumal die meisten Mikrocontroller bereits einen A/D-Wandler an Bord haben. Allerdings würde dieses Konzept das Generieren der Regelungssignale verzögern. Dem steht die Forderung entgegen, dass insbesondere die Strombegrenzung schnell und effizient reagieren muss, möglichst lange bevor ein Bauteil defekt wird.

Da das VariLab 402 von außen über USB steuerbar sein soll, ist ein Mikrocontroller mit USB-Schnittstelle auf dem Chip eine praktische Lösung. Auch hier gibt es zwar die Alternative, einen separaten USB-Baustein in das Konzept aufzunehmen. Doch diese Variante würde den Platzbedarf auf der Platine steigern und die Gesamtkosten erhöhen. Alles sprach dafür, eine All-in-One-Lösung zu entwickeln und zu realisieren.

Die Anzahl der verfügbaren I/O-Leitungen erschien nicht kritisch, solange ein LC-Display, zwei Dreh-Encoder, eine LED, ein Buzzer (Tongeber) und ein Drucktaster anschließbar sind. Nach kurzen Recherchen war entschieden: Ein Mikrocontroller, der die Vorgaben erfüllt, ist der ATxmega128A4-AU von Atmel. Dieser Typ hat zudem so viel Speicher an Bord, dass keine Engpässe zu befürchten sind.

Atmel Studio

Die Software für den ATxmega128A4-AU des VariLab 402 wurde unter der Entwicklungsumgebung Atmel Studio 6 (AS) erstellt, die Atmel kostenlos bereitstellt. Eingebunden sind Bibliotheken des Atmel Software Frameworks (ASF), was nicht zwingend notwendig ist, jedoch viel Programmierarbeit erspart. In den ASF-Bibliotheken sind Treiber enthalten, unter anderem für den USB-Port (Communication Device Class, CDC), die Dreh-Encoder (QDEC), die A/D- und D/A-Wandler (ADC und DAC), das interne EEPROM und die Timer. Für die meisten peripheren Komponenten des Mikrocontrollers bieten die Bibliotheken ebenfalls Treiber an.

Die Bibliotheken der ASF gleichen einem nicht ganz leicht zu durchdringenden Dschungel, denn sie sollen die zahlreichen Entwickel-, Evaluier- und Demo-Boards einschließlich der darauf befindlichen Mikrocontroller von Atmel unterstützen. Etwas Licht in den Dschungel bringt der zur Atmel Studio IDE gehörende Assistent (Wizard), der

dem Programmierer einige Denkarbeit abnimmt. Mit dem Assistenten können Treiber vergleichsweise unkompliziert in Projekte eingebunden werden. Allerdings werden dabei zahlreiche Ordner und Unterordner mit zahllosen Dateien generiert, wobei oft nicht klar ist, welche Funktionen sie haben und weshalb sie gerade in diesem Ordner stehen. Übrigens werden in den ASF-Bibliotheken Treiber, Komponenten und Services unterschieden. Wir haben uns nicht der Mühe unterzogen, die Unterschiede zu ergründen. Beim Durchsuchen des gesamten Bestands wird das Gesuchte stets gefunden, egal ob Treiber, Komponente oder Service. Falls die Suche nicht zum Ziel führt, muss der Programmierer das Fehlende in eigener Regie schreiben.

Wie schon erwähnt, sind die ASF-Bibliotheken auf Atmels zahlreiche Boards und Kits zugeschnitten. Ein Projekt startet daher mit der Wahl des Board und des darauf befindlichen Mikrocontrollers aus einer Liste. Es verwundert nicht, dass unser vom Elektor-Labor entwickeltes Board nicht in der Liste vorkommt. Vermutlich haben die Programmierer bei Atmel diese Situation vorhergesehen, als sie die Option „User Board“ implementiert haben. Wir haben diese Option gewählt, wir hätten uns auch für ein Atmel-Board mit identischem Mikrocontroller entscheiden können. Dann hätten wir diverse Dateien anpassen müssen, was wahrscheinlich mehr Arbeit verursacht als eingespart hätte.

Fernbedienung über USB

Die USB-Schnittstelle des VariLab 402 ist für die Kommunikation mit einem fremden System bestimmt, zum Beispiel mit einem PC. Über den USB-Port können die eingestellten Werte von Ausgangsspannung und Strombegrenzung sowie die tatsächlichen Werte von Ausgangsspannung und Ausgangsstrom zum PC übertragen werden. Weitere abfragbare Informationen sind die Temperatur, der Status der LED, des Optokopplers und des Buzzers. In umgekehrter Signalrichtung kann der PC diese Parameter über den USB-Port einstellen (außer den tatsächlichen Werten von Ausgangsspannung und Strom sowie der Temperatur). Dazu muss auf dem PC lediglich ein Terminal-Programm wie Tera Term oder RealTerm laufen. Wenn der serielle Port auf der Setup-Seite aktiviert ist, sendet das VariLab 402 die Daten einmal in der Sekunde als lesbare, durch Kommas getrennte Zahlen in folgendem Format:

```
<Eingestellte Ausgangsspannung>,<Maximaler Ausgangsstrom>,<Tatsächliche Ausgangsspannung>,<Tatsächlicher Ausgangsstrom>,<Temperatur>,<LED>,<Optokoppler>,<Buzzer>,<CR><LF>
```

Die ersten fünf Werte haben drei Dezimalstellen.

Die Kommandos, mit denen das VariLab 402 gesteuert wird, haben folgendes Format:

```
$<Kommando>=<Wert><CR><LF>
```

Darin steht <CR><LF> (Carriage Return/Line Feed) für den Taster „Enter“. Es genügt, nur eins der beiden Zeichen zu senden. Um beispielsweise über das Terminal-Programm am PC die Ausgangsspannung des VariLab 402 auf 10,0 V einzustellen, lautet das Kommando:

```
$V=10.0<Enter>
```

Der Wert kann sowohl ein Integer ohne Dezimalpunkt als auch eine Fließkommazahl mit Dezimalpunkt sein. Folgende Kommandos sind erlaubt:

Kommando	Parameter	Bedeutung
V	0.00...40.0	Ausgangsspannung einstellen
I	0.00...2.00	Strombegrenzung einstellen
BUZ	1 oder 0	Akustische Signale Ein/Aus
BL	1 oder 0	LCD-Beleuchtung Ein/Aus
LED	1 oder 0	LED Ein/Aus
OPTO	1 oder 0	Optokoppler Ein/Aus
BEEP	0...65535	Tonlänge in Millisekunden
SAVE	1	Einstellungen im EEPROM speichern

Meistens ist für die Kommunikation des VariLab 402 mit dem PC ein Treiber erforderlich. Für Windows ist ein Treiber im Download zu diesem Projekt [1] enthalten, er hat den Namen atmel_devices_cdc.inf. Die Treiberinstallation verläuft wie gewohnt: Nachdem das VariLab 402 über USB mit dem PC verbunden ist, erkennt Windows eine neue Hardware-Komponente.

Wenn Windows dazu auffordert, muss der Ort der genannten Inf-Datei eingegeben werden.

Da die USB-Verbindung eine vollwertige USB CDC (Communication Device Class) ohne USB/Seriell-Umsetzer ist, entfällt das Einstellen einer Baudrate im Terminal-Programm.

Innerhalb eines Projekts werden sämtliche ASF-Dateien in einem Unterordner mit dem Namen „ASF“ abgelegt. Eine so genannte Header-Datei ASF.h steht im Ordner SRC des Projekts, dort hat die IDE auch die Datei main.c abgelegt. Diese Header-Datei integriert die Header-Dateien der diversen ASF-Treiber, so dass es keiner Überlegung bedarf, welche Datei an welcher Stelle eingebunden werden muss. In einer Datei, die ASF-Funktionen nutzen soll, genügt die simple Anweisung #include ASF.h.

Unsere eigenen C-Dateien stehen im Ordner SRC, unsere Header-Dateien im Ordner INC. Für jede periphere Komponente existiert ein solches Paar aus c- und h-Datei. Beispielsweise heißen die zum Buzzer (Tongeber) gehörenden Dateien buzzer.c und buzzer.h. Außerdem gibt es Dateien, die zur Mensch-Maschine-Schnittstelle gehören, sowie unterstützende Dateien. Allgemein beginnen die Namen von Funktionen und Variablen mit dem Namen der Datei, in der sie definiert sind. Beim Buzzer, um beim Beispiel zu bleiben, ist die Funktion buzzer_beep(uint16_t ms) in der Datei buzzer.c zu finden, während die Deklaration in buzzer.h steht.

Sicher ist schon deutlich, dass die Software in C geschrieben wurde. Es hätte auch C++ sein können, doch im Hinblick auf die Verfügbarkeit haben wir C gewählt. Bei den meisten Anwendungen ist nicht entscheidend, ob sie in C oder C++ programmiert werden, doch bei der Benutzerschnittstelle hätten wir C++ den Vorzug gegeben. Das Ergebnis in C nutzt eigentlich auch Techniken, die C++ eigen sind, nur es hätte sich in C++ etwas einfacher gestaltet.

Software-Überblick

Eine häufig zu beantwortende Frage lautet: Realtime Operating System (RTOS) oder nicht? Noch vor wenigen Jahren hatte RTOS den Ruf des Exotischen, heute ist die Vielzahl der Varianten kaum noch überschaubar. FreeRTOS ist Open-Source, es ist auch für den Atmel-Mikrocontroller ATXmega128A4-AU verfügbar. Ein RTOS hat jedoch den Nachteil, dass eine weitere Bibliothek in das Projekt einbezogen wird und die Taskverarbeitung oft undurchsichtig ist. Wir haben uns deshalb für einen Mix entschieden, bei dem eine einfache Schleife die weniger vordringlichen Tasks in vorgegebener Reihenfolge abarbeitet, während eine interruptgesteuerte Schleife die zeitkritischen Aktionen übernimmt. Gewissermaßen werden zwei Tasks ausgeführt, eine im Vordergrund, die andere im Hintergrund.

Die interruptgesteuerte Schleife wird vom System-Tick-Timer gesteuert (SysTick), sie wird im Abstand 100 µs (10 kHz) durchlaufen. Zuerst werden die Ausgangsspannung und der Ausgangsstrom gemessen, anschließend wird geprüft, ob die Strombegrenzung aktiviert werden muss. Wenn dieser Ausnahmezustand eintritt, wird die Funktion als Interrupt-Service-Routine deaktiviert. Die Routine fährt die Ausgangsspannung sofort auf einen Wert zurück, bei dem der Strom unterhalb des Grenzwerts liegt. Falls notwendig wird die Ausgangsspannung auf 0 V reduziert. Erst wenn alle Sicherheitsmaßnahmen getroffen sind, wird der reguläre Betrieb fortgesetzt.

Aus der 10-kHz-Schleife werden eine 1-kHz-Schleife und eine 100-Hz-Schleife abgeleitet. Die 1-kHz-Schleife steuert den Buzzer (Tongeber) und sorgt dafür, dass Änderungen der Einstellungen für Spannung und Strom an die D/A-Wandler weitergegeben werden. Die 100-Hz-Schleife fragt die Dreh-Encoder, den Taster und den Temperatursensor ab.

Die drei Schleifen können Flags setzen, sie werden von der Hintergrundschleife verarbeitet. Eine dieser Flags signalisiert, dass sich das System in der Strombegrenzung befindet. Die Hintergrundschleife prüft die Flags periodisch und setzt sie in visuelle oder akustische Signale um. Wenn das VariLab 402 Warntöne produziert (dreimal nacheinander) und die Hintergrundbeleuchtung des LC-Displays aufleuchtet, ist die Strombegrenzung aktiv geworden.

Eine andere Flag zeigt an, dass der Benutzer einen Bedienschritt vorgenommen hat. Die Flag ist notwendig, um der Hintergrundschleife Änderungen eines Einstellwerts oder der Anzeige auf dem Display zu signalisieren. Nachdem der Benutzer einen neuen Wert für Spannung oder Strom eingestellt hat, erwartet er, dass dieser Wert beim Ausschalten des Geräts erhalten bleibt. Aus diesem Grund werden die variablen Werte nichtflüchtig im EEPROM des Mikrocontrollers gespeichert. Da das Speichern im EEPROM relativ viel Zeit in Anspruch nimmt, geschieht dies nicht in der 10-kHz-Schleife, sondern in der Hintergrundschleife.

Das gleiche Verfahren wird auch für die Anzeige auf dem Display angewendet. LC-Displays sind ausgesprochen träge, die Aktualisierung ist innerhalb einer schnellen Interrupt-Service-Routine nicht ausführbar. Diese Aktion wird deshalb in die Hintergrundschleife verlagert, sie aktualisiert

die Anzeige im Abstand 100 ms. Das Intervall 100 ms stellt einen Kompromiss zwischen einer raschen Aktualisierung und dem Ablesekomfort dar. Wenn die Aktualisierungen zu schnell aufeinander folgen, erscheint die Anzeige unruhig, insbesondere bei variierenden Inhalten.
Aufsetzend auf das Aktualisierungsintervall 100 ms des LC-Displays ist in der Hintergrundroutine ein Sekundenzähler implementiert. Er wird verwendet, um einmal in der Sekunde Messwerte und Einstellungen über die USB-Schnittstelle auszugeben. Der Sekundenzähler sorgt auch dafür, dass die Einstellungen nicht zu häufig in das EEPROM geschrieben werden. Wenn der Benutzer einen Wert ändert, müssen die Zwischenwerte nicht gespeichert werden. Erst wenn die Werte für mindestens zehn Sekunden unverändert bleiben, werden sie in das EEPROM gesetzt. Diese Maßnahme wirkt sich auch schonend auf die Lebensdauer des EEPROMs aus.
Wie die Hintergrundschleife die akustischen Signale generiert, soll hier nicht näher betrachtet werden. Zu erwähnen ist nur noch, dass sie auch die über USB empfangenen Kommandos verarbeitet.

Benutzeroberfläche

Das VariLab 402 wird über ein vierzeiliges alphanumerisches LC-Display gesteuert. Wie die **Bilder 1...6** zeigen, besteht die Benutzeroberfläche aus mehreren Seiten. Wenn das Gerät eingeschaltet wird, erscheint zuerst der so genannte Splash Screen. Eigentlich ist dies die Startseite, denn es werden keine Logos oder Symbole „gesplasht“. Stattdessen werden die Elektor-Projektnummer und die Versionsnummer eingeblendet. Nach einigen Sekunden erscheint die Hauptseite mit den für ein Labornetzgerät typischen Daten. Werden jetzt beide Dreh-Encoder gleichzeitig gedrückt, verzweigt die Anzeige zu einer Setup-Seite. Dort sind diverse Parameter einstellbar, zum Beispiel die Ausgabe akustischer Signale oder die Einschaltdauer der Display-Hintergrundbeleuchtung. Durch eine kurze Einschaltdauer der Beleuchtung wird die Stromversorgung des LC-Displays entlastet. Von dieser Seite führen mehrere Wege zu weiteren Seiten: Links (linken Dreh-Encoder drücken) zur Kalibrierung, rechts (rechten Dreh-Encoder drücken) zur Statusseite, und ferner (separaten Taster drücken) zurück zur Hauptseite. Auf der Statusseite sind diverse interne Größen und Werte zusammengefasst. Außer den genannten fünf Seiten existiert eine sechste Seite, denn für die Kalibrierung sind zwei Seiten vorhanden. Dort werden auf einer Seite die unteren Werte von Spannung und Strom kalibriert, auf der anderen Seite die oberen Werte.

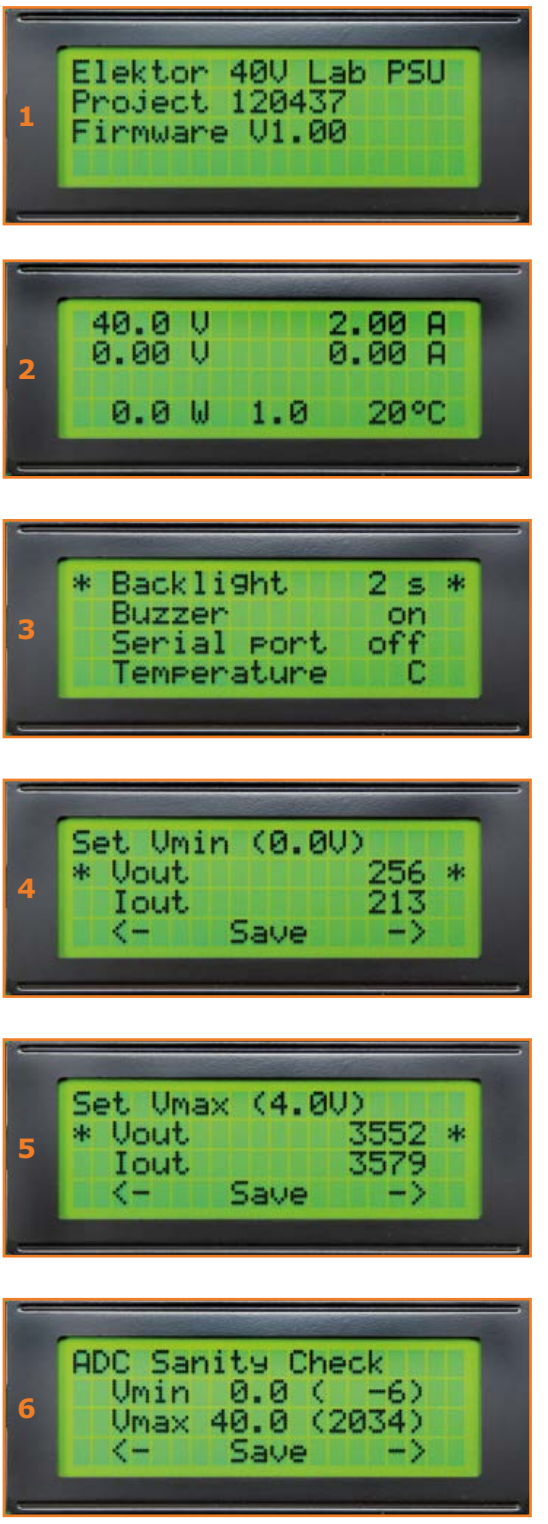


Bild 1...6.
Auf dem LC-Display erscheinen diese Seiten: 1 = Startseite, 2 = Hauptseite, 3 = Setup, 4 = Kalibrierung 1, 5 = Kalibrierung 2, 6 = Status.

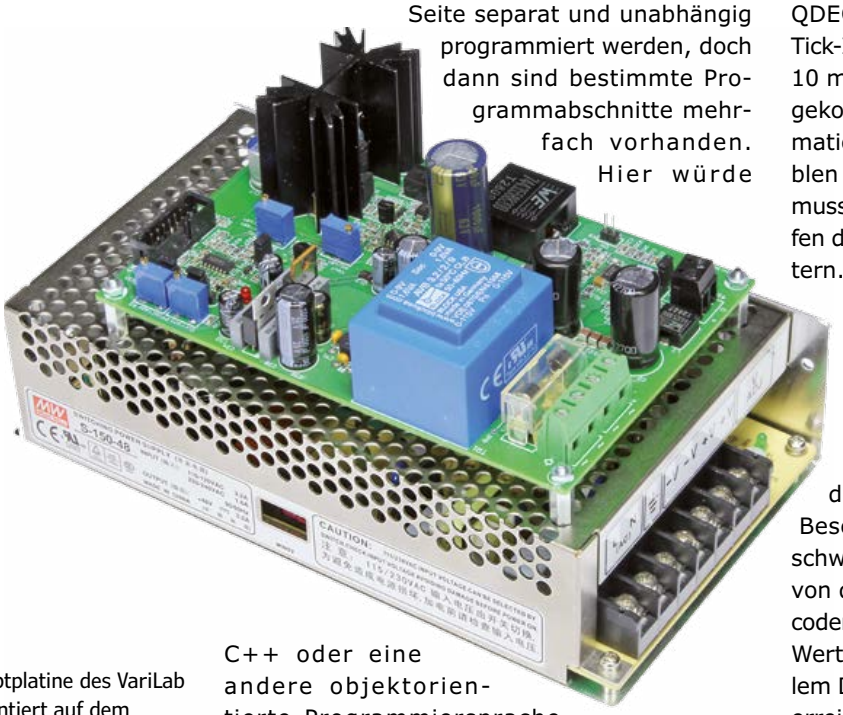


Bild 7.
Die Hauptplatine des VariLab 402, montiert auf dem Schaltnetzteil von Mean Well, das die Eingangsspannung 48 V liefert.

Das Programmieren einer Reihe von Seiten gestaltet sich immer etwas umständlich. Die Seiten haben einiges gemeinsam, beispielsweise die Funktionen der Bedienelemente, aber sie weisen auch Unterschiede auf, zum Beispiel die Anzeigeinhalte. Vom Programmierer kann jede Seite separat und unabhängig programmiert werden, doch dann sind bestimmte Programmabschnitte mehrfach vorhanden. Hier würde C++ oder eine andere objektorientierte Programmiersprache ihre Stärke zeigen, denn die Unterschiede und Gemeinsamkeiten lassen sich dabei in Klassen zusammenfassen. Es wäre auch möglich gewesen, C und C++ nebeneinander zu benutzen. Wir haben uns ausschließlich für C entschieden, zumal auch die ASF-Bibliotheken in C geschrieben sind. Und wir haben C benutzt, um nach dem Vorbild von C++ die Seiten zusammenzufügen. Die Basisklasse ist die Seite (page in page.c und page.h), in der die Funktionen deklariert werden, die andere Seiten verwenden (beispielsweise Initialisieren, Abbilden in der einen oder anderen Form oder Verarbeiten der Tastersignale). Anschließend werden für jede Seite die Unterschiede zur Basisklasse implementiert, gewissermaßen sind dies die abgeleiteten Klassen.

Ein Beispiel soll dies verdeutlichen: In der Datei page_calibrate_dac.c stehen die Funktionen, die nötig sind, um die DAC-Daten anzuzeigen. Die Funktion, die Seite zu aktualisieren, ist dagegen für jede Seite identisch, diese Funktion steht deshalb in der Datei page.c. Wenn die Seite der DAC-Kalibrierung aktiv ist, wird die Aktualisierungsfunktion der Basisklasse in page.c aufgerufen. Diese Funktion ruft anschließend die zugehörigen Funktionen der abgeleiteten Klasse auf.

Eine Anmerkung zu den Dreh-Encodern und insbesondere zu ihrer Kopplung an die globalen Variablen: Der Treiber für die Encoder stammt aus der ASF-Bibliothek, es handelt sich um den QDEC-Treiber (quadrature decoder). Die SysTick-Interruptschleife ruft den Treiber im Abstand 10 ms auf. Ein Encoder ist an die Datenstruktur gekoppelt, in der außer diversen anderen Informationen auch die Adresse der globalen Variablen steht, die vom Encoder bearbeitet werden muss. Die gerade beschriebenen Seiten verknüpfen diese Adressen mit den angezeigten Parametern. Durch solche Verweise wird das Nachvollziehen des Quellcodes zwar erschwert, doch ersparen wir uns zum Beispiel ein unnötiges Kopieren von Daten.

Normalerweise sind rund 100 Umdrehungen des Dreh-Encoders nötig, um die Spannung von 0 V auf 40 V zu stellen. Da uns dies unzumutbar erschien, haben wir einen Beschleunigungsalgorithmus für die Drehgeschwindigkeit eingebaut. Die Schrittweite hängt von der Rotationsgeschwindigkeit des Dreh-Encoders ab. Bei langsamem Drehen ändert sich der Wert mit der kleinsten Schrittweite, bei schnellem Drehen wird der anvisierte Wert im Eiltempo erreicht.

Kalibrierung

Die Kalibrierung des VariLab 402, oder genau genommen seiner Steuerung, ist leider unumgänglich, die Prozedur ist jedoch nicht allzu schwierig. Vorhanden sein müssen ein präzises, verlässliches Multimeter und natürlich ein funktionstüchtiges primäres Netzteil, das die Spannung 48 V liefert. Mit dem Multimeter müssen die Spannungen Vset und Iset gemessen werden, dies muss nicht unbedingt gleichzeitig geschehen. Die Spannungen liegen im Bereich 0...4 V, sie können an K5 gemessen werden.

Die Kalibrierung nehmen Sie wie folgt vor:

Verbinden Sie das Multimeter, Spannungsbereich mindestens 4 V, mit K5, Pin 1 (0 V) und Pin 2 (Iset) oder Pin 1 (0 V) und Pin 3 (Vset).

Schalten Sie das VariLab 402 ein und warten Sie, bis die Hauptseite erscheint.

Drücken Sie gleichzeitig beide Dreh-Encoder, um zur Einstellseite zu gelangen.

Drücken Sie den linken Dreh-Encoder (V), so dass Sie sich auf der ersten Seite der Kalibrierung befinden.

Stellen Sie die minimale Spannung ein. Dazu gehen Sie mit dem linken Dreh-Encoder zur obersten Zeile (Vout, die Auswahl ist durch beidseitige Sternsymbole „*“ markiert). Mit dem rechten Dreh-Encoder stellen Sie die vom Multimeter angezeigte Spannung auf exakt 0,0 V. In identischer Weise verfahren Sie mit Iset in der zweiten Zeile.

Drücken Sie den rechten Encoder (oder den Taster Save), so dass Sie sich auf der zweiten Seite der Kalibrierung befinden.

Stellen Sie die maximale Spannung ein. Dazu gehen Sie mit dem linken Dreh-Encoder zur obersten Zeile (Vout, die Auswahl ist durch beidseitige Sternsymbole „*“ markiert). Mit dem rechten Dreh-Encoder stellen Sie die vom Multimeter angezeigte Spannung auf exakt 4,0 V. In identischer Weise verfahren Sie mit Iset in der zweiten Zeile.

Drücken Sie den rechten Encoder, um zur Status-Seite zu gelangen. Dort müssen jetzt die Werte 0,0 V und 40,0 V angezeigt werden. In den Klammern stehen die binären Werte des D/A-Wandlers. Drücken Sie noch einmal den Taster Save. Wenn Sie Änderungen vornehmen möchten, gelangen Sie mit dem linken Dreh-Encoder zurück.

Drücken Sie den rechten Dreh-Encoder, um die Seiten der Kalibrierung ohne Speichern zu verlassen, falls Sie den Taster Save noch nicht gedrückt haben. Die Software führt nun einen Reset aus. Stellen Sie eine willkürliche Ausgangsspannung ein, zum Beispiel 7,41 V. Schalten Sie den Ausgang frei, indem Sie den Taster drücken. Nun muss die LED aufleuchten. Prüfen Sie mit dem Multimeter, ob der angezeigte Wert mit dem tatsächlichen Wert übereinstimmt. Eine Differenz von einigen zehn Millivolt ist für das VariLab 402 akzeptabel.

Die Funktionen der Bedienelemente dürften schon im Verlauf der Kalibrierung deutlich geworden sein, in **Tabelle 1** sind sie zusammengefasst.

Die Software des VariLab 402 gehört der Kategorie Open-Source an. Wir laden alle interessierten Leser ein, an der Weiterentwicklung mitzuarbeiten. Korrekturen und Vorschläge sind jederzeit willkommen. Das entwickelte Mikrocontroller-Board wird voraussichtlich in weiteren Elektor-Projekten zum Einsatz kommen, es ist universell konzipiert.

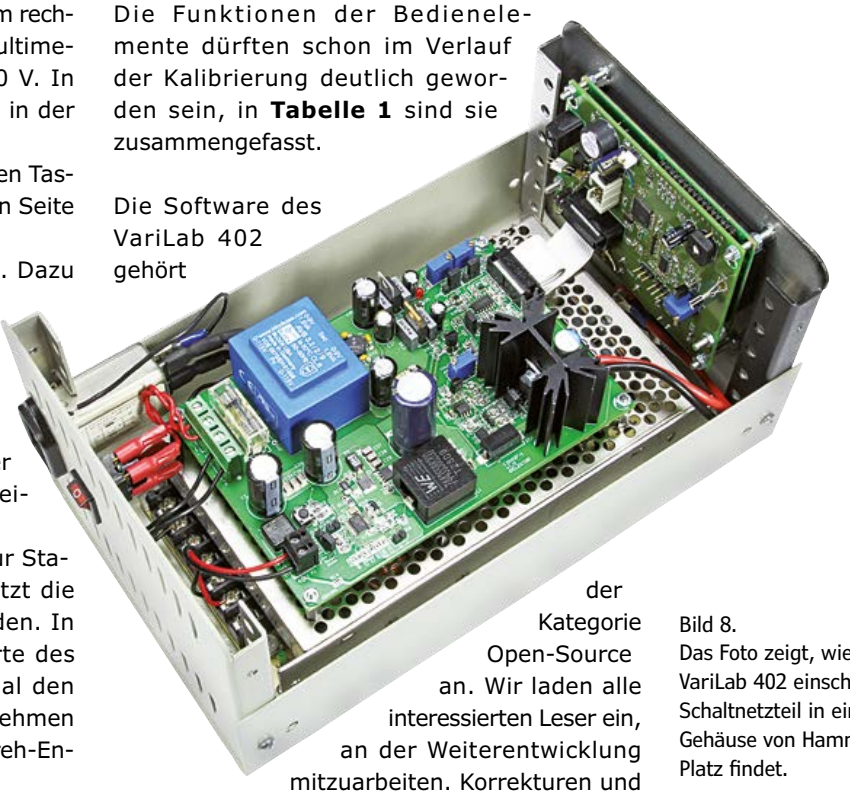


Bild 8.
Das Foto zeigt, wie das VariLab 402 einschließlich Schaltnetzteil in einem Gehäuse von Hammond Platz findet.

Geräteaufbau

Das Foto in **Bild 8** gibt einen Überblick über den Aufbau des VariLab 402. In dem kompakten Gehäuse ist das Netzteil von Mean Well (siehe

Anzeige auf dem LC-Display

Wenn ein Labornetzteil mit einem alphanumerischen Display ausgestattet ist, das 4 · 20 Zeichen darstellen kann, ist die Verlockung groß, alles Darstellbare in die Anzeige zu setzen. Wir haben große Anstrengungen unternommen, dieser Unsitte *nicht* zu folgen. Die Hauptseite der Anzeige sieht deshalb schlicht so aus:

Vset		Iset
Vout		Iout
Power (Vout · Iout)	„Crest”	Temperatur

Der „Crest“-Faktor (Scheitelfaktor) ist hier ein Maß für die Form des Ausgangsstroms. Solange der Strom konstant bleibt, ist der Crest-Faktor gleich 1.

Die Anführungszeichen bedeuten, dass der Wert nicht immer vollständig konform zur Definition Crest=V_{peak}/V_{eff} ermittelt wird. Die Software versucht zwar, diesen Ausdruck zu berechnen, doch die dazu nötigen Filter haben gelegentlich Probleme.

Es ist nicht ganz leicht, die Messperiode hier optimal einzustellen.

Teil 1) auf dem Boden montiert. Das Gehäuse, Abmessungen 127 · 152 · 254 mm, stammt von Hammond, es hat die Typenbezeichnung 1401A. Die Vorderwand und die Rückwand haben wir etwas nach außen gerückt, damit im Gehäuse optimal Platz vorhanden ist.

Die VariLab-402-Platine kann mit Abstandsbolzen auf dem Mean-Well-Netzteil montiert werden, dafür bieten sich Lüftungsöffnungen in der Oberseite an. Auf der Rückseite des Hammond-Gehäuses werden Durchbrüche für den Kaltgeräte-Netzanschluss und den Netzschalter hergestellt. Der Netzanschluss sollte einen Netzfilter enthalten, der Störungen aus dem Stromnetz fernhält. Der Masseanschluss (Erdung) wird mit dem Gehäuse verbunden. Vom Netzanschluss führen kurze Leitungen zum Netzschalter und von dort zur Anschlussklemme K4 auf der Platine. K5 ist die Anschlussklemme, an die der Eingang des Mean-Well-Netzteils angeschlossen wird. Durch diese Leitungsführung werden die beiden Komponenten vom Netzschalter gleichzeitig geschaltet. Alle Leitungen, die Netzspannung führen, müssen ausreichenden Querschnitt haben und wegen der elektrischen Sicherheit doppelt isoliert sein. Schließlich wird noch der Ausgang 48 V des Mean-Well-Netzteils mit K1 auf der VariLab-402-Platine verbunden.

In der Frontplatte des Gehäuses werden Ausschnitte für das LC-Display und die USB-B-Buchse angebracht. Achtung: Kein Teil der USB-Buchse darf mit dem Gehäuse in Kontakt stehen! In der Frontplatte fehlen noch die Bohrungen für die LED, die Dreh-Encoder, den Drucktaster und die beiden 4-mm-Buchsen des Ausgangs. Die Verbindungen zwischen den Platinen stellt ein 14-adriges Flachkabel mit zuvor aufgepressten Buchsenleisten her. Wenn der Temperatursensor (IC5) auf dem Steuer- und Anzeigemodul bleibt, misst er die mittlere Temperatur im Gehäuse. Er kann aber auch auf der Hauptplatine am Kühlkörper von T4 montiert und über Leitungen angeschlossen werden. Zum Schluss werden die Ausgangsbuchsen über Leitungen, Querschnitt 2,5 mm², mit Klemme K2 verbunden. Diese Leitungen werden dicht aneinander geführt und durch einen Ferritring geschlungen, der hochfrequente Störsignale dämpft.

Nachdem die Montage abgeschlossen ist und die beschriebene Kalibrierung durchgeführt wurde, ist das VariLab 402 einsatzbereit.

(140431)gd

Weblinks

[1] www.elektor-magazine.de/140373

Tabelle 1. Bedienfunktionen.

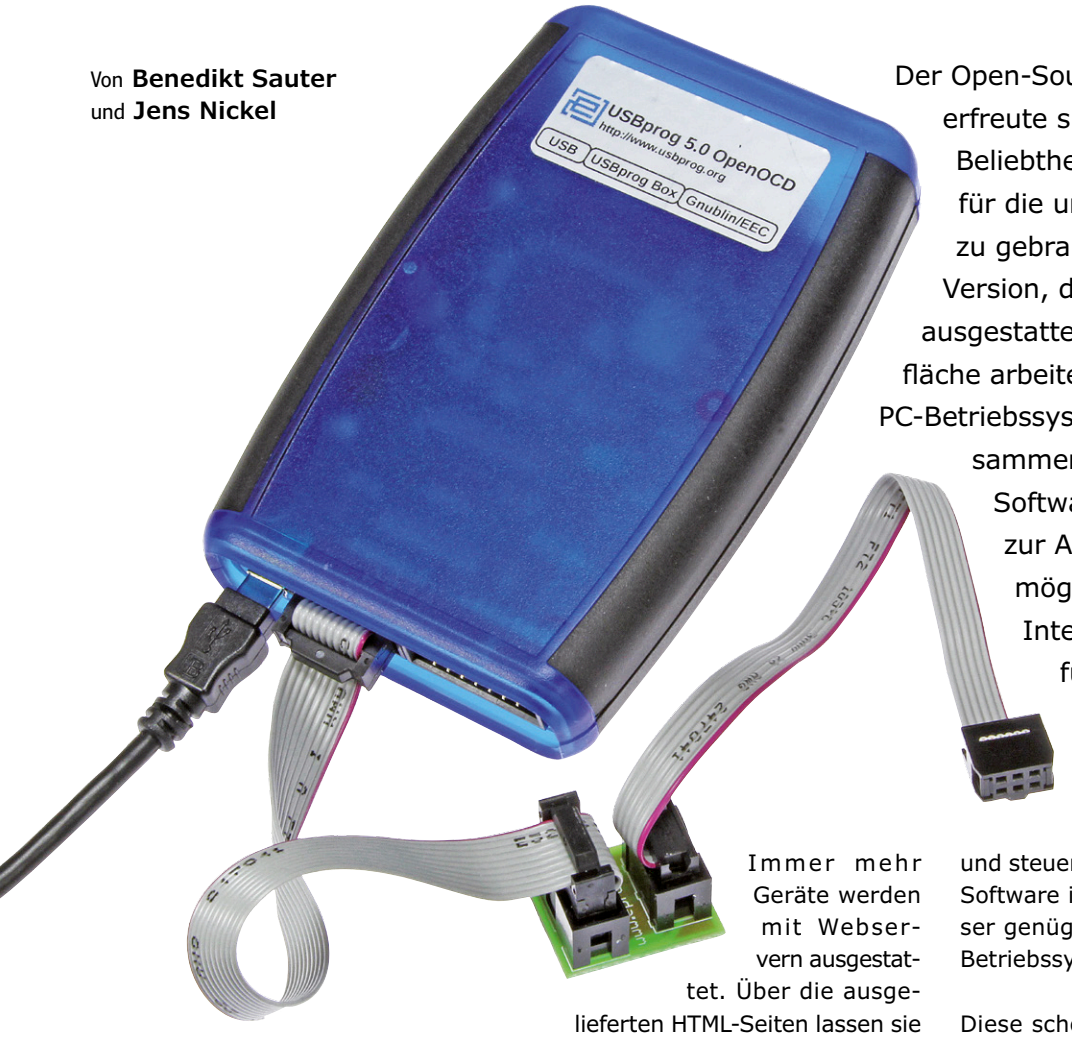
	Hauptseite	Setup-Seite	Kalibrierung	Status-Seite
Linker Dreh-Encoder	Ausgangsspannung	Parameter-Wahl	Parameter-Wahl	-
Rechter Dreh-Encoder	Maximaler Strom	Parameter-Werte	Parameter-Werte	-
Drucktaster Linker Dreh-Encoder	Backlight kurz ein. Aktiviert Setup-Seite, wenn gleichzeitig Taster des rechten Dreh-Encoders gedrückt wird	Zur Kalibrierung	Zur vorigen Seite	Zur Hauptseite
Drucktaster Rechter Dreh-Encoder	Backlight kurz ein. Aktiviert Setup-Seite, wenn gleichzeitig Taster des linken Dreh-Encoders gedrückt wird	Zur Status-Seite	Zur nächsten Seite	Zur Hauptseite
Drucktaster	Schaltet den Geräteausgang frei	Zur Hauptseite		Zur Hauptseite

- Die LED leuchtet auf, wenn der Ausgang freigeschaltet ist.
- Der Buzzer bestätigt jede Aktion mit einem Quittungston. Eine Folge von drei Tönen bedeutet, dass die Strombegrenzung wirksam wurde.

USBprog 5.0

Open-Source-Programmer mit Webinterface

Von **Benedikt Sauter** und **Jens Nickel**



Der Open-Source-Programmer USBprog erfreute sich bei Elektor-Lesern großer Beliebtheit, denn er war als „Multi-Tool“ für die unterschiedlichsten Controller zu gebrauchen. Hier kommt eine neue Version, die sogar mit einem Webserver ausgestattet ist. Dank der HTML-Oberfläche arbeitet der Programmer mit allen PC-Betriebssystemen und sogar Tablets zusammen, auch die Installation eigener Software entfällt. Eine Schnittstelle zur Automatisierung, eine Speichermöglichkeit für Hex-Files und die Integration des GDB-Debuggers für ARM sind weitere interessante Features.

Immer mehr Geräte werden mit Webservern ausgestattet. Über die ausgelieferten HTML-Seiten lassen sie sich dann von einem Computer aus konfigurieren und steuern. Auf dem Rechner muss dazu keine Software installiert werden, ein Standardbrowser genügt. Das Ganze funktioniert mit jedem Betriebssystem und auch vom Tablet aus.

Diese schöne Idee hat die Embedded Projects GmbH nun auf die Welt der Programmer ausgedehnt. Das Flashen und Auslesen von Firmware, das Setzen und Lesen von Fusebits und dergleichen mehr, das alles kann beim USBprog 5.0 über eine einfache HTML-Benutzeroberfläche erfolgen. Selbstverständlich hat das Team um Benedikt Sauter auch bei der neuesten Version des bekannten USBprog wieder voll auf den Open-Source-Gedanken gesetzt (siehe unten).

Eigenschaften

- AVR-Programmer (avrdude [5])
- ARM-JTAG-Debugger und -Programmer (openocd [4])
- Updatefähigkeit: Weitere unterstützte Controller in Vorbereitung
- Pegelwandler (einstellbar 1,8 V, 3,3 V und 5,0 V)
- Browseroberfläche für einfache Bedienung
- Automatische Bedienung per Kommandozeilen-Tool
- Verwendung aus Atmel Studio und anderen Programmen heraus
- Speichermöglichkeit für häufig verwendete Hexfiles

Anschluss am PC

Der USBprog 5.0 ist nichts anderes als ein kleines Linux-Board im formschönen Gehäuse. Der

kleine Kasten wird über USB mit dem PC verbunden und hierüber auch mit Strom versorgt. Per USB wird eine Netzwerkschnittstelle emuliert; unter Mac und Linux wird das Gerät automatisch gefunden und verbunden. Auf dem USBprog läuft ein DHCP-Server, der dem PC eine IP-Adresse zuteilt. Danach kann über die Adresse 10.0.0.1 per Browser auf den Programmer zugegriffen werden.
Unter Windows springt beim ersten Anschließen der bekannte Treiber-Dialog auf. Hier gibt man an, dass man einen Treiber manuell auswählen möchte. Als Geräteart wählt man „Network adapters“ bzw. „Netzwerkkarten“. Anschließend erscheint eine Auswahl von verschiedenen Herstellern. Hier wählt man „Microsoft Corporation“ und „Remote NDIS Compatible Device“.

Für ARM und AVR

Nachdem die Netzwerkverbindung steht, kann man sich über einen Browser (http://10.0.0.1) mit dem Programmer verbinden und sieht eine Oberfläche wie in **Bild 1**. Anschließend verbindet man den USBprog mit dem Zielprozessor; das Target wird dann auch mit Strom versorgt. Aktuell werden ARM-Prozessoren und Atmel-Mikrocontroller unterstützt; später werden noch PIC-Controller hinzukommen, ein Firmware-Update ist bereits in Vorbereitung.

Für die AVR-Controller verfügt der Programmer über den üblichen 10-poligen ISP-Wannenstecker (**Bild 2**), über ein Flachkabel geht es weiter zum entsprechenden Gegenstück auf dem Zielboard. Um sich auch mit 6-poligen ISP-Steckern verbinden zu können, bringt der USBProg eine kleine Adapterplatine mit. Die Wannenstecker muss man sich selbst einlöten, was aber keine großen Lötkünste erfordert.
Zum Flashen und Debuggen von ARM-Controllern ist ein Adapter auf einen 20-Pin-JTAG-Steckverbinder erforderlich, der ebenfalls zum Selbst-Löten mitgeliefert wird.
Daneben ist der updatefähige Programmer bereits mit einem 14-poligen Gnublin/EEC-Steckverbinder ausgestattet, über den sich später einmal die bekannten Gnublin-Erweiterungsboards steuern lassen werden – natürlich ebenfalls per Weboberfläche.

Manuelle Bedienung

In der Browseroberfläche wählt man nun zuerst den Prozessor aus, hierfür ist die Auswahlbox ganz oben zuständig. Um beispielsweise einen ATmega328P (Arduino Uno [1]) oder ATmega328 (T-Board 28 [2]) auszuwählen, reichen jeweils ein paar Buchstaben. Mit den Radio-Buttons rechts daneben wird die Spannung des Zielprozessors ausgewählt.

Zuerst sollte man probeweise einmal die Signatur des Controllers oder die Fuses auslesen. Es sollte

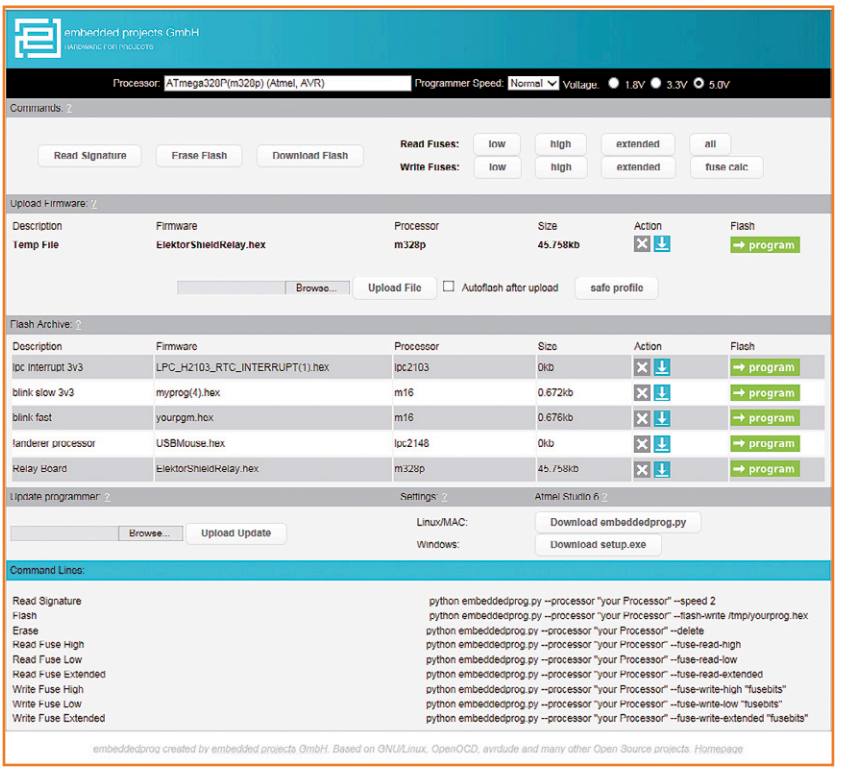


Bild 1.
Der Programmer lässt sich über eine HTML-Oberfläche steuern, die in jedem Browser dargestellt werden kann

Bild 2.
Anschlüsse des USBprog 5.0. Rechts erkennt man den 14-poligen Gnublin/EEC-Steckverbinder; nach einem Firmware-Update werden sich später auch einmal die Gnublin-Erweiterungsboards ansteuern lassen.

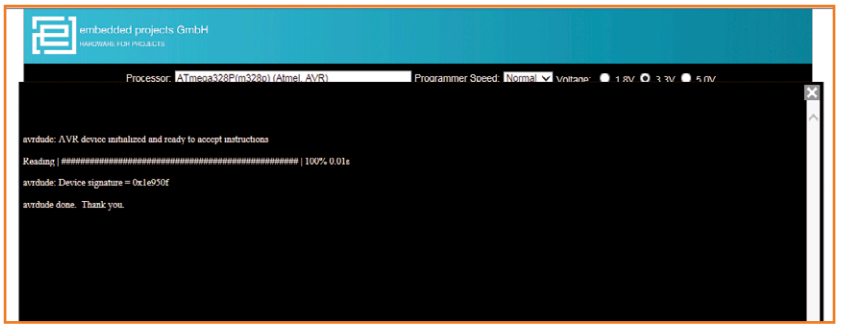
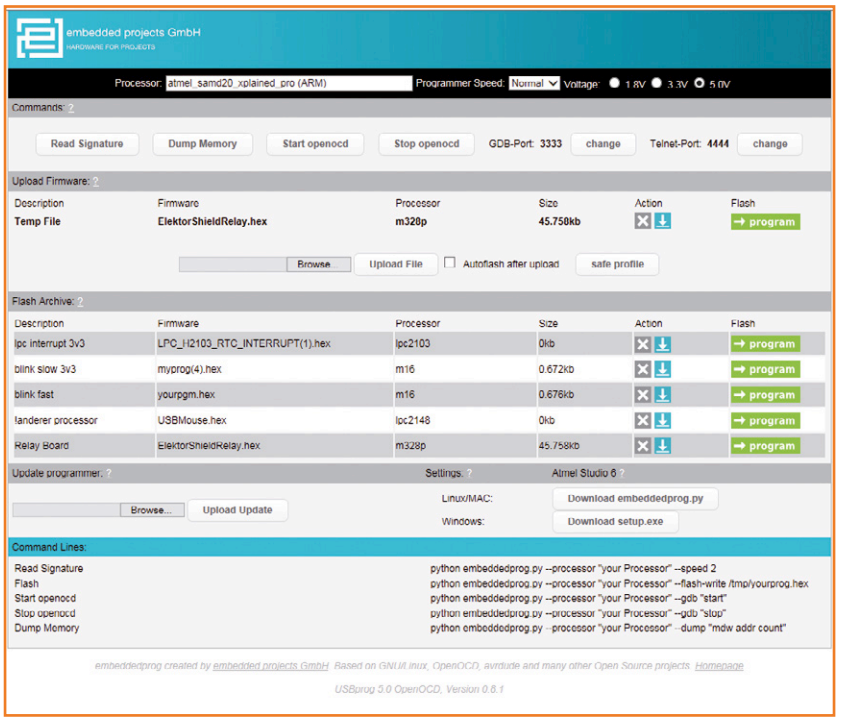


Bild 3.
Wie man hier sieht, nutzt der Programmer das Open-Source-Programmierinterface avrdude.

sich nach ein paar Sekunden jeweils ein schwarzes Textfenster mit dem Ergebnis öffnen (**Bild 3**). Danach geht es an das Programmieren eines ersten Hex-Files, hierfür ist der Abschnitt „Upload Firmware“ der Benutzeroberfläche zuständig. Mit „Browse“ wählt man ein File aus, mit „Upload File“ wird dieses zuerst auf den Programmer gespielt und schließlich mit dem grünen Button „-> program“ auf den Zielprozessor übertragen.

Man kann das Hex-File auch zusammen mit einem kurzen Titel im Programmer abspeichern, wenn man auf den Button „safe profile“ klickt. Die abgespeicherten Files erscheinen im Abschnitt „Flash Archive“. So lassen sich zum Beispiel mehrere Kleinserien von Produkten recht rasch mit der jeweiligen Firmware versorgen.

Bild 4.
Die Oberfläche nach dem Auswählen eines ARM-Controllers.



Bedienung per Skript

Die manuelle Bedienung des Programmers im Webbrowser ist komfortabel, deckt aber noch nicht alle Anwendungsszenarien ab. Beim Entwickeln von Firmware arbeitet man meist mit einer Umgebung wie Atmel Studio; da spart es Zeit, wenn man aus der IDE heraus das kompilierte Programm auch gleich in den Controller flashen kann. Auch das automatisierte Flashen per Batch-Datei oder eigener PC-Software sollte mit dem Programmiergerät natürlich möglich sein.

Daher hat das Team um Benedikt Sauter ein kleines Tool in Python geschrieben, das auf dem Entwicklungsrechner per Kommandozeile aufgerufen wird. Ein Vorteil von Python ist, dass es Interpreter für alle gängigen Betriebssysteme gibt. Auf einem Mac oder Linux-Rechner kann man „embeddedprog.py“ direkt ohne zusätzliche Installation nutzen. Dort gehört Python zu den Standards des Betriebssystems. Man muss lediglich das Python-Tool selbst auf den Rechner spielen, was man über die Browseroberfläche mit dem Button „Download embeddedprog.py“ erledigen kann.

Auf der Kommandozeile ruft man nun den Python-Interpreter auf, der das Python-Programm ausführt. Der Prozessor und die Aktion, die der Programmer ausführen soll, werden als Parameter mitgegeben. Um beispielsweise die Signatur eines ATmega328P auszulesen, lautet der Aufruf (aus dem Ordner heraus, wo das Tool abgelegt ist):

```
python embeddedprog.py --processor m328p
--speed 2
```

Weitere Kommando-Parameter sind auf der Web-Oberfläche unten aufgeführt.
Wichtig: Beim ersten Aufruf des Python-Tools muss einmalig die IP-Adresse des Programmers angegeben werden. Diese wird anschließend lokal in einer Konfigurationsdatei gespeichert und ab da automatisch mit angefügt.

```
python embeddedprog.py --eeprog-ip
10.0.0.1 --eeprog-port 8888 --processor
m328p --speed 2
```

Unter Windows muss zuerst Python installiert werden [3]. Das Python-Tool embeddedprog.py kann man sich herunterladen und an passender Stelle ablegen, auf der Kommandozeile

muss dann aber der Pfad auf den Python-Interpreter und auf das Python-Tool mitangegeben werden. Komfortabler fährt man mit einer Exe-Datei, die auf dem Windowsrechner installiert wird (die passende Setup-Datei lädt man sich über den Button „Download setup.exe“ herunter). Bei der Installation werden Windows auch die Pfade bekannt gemacht; außerdem wird die Konfigurationsdatei mit der IP-Adresse angelegt. Auf der Windows-Kommandozeile (die man im Startmenü mit „cmd“ erreicht) ruft man nun einfach die Exe-Datei auf:

```
embeddedprog.exe --processor m328p
--speed 2
```

Mit vielen Programmiersprachen lassen sich Kommandozeilenaufrufe absetzen (und die Ausgabezeilen auswerten), so dass man den Programmer mit selbst geschriebener Software ansteuern kann. In einer der nächsten Ausgaben stellen wir eine entsprechende Erweiterung des EFL-Konfigurators vor. Auch die Bedienung direkt aus Atmel Studio 6 heraus ist dank des Python-Tools möglich (siehe Kasten).

Debuggen von ARM-Prozessoren

Der neue USBprog kann auch zum Debuggen von ARM-Prozessoren verwendet werden. Hierfür wird der Open-Source-Debugger openocd [4] genutzt, der auf dem Programmer läuft. Vor dem Debuggen muss man die Firmware in den Prozessor übertragen. Anschließend startet man den Debugger per Browser („Start openocd“, siehe **Bild 4**) oder über die Kommandozeile.

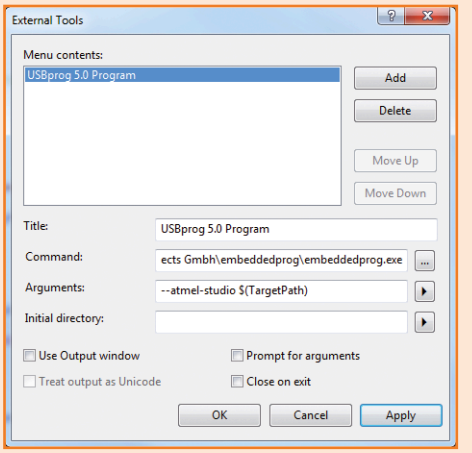
Der Debugger kann entweder über ein einfaches Telnet-Interface oder über eine GDB-Schnittstelle gesteuert werden. Größere Entwicklungsumgebungen bieten hier häufig ein entsprechendes Interface. Im Internet findet man viele Anleitungen, wenn man nach „ARM Eclipse GDB“ sucht. Als „Remote Address“ muss man anstelle des Eintrags „localhost“ die IP-Adresse des Debuggers eintragen (10.0.0.1).

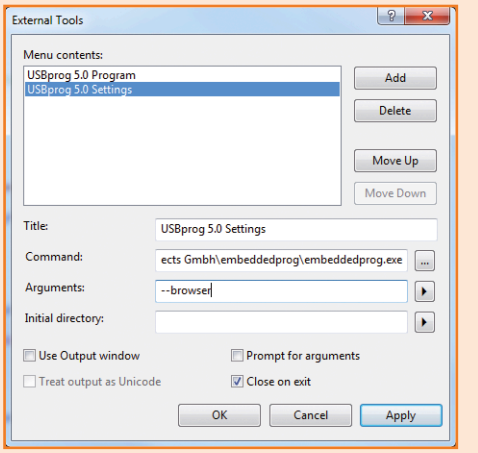
Das Innenleben

Bild 5 zeigt einen Blick ins Innere des Programmers: Der Hauptprozessor ist ein LPC3131, dem Arbeitsspeicher vom Typ A43L4616 (8 MB) zur Seite gestellt ist. Für die Pegelwandlung wurde

Zusammenarbeit mit Atmel Studio

Was wäre ein Programmer für AVR-Controller, der nicht auch aus der kostenlosen Entwicklungsumgebung Atmel Studio heraus bedienbar wäre? Das klappt auch beim USBprog - dank der Ansteuerung per Kommandozeilen-Tool (siehe Text). Zwei Aktionen dürften dabei am wichtigsten sein: nämlich das Flashen von Hexfiles auf den AVR-Controller und das Aufrufen der Browser-Oberfläche von Atmel Studio aus (zum Beispiel um Settings einzugeben).





Für beide Aktionen kann man in Atmel Studio unter Tools -> External Tools einen Menüpunkt anlegen. Nach einem Klick auf den Button „Add“ ist hierfür jeweils in den Textboxen „Title“, „Command“ und „Arguments“ ein Eintrag vorzunehmen, danach klickt man auf „Apply“. Wir beginnen mit „USBprog 5.0 Program“ für das Uploaden von Firmware (siehe Screenshot). Unter „Command“ ist jeweils der gesamte Pfad auf das Tool embeddedprog.exe (siehe Text) einzutragen.
Hat man für beide Aktionen einen Eintrag wie in den Screenshots gezeigt angelegt, dann erscheinen die beiden Einträge im Atmel-Studio-Hauptmenü unter „Tools“.
Betätigt man zum ersten Mal „USBprog 5.0 Program“, dann springt die Browseroberfläche auf. Hier muss man zuerst den Prozessor und dessen Betriebsspannung einstellen. Ab jetzt kann man über diesen Menüpunkt direkt aus Atmel Studio heraus Hexfiles in den Controller flashen.

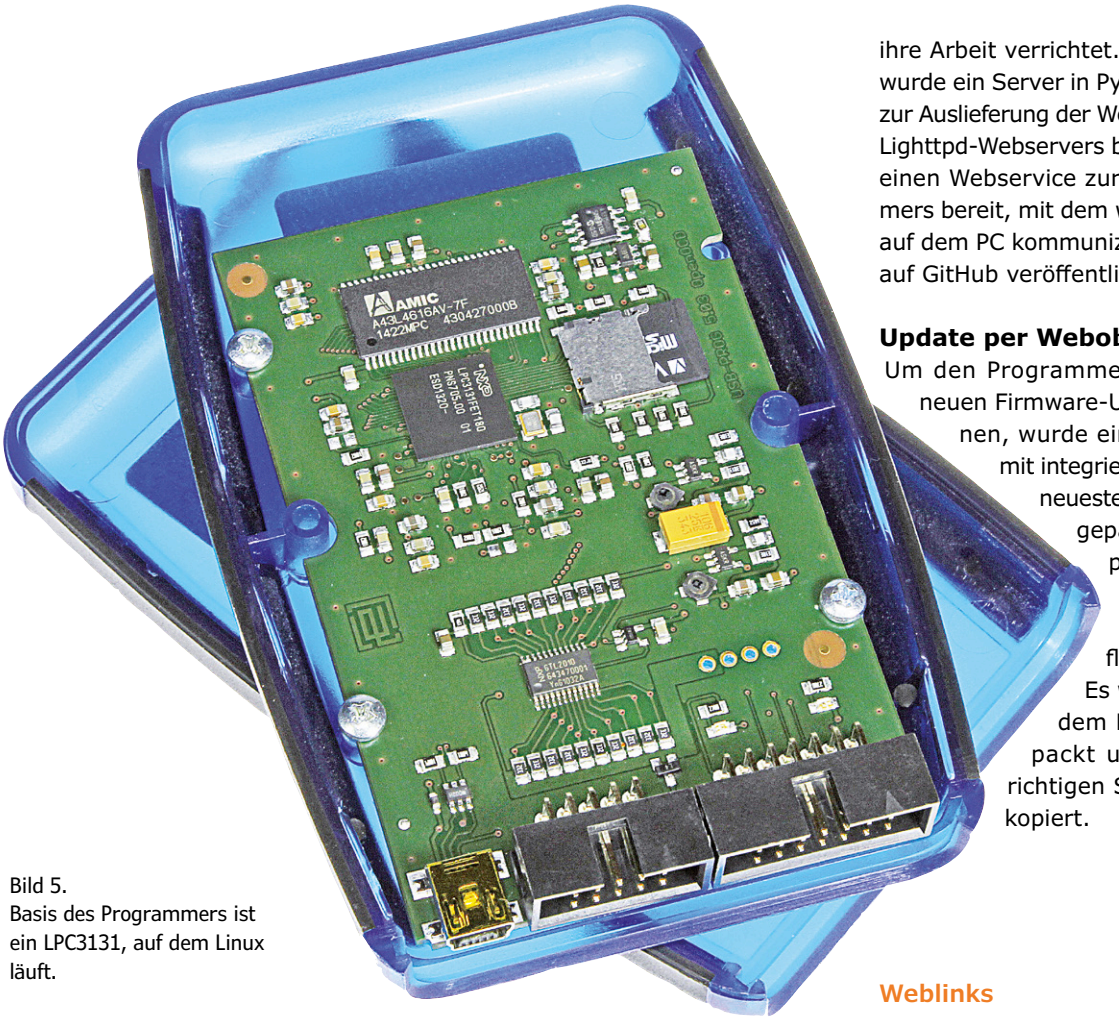


Bild 5.
Basis des Programmers ist ein LPC3131, auf dem Linux läuft.

der Baustein GTL2010PW ausgewählt. Er ermöglicht eine einfache bidirektionale Wandlung der Signale. Um die Spannung für die Zielhardware (1,8 V, 3,3 V und 5,0 V) per Software auswählen zu können, wurde ein LDO AP2127K-ADJ verwendet, der von einem digitalen Poti MCP4131 angesteuert wird. Bis zu 300 mA Ausgangsspannung sind möglich.

Auf dem Prozessor läuft ein Embedded-Linux, auf dem wiederum Standardsoftware wie avrdude (für das Programmieren der Atmel-Prozessoren [5]) ihre Arbeit verrichtet. Als zentrale Komponente wurde ein Server in Python geschrieben, der sich zur Auslieferung der Weboberfläche wiederum des Lighttpd-Webservers bedient. Außerdem stellt er einen Webservice zur Steuerung des Programmers bereit, mit dem wiederum die Python-Tools auf dem PC kommunizieren. Alle Quelltexte sind auf GitHub veröffentlicht [6].

Update per Weboberfläche

Um den Programmer in Zukunft einfach mit neuen Firmware-Updates versorgen zu können, wurde eine Updatefähigkeit gleich mit integriert. Hierzu lädt man sich die neueste Version des Updates (eine gepackte Datei) von der USBprog-Webseite [7] herunter. Das Update kann dann einfach über die Browseroberfläche eingespielt werden. Es wird nach dem Upload auf dem Programmieradapter entpackt und entsprechend an die richtigen Stellen im Betriebssystem kopiert.

(140285)

Weblinks

[1] www.elektor.de/arduino-uno
[2] www.elektor.de/t-board-28-130581-93
[3] www.python.org
[4] www.openocd.org
[5] www.nongnu.org/avrdude
[6] https://github.com/embeddedprojects/usbprog5
[7] http://usbprog5.embedded-projects.net

Präzise Ausgangspannung

Ohne integrierte Spannungsregler wäre die Stromversorgung elektronischer Systeme wesentlich aufwendiger. Spannungsregler sind seit Jahrzehnten im Einsatz, mit festen oder einstellbaren Ausgangsspannungen, für kleine oder große Leistungen.
Ein wahrer Oldtimer unter den Spannungsreglern ist der LM317, er wurde von National Semiconductor im Jahr 1971 auf den Markt gebracht. Das interne Chip-Layout des LM317 ist zwar heute nicht mehr mit dem Original identisch, doch die elektrischen Eigenschaften, die Anschlussbelegung und die Beschaltung sind unverändert geblieben.
Beim LM317 kann die Ausgangsspannung, wie **Bild 1** zeigt, mit zwei am Eingang „Adjust“ liegenden Widerständen festgelegt werden. Die Eingangsspannung darf bis 40 V betragen, solange die Spannungsdifferenz zwischen Eingang und Ausgang 15 V nicht übersteigt. Der hindurchfließende Strom erreicht seine Grenze bei 2 A.
Die Ausgangsspannung des LM317 lässt sich hochpräzise einstellen, wenn die interne Referenzspannung bekannt ist. Abhängig vom Hersteller liegt diese Spannung zwischen 1,2 V und 1,3 V. Gemessen wird die interne Referenzspannung, indem das aktuelle Exemplar wie in **Bild 2** skizziert beschaltet wird. Widerstand R1 kann einen Wert im Bereich 240...470 Ω haben. An den Eingang wird eine Gleichspannung größer 3 V und kleiner 10 V gelegt. Die Eingangsspannung muss mindestens 3 V betragen, weil der LM317 für die Funktion als Regler dieses Spannungsgefälle braucht. In der skizzierten Konfiguration ist die Spannung am Reglerausgang gleich der internen Referenzspannung. An Exemplaren einzelner Hersteller wurden folgende Referenzspannungen gemessen: ST317: 1,249 V, UA317: 1,275 V, SSS317: 1,231 V.
Ferner ist zu berücksichtigen, dass der Anschluss „Adjust“ einen Strom von ungefähr 50 µA abgibt, der Strom fließt ebenfalls durch Widerstand R2 in Bild 1. Auch dieser Strom kann abhängig vom Hersteller variieren, ein Blick in das zugehörige Datenblatt gibt Auskunft.

Die Ausgangsspannung eines LM317 lässt sich wie folgt berechnen:

$$R2_{theoretisch} = (U_{out}/U_{ref} - 1) \cdot R1$$

$$R2_{adjust} = (U_{out} - U_{ref}) / I_{adjust}$$

$$R2_{total} = R2_{theoretisch} \cdot R2_{adjust} / (R2_{theoretisch} + R2_{adjust})$$

Es liegt nahe, R2 als Mehrgang-Trimmpoti auszuführen und so einzustellen, dass die Ausgangsspannung exakt dem gewünschten Wert entspricht. Da aber auch das Berechnen der Widerstandswerte R1 und R2 als Festwiderstände zum exakten Ergebnis führt, kann das kostenträchtige Mehrgang-Trimmpoti eingespart werden. Die gleiche Überlegung gilt uneingeschränkt auch für den negativen Spannungsregler LM337. Vorstehend wurde vorausgesetzt, dass die Betriebstemperatur des Spannungsregler-Chips konstant ist. Genau betrachtet driften bei Temperaturänderungen in geringem Maß sowohl die Referenzspannung U_{ref} als auch der Einstellstrom I_{adjust}, die Ausgangsspannung hängt auch vom Ausgangsstrom ab.

(140341)gd

Nach einer Idee von
Peter Krueger (D)

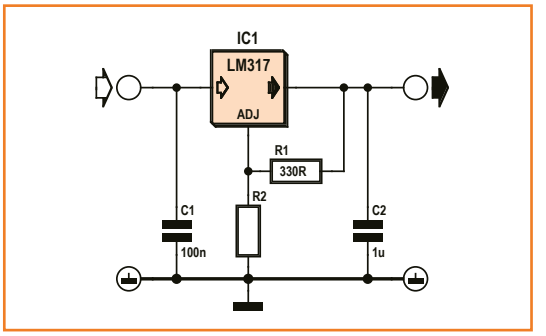


Bild 1.
Grundschaltung des Spannungsreglers LM317.

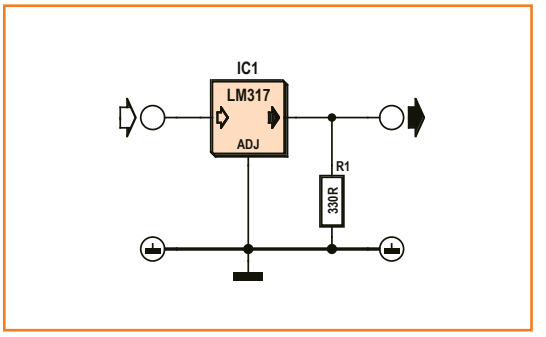


Bild 2.
Wenn Anschluss „Adjust“ an Masse liegt, ist die interne Referenzspannung am Ausgang messbar.

Modernes Messwerk in altem Gewand

Neues Innenleben für alten Tacho

Von **Michael Möge**

Zeigerinstrumente sind Relikte aus alten Elektronikzeiten und nicht mehr zeitgemäß, aber oft einfach nur schön, moderne Anzeigen dagegen sehr funktionell und meist hässlich. Was spricht also dagegen, ein altes Zeigerinstrument mit einem modernen Innenleben auszustatten? Dann gehen Funktionalität und Ästhetik zusammen.

Für ein älteres Sportboot sollte der alte mechanische Geschwindigkeitsmesser, das Sumlog, stilecht erneuert werden. Hightech in alter Optik eben. Der neue Geber unter dem Boot ist nicht mehr über eine Welle, sondern über ein Kabel mit der Anzeige verbunden. Über dieses Kabel wird ein von einem Reedkontakt erzeugtes Rechtecksignal mit zur Schiffsgeschwindigkeit proportionaler Frequenz (10 Hz bis 60 Hz) geleitet.
Die Aufgabe war nun, eine Elektronik zu bauen, die die Frequenz des Rechtecksignals misst und das Resultat so aufbereitet, dass der Wert in einem Rundinstrument mit Zeiger und mindestens 270° Anzeigewinkel angezeigt wird. Dabei sollte sich das alte Sumlog-Gehäuse samt schwerem Metallzeiger und Skalenoptik weiter verwenden lassen. Und wie es sich für ein Sumlog gehört, sollte auch der zurückgelegte Weg nicht rückstellbar angezeigt werden.

Vom 555 zum Mikrocontroller

Erste analoge Versuche mit einem NE555 und einer einfachen RC-Integration waren rein elektrisch durchaus erfolgreich, die Anzeige mit handelsüblichen 270°-Drehspulinstrumenten aber nicht. Keines der Messwerke war in der Lage, den schweren Metallzeiger sicher zu tragen. Dafür geeignete Messwerke waren zu groß oder hatten einen Drehwinkel von nur 90°. Also musste das ganze Konzept grundlegend geändert werden.
So wurde ein Mikrocontroller zur Auswertung des Gebersignals eingesetzt. Ein vom Controller angesteuerter Modellbauservo als Messwerkersatz war schon ein viel versprechender Ansatz, aber noch nicht ganz befriedigend. Servos mit einem Stellwinkel von 270° sind sehr schwer erhältlich, üblich sind maximal 180°. Ein zusätzliches Getriebe wäre zu aufwändig. Die Lösung brachte ein spezieller Schrittmotor, wie er auch so ähnlich von Autobauern für Tachoanzeigen verwendet wird. Solche Schrittmotoren sind erstaunlich stabil und für kleines Geld als Kfz-Ersatzteil, bei eBay oder gegebenenfalls auf dem Schrottplatz erhältlich.
Ich habe den verwendeten Schrittmotor (Stepper) bei eBay erstanden [1]. Er dreht nicht rund um 360°, sondern in 1260 Schritten um 315° von Anschlag zu Anschlag. Er besitzt solide Endanschläge und ein großes Stell- und Haltemoment. Allerdings gibt es keine Rückmeldung über die Zeigerposition. Beim Start wird der Stepper unabhängig von der tatsächlichen Position des Zeigers einfach um 1260 Schritte nach links gegen den Anschlag gedreht. Er steht dann nach circa 0,5 s definitiv auf 0. Der Stepper wird in acht Phasen über zwei Spulen bewegt, die direkt von einem ATtiny2313 von Atmel angesteuert werden können.

Die Hardware in **Bild 1** ist sehr übersichtlich und so gestaltet, dass entweder ein Servo (dann mit 180°-Anzeige) oder der Stepper mit 315°-Anzeige eingesetzt werden kann. Um den ATtiny2313 herum sind die übliche Reset-Anordnung, der Quarz und mit R6/C2 ein einfaches RC-Eingangsfilter für den Sensoreingang angeordnet. Je nach Bedarf können entweder die Spulen des Schrittmotors oder der Servo direkt am Prozessor angeschlossen werden. Statt des speziellen Steppers wäre auch ein „normaler“ kleiner bipolarer Schrittmotor denkbar, wenn der 0-Punkt mit einem festen mechanischen Anschlag fixiert wird. Ein Schrittmotor ohne Getriebe und 1,8°-Schritten löst die 315° Anzeigewinkel in 175 Schritte auf. Die Kraftentwicklung ist in der Regel so groß, dass mit deutlich reduziertem Spulenstrom gearbeitet werden kann. Gleichwohl ist eine direkte Ansteuerung der Spulen durch den Controller nicht üblich.
Nicht nur die Geschwindigkeit, auch der Weg soll angezeigt werden. Dafür wird ein elektromagnetisches Zählwerk verwendet, das den Anzeigewert dauerhaft und nicht rückstellbar ausgibt. Im Prozessor werden die Geberpulse summiert und bei Erreichen einer vorgegebenen Zahl ein 10-ms-Impuls an PB1 ausgegeben. Zählwerke für 5 VDC sind kaum bis nicht erhältlich, die meisten elektromechanischen Zähler [2] arbeiten in einem Gleichspannungsbereich von 10...30 V. Die Impulse des Controllers werden deshalb mit der Schaltung um T1 und T2 verstärkt. Der verwendete Zähler ist ein 24-V-Modell, andere Zählwerke könnten unter Umständen auch direkt vom Controller oder nur mit T1 angesteuert werden.
Die kleine Schaltung, auf einer Lochrasterplatine aufgebaut, fand im vorgegebenen Gehäuse ausreichend Platz (**Bild 2**). Der schwere Metallzeiger wird problemlos vom Steppermotor und selbstverständlich auch vom Servo bewegt. Ein zur Sicherheit installierter Resetknopf erwies sich in der Praxis als überflüssig. Selbst bei starken Stößen und Schwingungen hält der Stepper den Zeiger fest am Platz.

Die Software

Die Software ist natürlich noch interessanter als

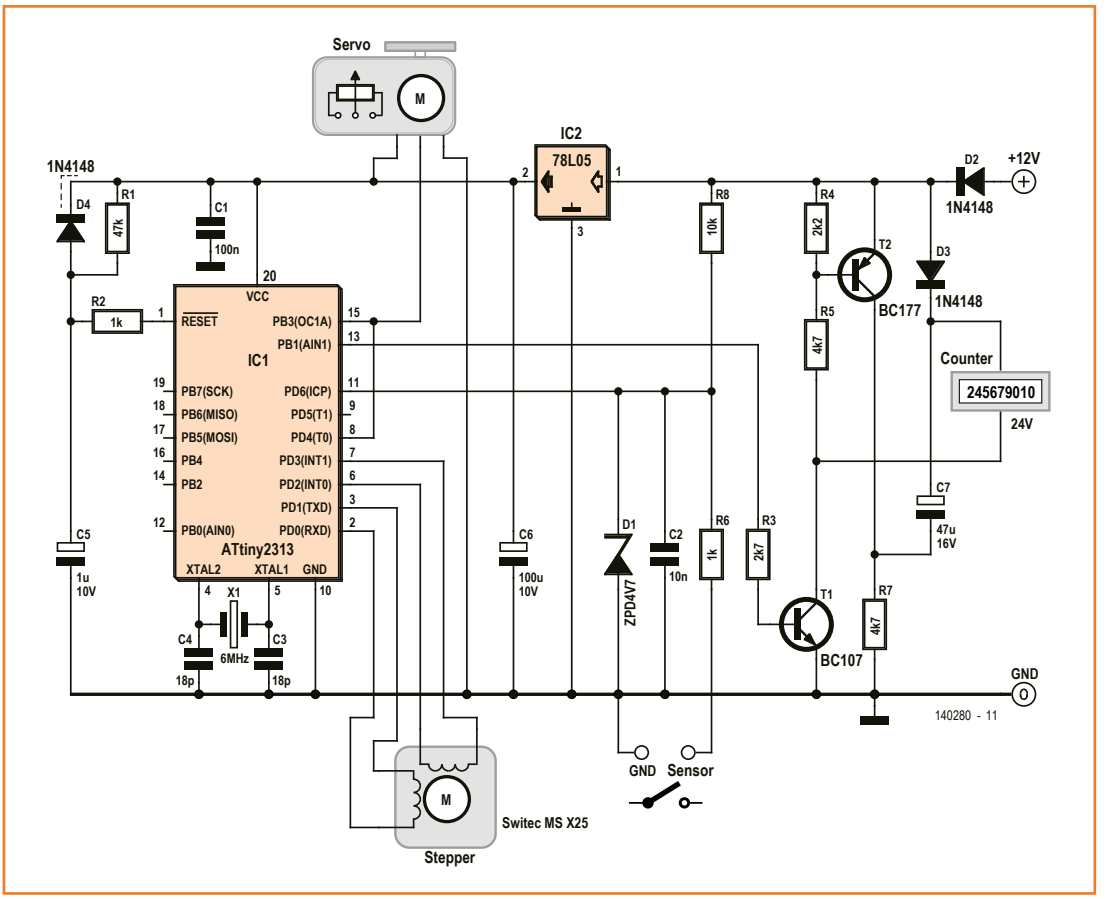


Bild 1.
An den Controller kann ein Servo- oder ein Schrittmotor angeschlossen werden.

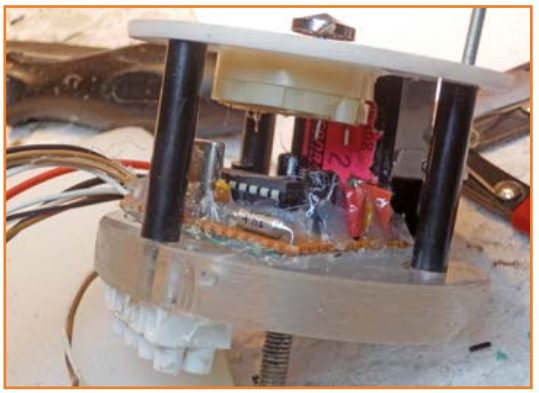


Bild 2.
Die Elektronik passt perfekt in das alte Gehäuse.

die Hardware und soll, da sie je nach den vom Leser verwendeten Bauteilen angepasst werden muss, ausführlich besprochen werden (sie ist zudem reichlich kommentiert). Sie kann von [3] kostenlos heruntergeladen werden. **Bild 3** zeigt den Ablaufplan. Die Quarzfrequenz von 6 MHz wird durch 64 auf 93750 Hz geteilt (Einstellung mit Register TCCR1B) und vom Timer 1 als Zähltakt verwendet. Timer 1 wird im 10-bit-PWM-Modus (Einstellung mit Register TCCR1A) betrieben. Die Zähltaktimpulse werden im Register TCNT1 von 0 bis 1023 herauf- und dann wieder auf 0 herunter gezählt. Bei 93750 Hz und 2 x 1024 Zählschritten hat dieses Dreiecksignal eine Periodendauer von 21,84 ms. Servos sind in der Regel für Impulslängen von 20 ms ausgelegt, Versuche zeigten aber, dass Werte zwischen 15 ms und 35 ms problemlos verarbeitet wurden. Für exakte 20 ms wäre ein Quarz mit 6,5536 MHz erforderlich. Erreicht der fallende Wert im Register TCNT1 den Wert im Register OCR1A, wird der Ausgang OC1A (PB3) auf High gesetzt, bei steigendem Wert im Register TCNT1 wieder auf Low. An OC1A=PB3 steht ein Rechtecksignal mit 21,85 ms Periodendauer und einer Pulsbreite an, die vom Wert im Register OCR1A abhängt. Bei 23 beträgt die Pulsbreite etwa 0,5 ms, bei 129 circa 2,7 ms. Mit dem Signal an PB3 lässt sich direkt ein Modellbauservo ansteuern. Durch Verändern des Wertes im Register OCR1A kann der Servo verstellt werden. Die üblichen 180°-Drehwinkel werden in 106 Schritte (129–23) aufgelöst. Wird der Stepper verwendet, wird das Register OCR1A fest auf 25 gesetzt, um Pulse mit annähernd 50 Hz zu erhalten.

Das 21,84-ms-Signal von OC1A (PB3) gelangt (per Drahtbrücke) an den Eingang T0 (PD4) von Timer 0, der im CTC-Modus (Einstellung mit Register TCCR0A und TCCR0B) arbeitet. Dabei addiert er die Eingangspulse an T0 bis zum Referenzwert im Register OCR0A, toggelt den Ausgang OC0A (PB2), setzt sich auf 0 zurück und startet erneut. An PB2 wird ein symmetrisches Rechtecksignal mit einer Periodendauer von 1 s (genau 1,00464 s) ausgegeben, wenn der Referenzwert in OCR0A 23 beträgt. Mit dem Wert von 23 (statt 25) wird die Ansteuerung mit 21,84 ms statt 20 ms Periodendauer ausgeglichen.

In der Abfrage-Routine für den Reedkontakt (Zeile 89 bis 96) wird das Eingangssignal an PD6 abgefragt (klassisches Polling). In zwei kurzen Schleifen wird eine Kontaktentprellung vorgenommen. Ist somit der Eingangspuls sicher erkannt, werden die Zählvariablen *speed* und *way* inkrementiert. In der Auswerteroutine (Zeile 61 bis 85) wird zu Beginn der Low-Phase an PB2, also einmal pro Sekunde, der momentane Messwert in der Variablen *speed* in die Variable *speed_new* gespeichert. Der vorherige Messwert wird nach *speed_middle* und der vorvorherige nach *speed_old* umgeleitet. Damit stehen für die Auswertung drei Messwerte über 3 s zur Verfügung, deren Mittelwert verwendet wird. Rollierend werden diese Messwerte im Sekundentakt erneuert. In einer mathematischen Umrechnung wird aus diesem Mittelwert der Anzeigewert gebildet (Zeile 70). Welche Umrechnungsformel erforderlich ist, hängt von der konkreten Anwendung ab und ist hier als Beispiel für den vorgegebenen Geber aufgeführt: bei etwa 10 Hz = 2,5 Kn muss der Zeiger nahe Linksanschlag und bei 55 Hz = 10 Kn nahe Rechtsanschlag stehen. Bei Nachbauten wird sicher eine andere mathematische Formel notwendig werden. Der umgerechnete Anzeigewert muss wieder in der Variablen *speed* stehen (beim Servo 23...129; beim Schrittmotor 0...1260). Mit der Zählvariablen *way* werden die Geberimpulse unmittelbar gezählt. Bei Erreichen einer vorgegebenen Schwelle (hier 1820 für 0,1 Seemeile) wird an PB1 ein 10-ms-Impuls für das elektromagnetische Zählwerk ausgegeben und *way* wieder auf 0 gesetzt.

Stepper oder Servo?

Bei Verwendung eines Servos wird der Wert der Variablen *speed* einfach in das Register OCR1A geschoben, beim Stepper wird mit speed als Übergabevariable die Funktion MOTOR() aufgerufen. Danach wird *speed* auf 0 gesetzt, und die Zählerei beginnt von vorne. Mit der Hilfsvariablen *flag* wird verhindert, dass mehr als eine Auswertung pro Sekunde erfolgt (Messzyklus 1 pro Sekunde). Die Messwertverarbeitung kostet natürlich Zeit, die vom Messintervall abgeht. Um das in etwa auszugleichen, wird nur mit 1 Sekunde gerech-

net, wobei real 1,00464 s verwendet werden.
Die Ansteuerung des Steppers erfolgt über die Ausgänge PD0 bis PD3. Über die Funktion MOTOR() wird die Stepperbewegung gesteuert. In den globalen Variablen *actual_position* und *coilstep* sind die Ist-Position (0 bis 1260) und die letzte Phasenposition in der 8 Phasen umfassenden Ansteuerung des Steppers (0 bis 7) gespeichert. Die Sollposition wird mit der lokalen Variablen *new_position* übergeben. Im Array *coil[n]* sind die acht Ausgabewerte der Phasenansteuerung für den Port D abgelegt. Da PD4...6 Eingänge sind, stört die Zuweisung einer 0 nicht. Eine Maskierung für Port D ist daher verzichtbar. Die Drehrichtung wird mit der Variablen *direction* errechnet. Dann wird der Stepper so lange gedreht, bis *actual_position* und *new_position* übereinstimmen, In den Zeilen 114 bis 117 wird der Phasenumlauf des Steppers festgelegt. Die maximale Drehgeschwindigkeit (kleiner 5 ms von Step zu Step) macht eine Verzögerungsschleife (Zeile 120) erforderlich. Die Ist-Position wird mit *actual_position* und *coilstep* in globalen Variablen gespeichert und steht somit beim nächsten Funktionsaufruf wieder zur Verfügung.

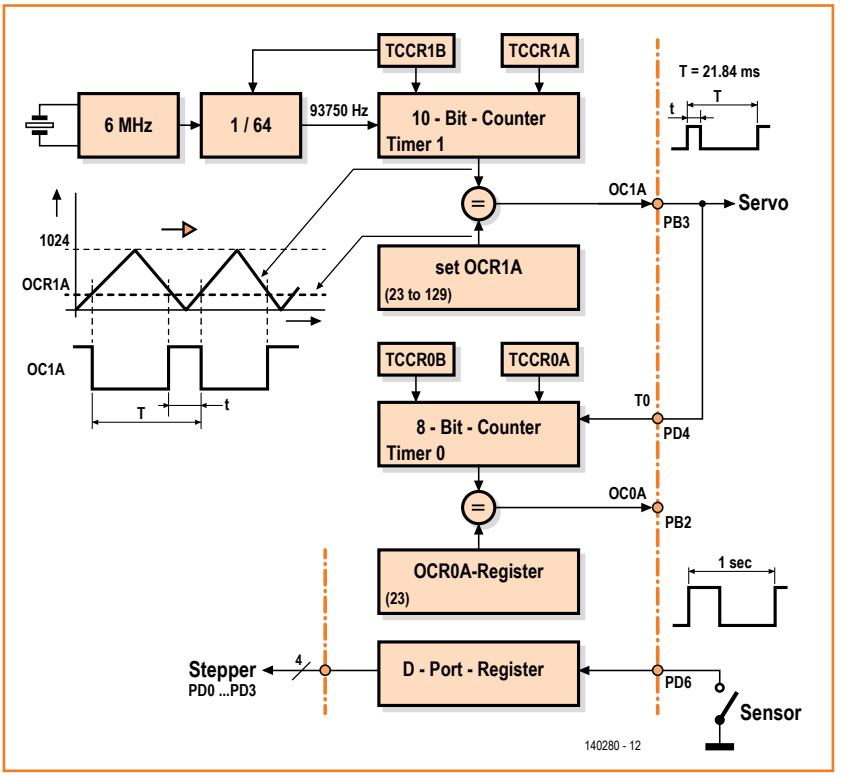


Bild 3.
Ablaufplan der Software.

Andere Aufgaben

Die Software ist in C++ mit AVR-Studio 4.12 geschrieben und relativ einfach gehalten, so dass sie leicht für andere Anwendungen modifiziert werden kann. So ist bereits eine Variante für den Ersatz des mechanischen Tachos an einem älteren Motorrad in Arbeit. Ein Dauermagnet an der Felge und ein Reedkontakt an der Vorderradgabel erzeugen drehzahlabhängige Rechteckimpulse, wie sie die Anzeigeeinheit benötigt.
In einem älteren Unimog wird der mechanische Drehzahlmesser ersetzt. Die Geberimpulse werden am Anschluss W der Lichtmaschine abgegriffen, nachdem sie per Z-Diode und Vorwiderstand in ein 5-V-Rechtecksignal gewandelt wurden. Verzögerungs- und Messzeiten sind selbstverständlich anzupassen. Elektrik und Programm stellen meist keine große Herausforderung dar, anders aber die jeweilige Mechanik. Es bedarf schon etwas handwerklichen Geschicks, um ein altes Instrument mit modernem Innenleben auch weiterhin „alt“ aussehen zu lassen.

(140280)

Weblinks

[1] Stepper SWITEC MS X25 u.a. bei eBay
[2] zum Beispiel bei: www.gemmer-zaehlerbau.de
[3] Software in C unter: www.elektor.magazine.de/140280

Glühlampenelektronik

Sinnvoller Einsatz von Glühlampen in Schaltungen

Von **Peter E. Tiefenthaler** (D)

Als Beleuchtungsmittel gehören Glühlampen zu den gefährdeten Arten, doch als „Sicherungen mit automatischer Strombegrenzung und integrierter Überlastungsanzeige“ könnten sie vielleicht noch einige Jahre unbehelligt vor eurokratischen Regulierungen weiterexistieren!

Glühlampen können mehr als nur leuchten. Als Strom- oder Leistungsbegrenzer haben sie gegenüber „normalen“ Sicherungen in etlichen Anwendungen Vorteile: Während Sicherungen bei einem bestimmten Strom ihren Widerstandswert abrupt von null auf unendlich ändern, begrenzen Glühlampen den Stromanstieg zunächst wenig und dann zunehmend stärker.

Sie zeigen sogar durch Aufleuchten an, dass der Strom zu groß „scheint“. Wie eine Sicherung können auch sie bei Überlast durchbrennen. Nicht zuletzt sind Glühlampen gut erhältlich, preiswert und einfach zu handhaben.

Mehr als nur Licht

Technisch gesehen verhalten sich Glühlampen mit Metallfaden wie Kaltleiter bzw. NTCs. Ihr Widerstand ist im kalten Zustand (bei niedrigen Spannungen) klein. Durch ausreichend Strom erwärmt sich die Glühwendel, und ihr Widerstand steigt an. Diesem Effekt verdanken sie z.B. den klassischen Einsatz zur Amplitudenstabilisierung in Wien-Generatoren.

Der Kaltwiderstand einer (nicht mehr) handelsüblichen 60-W-Glühbirne liegt bei etwa 70 Ω. Angeschlossen an 230 V und somit leuchtend fließen um die 260 mA, was einen Heißwiderstand von rund 900 Ω ergibt. Der Kaltwiderstand liegt also bei knapp weniger als einem Zehntel des Heißwiderstands – ein guter Näherungswert für Pi x Daumen-Berechnungen. In **Bild 1** ist der Widerstand als Funktion der anliegenden Spannung für eine 12-V-Autoglühbirne angegeben. Der genaue Zusammenhang hängt von der konkreten Bauart ab.

Antike Kohlefadenlampen funktionieren übrigens anders herum: Sie sind PTCs.

Strom- und Leistungsbegrenzung

Der Einsatz von Glühlampen als Sicherungen ist zwar keine Erfindung des dritten Jahrtausends, doch ist dieser Trick mittlerweile selbst bei erfahrenen Elektronikern in Vergessenheit geraten. So war zu Röhren-Radio-Zeiten ein Netzkabel mit eingefügter 60-W-Glühbirne als Schutzmaßnahme ein unersetzliches Reparatur-Hilfsmittel (**Bild 2**). Auch heute noch bewährt sich dies bei der ersten Überprüfung von Geräten unbekannten Zustands, um sie schonend hochzufahren und/oder die Elkos zu re-formieren.

Bei vielen US-Röhrenprüfgeräten ist ein Vorwiderstand plus Lämpchen im Netzkreis als „anzeigende Sicherung“ fast schon Standard (**Bild 3**). Auswirkungen schwankender Netzspannung können so einfach im Zaum gehalten werden. Ein weiteres Lämpchen dient als Sicherung bei der Vorspannungs-Erzeugung.

Eine ähnliche Funktion haben Glühlämpchen auch in manchen Röhrenverstärkern, wo sie als Vorwiderstände einen unzulässigen Anstieg des Schirmgitter-Stroms begrenzen, dabei aufleuchten und somit als Aussteuerungs-Anzeige dienen. In den meisten Röhrenverstärkern fehlt jede Absicherung der Anodenstrom-Versorgung. Als Sicherung und Strombegrenzung kann man hier auch Glühlampen nachrüsten. Auch Sicherungs-Widerstände, die bei einer bestimmten Temperaturbelastung unterbrechen, sollten wenn nötig besser durch eine Glühlampe als durch einen normalen Widerstand ersetzt werden.

Geeignet sind für geringen Spannungsabfall Taschenlampen-Birnen mit z.B. 2,5 V bei 150...200 mA oder für einen größeren Spannungsabfall auch Skalen- oder Kfz-Lämpchen. Es können auch mehrere Exemplare in Serie oder parallel geschaltet werden. Lämpchen können an Stelle des Widerstands oder aber zusätzlich an den Punkten A und/oder B in **Bild 4** eingefügt werden.

In einem Bleibatterie-Ladegerät kann eine Glühlampe in gewissen Grenzen den Ladestrom stabilisieren (**Bild 5**).

Selbstverständlich lassen sich zu hohe Versorgungsspannungen mit Hilfe von Glühlampen recht einfach reduzieren. Eine geeignete Lampe kann z.B. als Vorwiderstand bei der Speisung von 110-V-Geräten aus dem 230-V-Netz dienen.

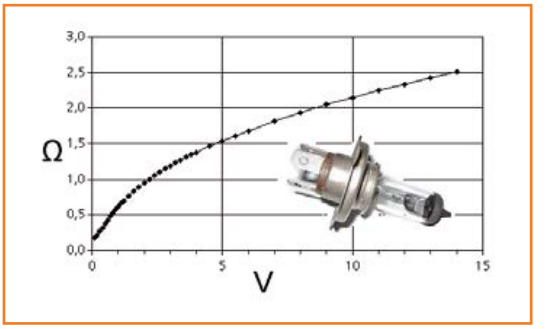


Bild 1.
Der Widerstand einer gewöhnlichen Halogen-Glühlampe für Autoscheinwerfer als Funktion der angelegten Spannung.

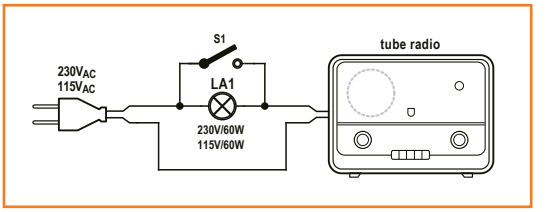


Bild 2.
Eine Glühlampe in der Netzleitung vermeidet Schäden beim Test alter Geräte.

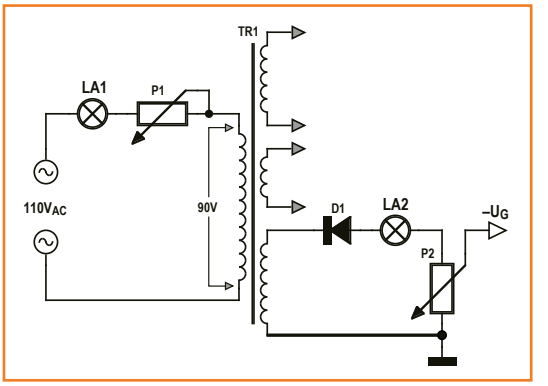


Bild 3.
Typischer Einsatz von Glühlampen in US-Röhrenprüfgeräten.

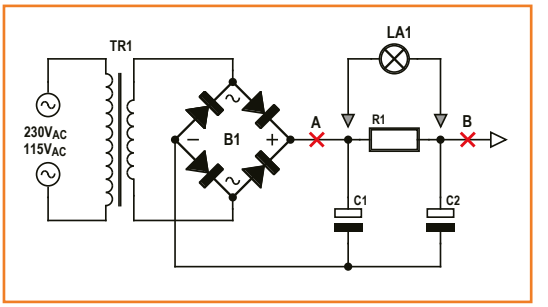


Bild 4.
Glühlampen zur Strombegrenzung in Verstärkern.

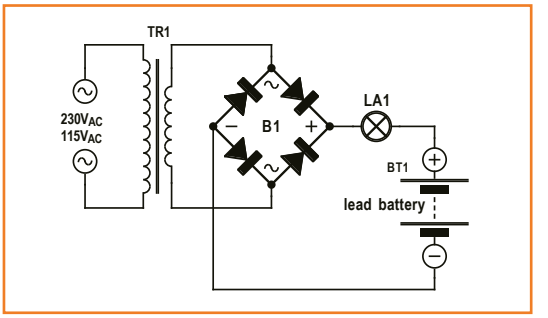


Bild 5.
Glühlampen zur Strombegrenzung beim Laden von Blei-Akkus.

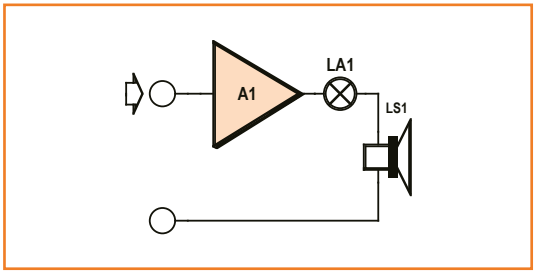


Bild 6.
Glühlampen zum Schutz von Lautsprechern.

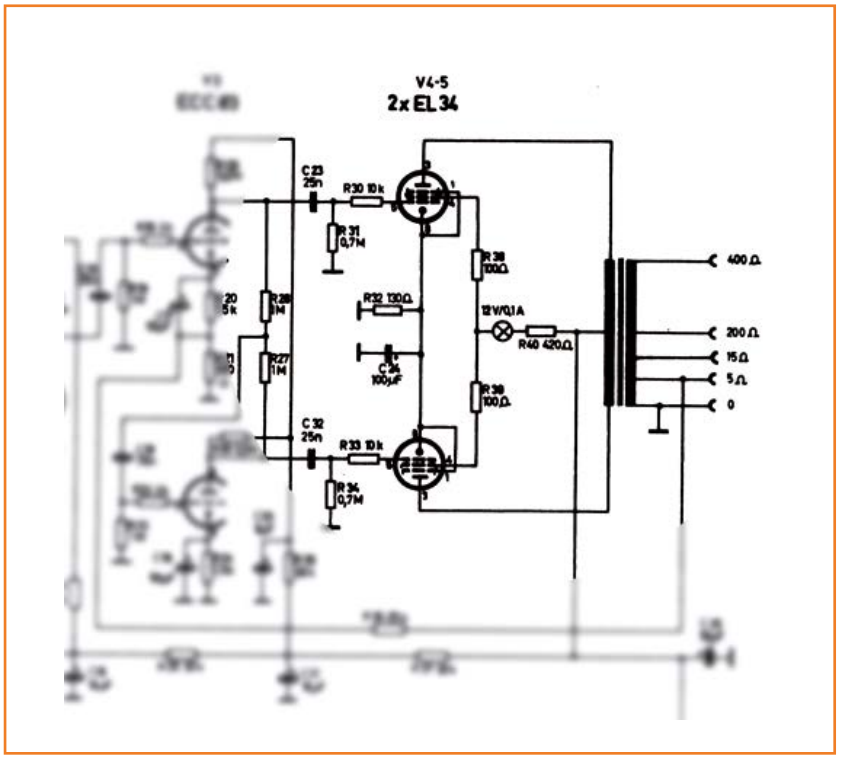


Bild 7.
Ausschnitt aus einem Originalschaltplan eines Röhrenverstärkers (RIM Gigant). Auch hier dient eine Glühlampe als Leistungsbegrenzer.

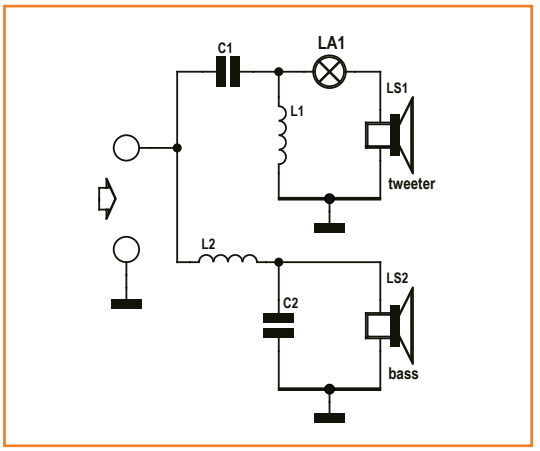


Bild 8.
Eine Glühlampe schützt den empfindlicheren Hochtöner.

Lämpchen & Lautsprecher

Für Lautsprecher können Glühlampen eine preiswerte und einfach installierbare Lebensversicherung darstellen. Eine Glühlampe am Verstärker-Ausgang (**Bild 6**) schützt Lautsprecher sicher vor Überlastung. Dies vermeidet gleichzeitig häusliche Unstimmigkeiten, wenn etwa der digital sozialisierte Nachwuchs beim Feiern die Lautstärke auf logisch „high“ = Rechtsanschlag dreht.

Der niedrige Kaltwiderstand der Glühlampe ist bei Zimmerlautstärke vernachlässigbar, da hier nur Leistungen im Milliwatt-Bereich auftreten. Bei höherer Ausgangsspannung nimmt das Lämpchen zunehmend mehr Leistung auf und begrenzt so den Lautstärkeanstieg. Zudem steigt der Lastwiderstand des Verstärkers, was zu einer weiteren Begrenzung der Ausgangsleistung führt. Diese ist bei höheren Leistungen stärker ausgeprägt, was zu einer gewissen Kompression und Begrenzung des Signals bei höheren Pegeln führt.

Die Lämpchen sind in der Signalkette nicht hörbar. Erst bei hohen Pegeln machen sie sich bemerkbar. Das ist auch gut so, denn so zeigt sich, dass der Lautsprecher-Schutz anspricht. Klangliche Einbußen sind preiswerter als die Anschaffung neuer Lautsprechersysteme.

Die historische Schaltung von **Bild 7** zeigt, dass auch schon zu Röhrenverstärker-Zeiten Glühlampen (sogar serienmäßig) als Leistungsbegrenzer eingesetzt wurden.

In PA-Anlagen haben sich Lämpchen als Sicherungen für die empfindlicheren Hochtöner seit langem bewährt (**Bild 8**). Auch Werkstatt- und Test-Lautsprecher sind auf diese Weise gut geschützt. Für die Auswahl der richtigen Glühlampe eignet sich die oben angeführte Faustregel ($Z_{heiß}$:Z_{kalt} = 10:1). Eine 12-V-Kfz-Soffitte mit 5 W ist ein zuverlässiger und brauchbarer Zimmerlautstärke-Begrenzer für unterschiedlichste Lautsprecher. Für eine HiFi-Anlage ist eine 6-V-Lampe mit 5 W womöglich besser geeignet.

Elektrogitarren und ihre Verstärker beginnen meist erst bei etwas höheren Lautstärken zu „singen“. Diese Darbietungen finden nicht immer den Beifall der Nachbarschaft. Hier kann ein sogenannter „Power Soak“ bzw. Leistungsminderer helfen: Eine Schaltung aus Lastwiderständen ergänzt durch L- und C-Glieder, welche die Impedanz eines Lautsprechers nachbilden und den Großteil der Verstärkerleistung „verbraten“. Für den Lautsprecher bleibt so nur noch ein Teil der

Leistung übrig, womit dieser „laut“ klingt, aber leise ist. Ersetzt man die Leistungswiderstände des Power Soaks durch Glühlampen entsprechender Dimensionierung (LA2 in **Bild 8**), wird der Klang durch den oben erwähnten Kompressions-Effekt noch voller und realistischer.

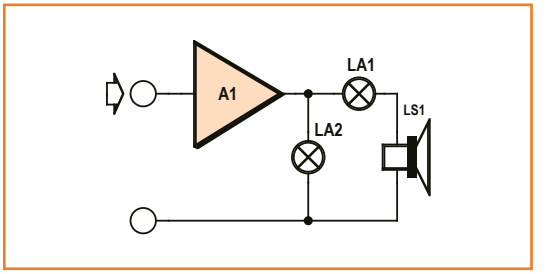


Bild 9.
LA2 als Power Soak bei Gitarrenverstärkern.

Röhrifizierung

Glühlampen und Elektronenröhren sind ja beide innen hohl und leer. Das ist zwar kein hinreichender Grund, aber trotzdem eignen sich Glühlampen vorzüglich, um Transistor-Verstärker mit geringem Aufwand wie Röhrenverstärker klingen zu lassen. Zunächst einmal kann man wie in Bild 4 im Netzteil eine Glühlampe einsetzen, um die weichere Versorgungsspannung einer Röhrengleichrichtung zu simulieren. Übrigens kann ein Glühbirnchen bei Röhrenverstärkern beim Ersatz einer Gleichrichterröhre durch Halbleiter-Gleichrichter den Spannungsabfall kompensieren.
Zur Röhrifizierung eines Transistorverstärkers kommt eine Lampe zwischen Verstärkerausgang und Lautsprecher. Noch ausgeprägter wird der Röhren-Effekt, wenn man die Emitter-Widerstände einer konventionellen Transistor-Endstufe durch kleine Glühlämpchen ersetzt (**Bild 10**). Der Verstärker begrenzt dann weicher und klingt (fast) wie ein Röhrengerät.

(140188)

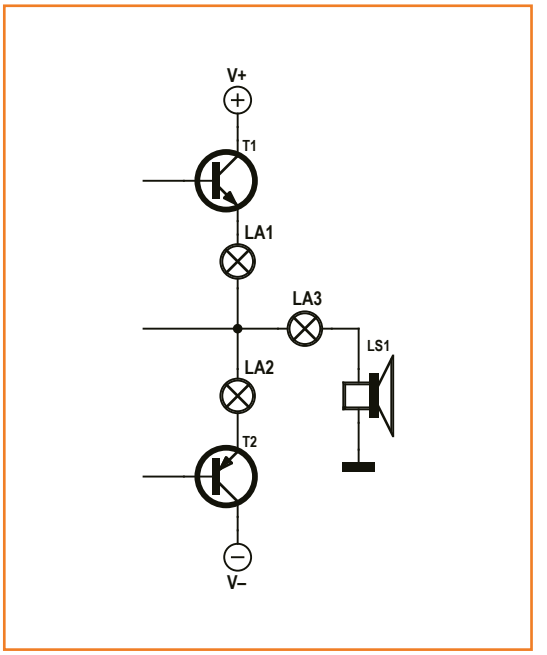
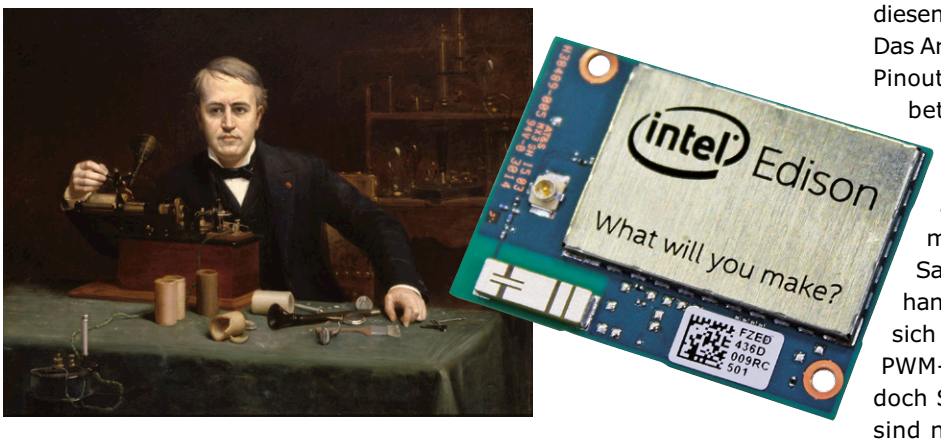


Bild 10.
Röhrifizierung eines Transistorverstärkers.

Intel Edison: What will you make?



Von **Clemens Valens**
(Elektor.Labs)

Ende September 2014, weniger als ein Jahr nach der Einführung des Galileo-Boards (jetzt schon in der Version 2) stellte Intel sein winziges Edison-Modul vor. Vermarktet wurde es als „low-cost, product-ready, general purpose" Computing-Plattform, geeignet für Entwickler jeglicher Couleur, vom ambitionierten Amateur (wie Steve & Steve in ihrer Garage in den späten 70ern) über Ingenieure der Consumer-Elektronik bis zu Erfindern in der Welt des Internet of Things und der Elektronikbekleidung. Ein Blick auf die technischen Daten (siehe Kasten) zeigt die beeindruckenden Möglichkeiten des Edison-Moduls. Es sieht aus wie das Innere eines Smartphones und wahrscheinlich ist es auch nicht allzu schwer, damit eins zu bauen.

Yocto-Linux

Edison wird mit vorinstalliertem Yocto-Linux ausgeliefert, das sofort aktualisiert werden sollte. Yocto soll die Erstellung von benutzerdefinierten Linux-Distributionen für Embedded-Produkte vereinfachen. Für den typischen Edison-Benutzer ist dies nicht von Bedeutung, obwohl es große Flexibilität für den fortgeschrittenen Benutzer bietet, der das OS anpassen will.

Auch Edison nutzt Arduino

Ohne Breakout-Board (BoB) kann man mit Edison nicht viel anfangen, daher bietet Intel das Edison-Kit für Arduino und das Edison-Breakout-Board an (die Spezifikationen sind unten zu sehen). In diesem Artikel verwende ich das Kit für Arduino. Das Arduino-BoB ist zu 99 % kompatibel mit dem Pinout des Arduino 1.0. Das fehlende Prozent betrifft zwei PWM-Kanäle, es gibt nämlich nur vier statt sechs. Als Ersatz ist das BoB mit dem so genannten *PWM swizzler* ausgestattet (ein Swizzler ist ein Hotdog mit spiralförmigem Einschnitt, der das Saucenaufnahmevermögen verdreifacht!). Es handelt sich um einen Jumper-Block, mit dem sich die PWM-Signale auf die sechs Arduino-PWM-Pins verteilen lassen. Eine feine Sache, doch Shields mit mehr als vier PWM-Signalen sind nicht möglich. Aber Sie können dann ja PWM-Signale per Software erzeugen.

Der Swizzler hat einen weiteren Haken: Die PWM-Ausgänge des Edison beziehen sich jeweils auf einen bestimmten Arduino-Pin. Leiten Sie solch einen Ausgang zu einem anderen Arduino-Pin, ist der vorgesehene Pin nicht mehr frei. Benutzen Sie also den Swizzler wirklich nur, wenn Sie müssen!
Es gibt noch ein paar andere Feinheiten, die Sie in Schwierigkeiten bringen könnten. Ich empfehle, den „Intel Edison Kit für Arduino Hardware Guide" von Anfang bis Ende sehr sorgfältig zu studieren.

Am Anfang

Es beginnt mit einigen großen Downloads von der Intel-Website. Sie benötigen die Arduino-IDE für Edison (und Galileo), den USB-Seriell-Port-Treiber von FTDI und einige weitere Treiber für das Edison-Board. Sie sollten die neueste Ausgabe von Yocto herunterladen. Mit Ausnahme der FTDI-Treiber sind alle diese Dateien auf der Edison-Download-Seite zu finden. Während Sie auf den Download warten, können Sie zwei Micro-USB-Kabel hervorkramen (oder ein Micro-USB-Kabel und ein 12-V-Steckernetzteil). Sie sollten auch Ihr Kit zusammenbauen. Im Gegensatz zu Intel glaube ich, dass Sie dazu klug genug sind. Falls Sie auf Probleme stoßen, holen Sie sich Rat auf der Intel-Website [1].
Und dann:

1. Installieren Sie alle Treiber auf Ihrem Computer und achten Sie darauf, dies mit Administratorrechten zu tun.
2. Laden Sie das neueste Yocto-Image für das Edison-Modul herunter. Dazu benötigen Sie ein Micro-USB-Kabel am Micro-USB-Anschluss J16, der dem Schalter am nächsten ist. Stecken Sie das Kabel am PC ein, so wird das Modul als Lokaler Datenträger angezeigt. Überprüfen Sie, ob es leer ist und löschen gegebenenfalls vorhandene Dateien. Dann entpacken Sie das Yocto-Image auf den Lokalen Datenträger, den externen Speicher des Edison.
3. Flashen Sie anschließend die Software vom externen Speicher des Edison in den Edison selbst. Verbinden Sie das Micro-USB-Kabel mit dem anderen Micro-USB-Anschluss (J3) in der Ecke des Boards. Eine serielle Schnittstelle wird angezeigt. Ermitteln Sie die Nummer im Geräte-Manager (Windows, detaillierte Verfahren für Linux und Mac finden sich auf der Intel-Website [1]) und öffnen Sie diesen Anschluss mit einem Terminal-Programm wie Tera Term oder PuTTY und mit den Porteinstellungen 115200 Baud, 8N1. Drücken Sie die Eingabetaste zweimal, um den Login-Prompt („Edison login:“) zu erhalten. Geben Sie „root“ ein gefolgt von Enter, dann „reboot OTA“ und wieder Enter. Warten Sie, bis der Login-Prompt (nach etwa 2 min) erscheint. Fertig!
4. Entpacken Sie die Arduino-IDE und starten Sie arduino.exe.
5. Schließen Sie das Micro-USB-Kabel wieder an J16 an. Neben dem Laufwerk wird nun ein weiterer serieller Port kreiert: „Intel Edison Virtual Com Port“, bei mir COM33. Der Port wird für die Edison-Arduino-Programmierung genutzt.
6. Gehen Sie in der Arduino-IDE auf Tools, Board und wählen das Edison-Modul.
7. Gehen Sie in der Arduino-IDE auf Tools, Serial Port und wählen Sie den Anschluss, der dem virtuellen COM-Port des Edison entspricht (COM33 in meinem Fall).
8. Klicken Sie in der Arduino-IDE auf Files➡Examples➡01.Basics➔Blink, um das einfachste Testprogramm zu laden.
9. Dann klicken Sie zum Abschluss in der Arduino-IDE auf den Upload-Knopf, lehnen sich zurück und genießen das Blinken der LED in der Nähe von J4 in der Mitte des Boards.

Eigenschaften Edison-Modul

- 22 nm Intel SoC mit einer „dual core, dual threaded Intel Atom CPU“ mit 500 MHz und einem „32-bit Intel Quark microcontroller“ mit 100 MHz
- 40 General-purpose-I/O-Ports (GPIO), konfigurierbar zum Beispiel als SD-Karteninterface, 2x UART, 2x I²C, SPI, I²S, 4x PWM, USB 2.0 OTG
- 1 GB RAM (LPDDR3)
- 4 GB Flashspeicher (eMMC)
- Dual-Band WLAN
- Bluetooth 4.0
- Stromversorgung: 3,3...4,5 V
- Leistungsaufnahme im Standby-Modus: ohne Funk: 13 mW; Bluetooth 4.0: 21,5 mW; WLAN: 35 mW
- 35,5 × 25,0 × 3,9 mm
- 0...40°C
- Yocto-Linux v1.6

Eigenschaften Edison-Board für Arduino

- Unterstützt Arduino-Sketchs, Linux, WLAN und Bluetooth
- Board-I/O-kompatibel mit Arduino Uno (4x PWM statt 6x PWM)
- 20 digitale I/O-Pins, mit 4 (nicht 6) Pins als PWM-Ausgänge
- 6 analoge Eingänge
- UART (Rx/Tx)
- I²C
- ICSP 6-polige Stiftleiste (SPI)
- Micro-USB-Geräteverbinder ODER (über mechanischen Schalter) zugehöriger USB-Host-Verbinder Typ A
- Micro-USB-Geräteverbinder (verbunden mit UART)
- SD-Kartenverbinder
- DC-Niederspannungsbuchse (7...15 VDC Eingang)

Eigenschaften Edison Breakout Board

- Etwas größer als das Edison-Modul
- Stellt die originalen I/O (1,8 V) des Edison-Moduls zur Verfügung
- I/O-Array von Durchsteck-Lötpunkten im 0,1-Inch-Raster
- USB OTG mit Micro-USB-Verbinder Typ AB
- USB OTG Schalter für Stromversorgung
- Batterielader
- „USB to device – UART“-Brücke mit Micro-USB-Verbinder Typ B
- DC-Niederspannungsbuchse (7...15 VDC Eingang)

Zugabe: WLAN

Wenn Sie so weit gekommen sind, sind Sie bereit zum Programmieren des sehr komplexen und fast kompatiblen Arduino-Boards. Wir nehmen uns hier gleich vor, den Edison mit dem WLAN zu verbinden. So funktioniert es:

1. Schließen Sie das Micro-USB-Kabel an J3

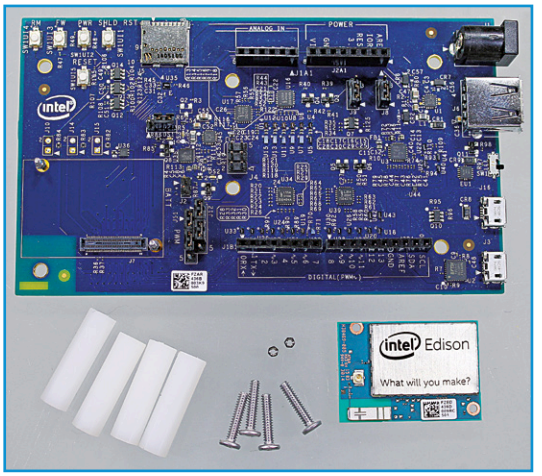


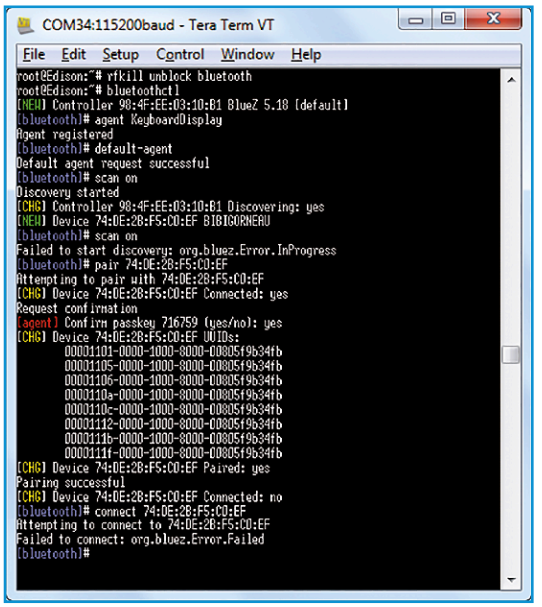
Bild 1.
Das Edison-Kit für Arduino ausgepackt.

in der Ecke der Platine an und öffnen Sie die serielle Schnittstelle aus dem vorherigen Schritt 3 mit einem Terminal-Programm wie Tera Term oder PuTTY (Porteinstellungen wie oben). Drücken Sie die Eingabetaste zweimal, um den Login-Prompt („Edison login:“) zu erhalten. Geben Sie „root“ ein gefolgt von Enter.

2. Geben Sie „configure_edison --setup“ (doppelter Bindestrich!) ein und drücken Sie Enter, so dass ein Assistent startet, der Sie durch die Konfiguration leitet. Wenn Sie darin einen Schritt nicht ausführen möchten, überspringen Sie ihn einfach mit Enter.

3. Wenn Sie alle Schritte durchlaufen haben, erhalten Sie eine Meldung, dass Sie nun mit dem Modul Verbindung aufnehmen können, indem Sie die IP-Adresse xyz in einem Browser angeben. Ist dies geschehen und es erscheint eine Begrüßungsseite, dann seien Sie glücklich! Ich war es nicht, denn das Modul war in meinem Router zwar aufgeführt, aber ohne Namen.

Bild 2.
Aktivierung von Bluetooth auf dem Edison. Alles geht perfekt, doch mit meinem PC gab es leider keine Verbindung...



Und nun?

Laut meinem Router hatte mein Edison-Modul keinen Hostnamen. Der Befehl „hostname –fqd“ im Terminal ergab den Fehler „Host name lookup failure“. Der Versuch, Google zu „pingen“, führte zu „bad address ‘www.google.com’“. Eine Internet-Recherche förderte Menschen zu Tage, die ähnliche Probleme mit Raspberry Pi hatten, aber keine der vorgeschlagenen Lösungen hat geholfen, weil sie entweder scheinbar gar nichts bewirkten oder die Befehle von Yocto-Linux nicht erkannt wurden. Wenn jemand weiß, wie man das Problem lösen könnte, bitte lassen Sie es mich wissen. In der Zwischenzeit werde ich weiterhin Linux hassen.

Die glücklichen Benutzer, die mit dem Browser das Edison-Modul finden können, dürfen die Arduino-WLAN-Beispiele ausprobieren. Sie sollten prima funktionieren, solange man die Strings „ssid“ und „pass“ fehlerfrei in den Sketches eingibt. Der Edison verhält sich dann wie ein Arduino mit WLAN-Shield.

Ich konnte keine Bluetooth-Beispiele für Arduino finden. Das Tutorial, das ich auf der Intel-Website vorfand, zeigt, wie die Aktivierung von Bluetooth funktionieren sollte, aber es gibt keinen Anhaltspunkt, wie man Arduino-Code und die Edison-Bluetooth-Funktion kombinieren kann. So ist dieses Feature im Moment den Linux-Programmierern vorbehalten.

Fazit

Das Edison-Modul von Intel ist großartig und bietet eine Unmenge von Möglichkeiten. Aber wie ist seine Macht zu entfesseln? Ohne richtig funktionierendes WLAN gibt es nicht viel, was Sie damit tun können. Und ohne BoB sowieso nicht. Fehlende Unterlagen und nicht vorhandene einfach zu verstehende Codebeispiele, das ist nicht das, was Einsteigern hilft. Edison ist wahrscheinlich ideal für Menschen, in deren Adern Linux statt Blut fließt und die alle möglichen Kombinationen von Boards zu wahnsinnig interessanten Applikationen verbinden können. Für Menschen, die unerschrocken die Standard-Yocto-Distribution bearbeiten und Pakete, die sie brauchen, problemlos integrieren. Für Menschen, deren höchstes Glück es ist, mit einem Texteditor wie vi zu arbeiten.

Und dann ist da noch ein Detail, das mich nörgeln lässt: Edison ist nur bis 40°C spezifiziert. Das ist nicht viel. Ich habe bei mir eine Hosentaschentemperatur von 33°C ermittelt, und das bei einer Umgebungstemperatur von 21°C. Also bleiben Sie und Ihre Kleidung vor allem eines: cool!

(140439)

Weblink

[1] Intel Edison: www.intel.com/content/www/de/de/do-it-yourself/edison.html

Professionelle Hard- & Software zum Sonderpreis!

Exklusiv für Studenten!

Als neuer Vertriebspartner von National Instruments bietet Elektor ab sofort die Produkte der NI-Plattform für Ausbildung und Lehre für Studenten und schulische Einrichtungen an. Diese edukative Plattform vereint Hardware, Software und Unterrichtsmaterial, um Schülern und Studenten ein attraktives und inspirierendes Lernumfeld zu ermöglichen.

LabVIEW

Mit der Systemdesignsoftware *LabVIEW* können Studenten praxisorientiert anhand von Projekten und Systemen in einer einzigen Umgebung lernen und sich so Fähigkeiten und Verfahrensweisen aneignen, die im späteren Berufsleben unschätzbar sind.

NATIONAL INSTRUMENTS LabVIEW

Circuit Design Suite

Die *Circuit Design Suite* umfasst *Multisim* und *Ultiboard* und ist eine vollständige Plattform für Entwurf, Simulation und Validierung von Schaltplänen sowie den Leiterplattenentwurf. Die Suite verfügt über Funktionen, die speziell auf die Anforderungen von Studenten zugeschnitten sind, die bei der Entwicklung von elektronischen Konzepten hilfreich sind.

myDAQ

Bei *myDAQ* handelt es sich um ein kostengünstiges Datenerfassungsgerät, das überall und jederzeit Messungen und Analysen physikalischer Signale ermöglicht. *myDAQ* ist kompakt und portabel, sodass Studenten auch außerhalb des Labors und unter Einsatz branchenüblicher Werkzeuge und Methoden praktische Erfahrungen sammeln können.

Studentenversionen:

- Der Preis von ***LabVIEW*** und ***Circuit Design Suite*** beträgt jeweils nur 23,95 Euro.
- Das ***myDAQ Education-Kit*** bestehend aus *myDAQ* + 3 Software-DVDs (*LabVIEW, Circuit Design Suite* und *DIAdem*) kostet nur 174,95 Euro.

Jetzt bestellen unter www.elektor.de/ni-plattform!

IIR-Sinus-Generator

Kompakter Code für kleine Controller

Von
Dr.-Ing. Rüdiger Knörig
(D)

Sinusförmige Signale kann man digital ziemlich einfach erzeugen, ohne dass man dafür auf Tabellenwerte und aufwendige Interpolationen zurückgreifen müsste. Man muss lediglich einen digitalen Filter rückkoppeln und schon „schwingt“ der digitale Generator wie er soll. Verwendet man IIR-Filter, dann hält sich auch der mathematische Aufwand in Grenzen und der nötige Code passt prima selbst in einen kleinen Mikrocontroller.

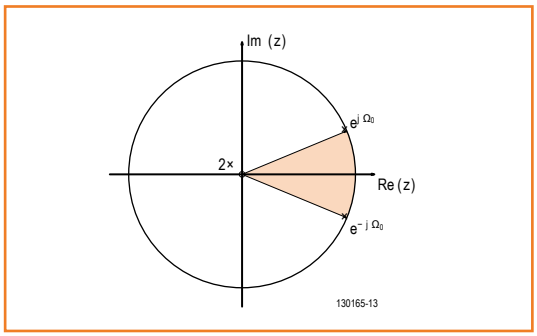


Bild 1.
Pol/Nullstellenplan des sinuserzeugenden Filters.

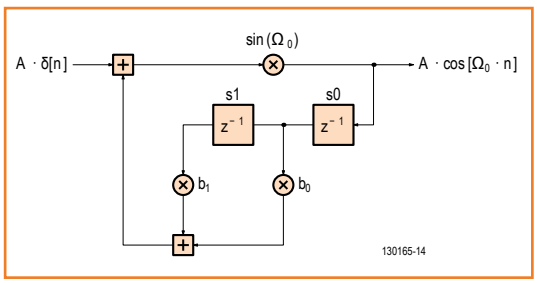


Bild 2.
Sinuserzeugendes IIR-Filter 2. Ordnung.

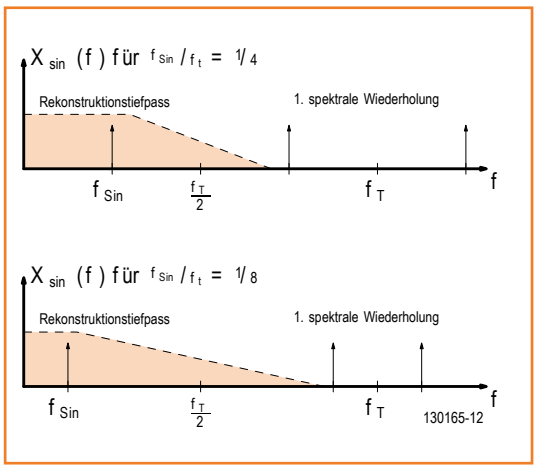


Bild 3.
Anforderungen an den Rekonstruktionstiefpass in Abhängigkeit von $\Omega_0 = 2\pi \cdot f_{sin} / f_T$.

Der übliche Weg zur Erzeugung von Sinussignalen mit Mikrocontrollern ist die Verwendung einer Wertetabelle. Mit diesem DDS (**D**irekte **D**igitale **S**ynthese) genannten Verfahren kann man recht einfach per Interpolation Sinussignale mit verschiedenen Frequenzen ableiten. Nachteilig ist der hohe Speicherbedarf für die Tabelle, der die Verwendung kleinerer Mikrocontroller mit weniger als 1024 Byte an Daten/Programmspeicher verhindert oder aber die Nutzung langsamerer Speicherarten erzwingt.
Das hier vorgestellte Verfahren kommt ohne Tabelle aus und benötigt im einfachsten Fall nur zwei Register als Zustandsspeicher. Da die Sinusfrequenz hier nur von einem Koeffizienten abhängt, lässt sich bei Verfügbarkeit einer ausreichend genauen (Fließkomma-)Multiplikation die Frequenz sogar zur Laufzeit ändern.
Das Prinzip wurde z.B. in [1] beschrieben - allerdings auf der Basis von Signalprozessoren. Das Verfahren lässt sich aber auch an die Möglichkeiten kleiner Mikrocontroller anpassen. Nachfolgend wird zunächst die Theorie erläutert. Für tiefere Einblicke kann man auch die klassische Literatur zu diesem Thema [2][3] zu Rate ziehen.

Theorie und Umsetzung

Die mathematischen Grundlagen sind mit vielen Formeln verbunden. Wer auf die genaue Herleitung Wert legt, der sei auf die detaillierte Beschreibung in der Datei „IIRsinus.pdf“ von der Elektor-Webseite zu diesem Artikel [4] verwiesen. Hier die Kurzfassung:
Ein kontinuierliches Sinussignal wird in ein (zeit)-diskretes Signal umgewandelt. Es wird dann ein digitales Filter gesucht, welches dieses diskrete Signal als Impulsantwort hat. Über mehrere

Rechenschritte erhält man ein Filter mit nur einer Nullstelle und zwei Polstellen. Eine zusätzliche Nullstelle im Ursprung macht das Filter kausal. Aus der resultierenden Gleichung wird der in **Bild 1** dargestellte Pol/Nullstellenplan abgeleitet. Diese Lösung erfordert eine hohe numerische Genauigkeit.

Resultat ist ein reines IIR-Filter 2. Ordnung (siehe **Bild 2**), dessen Filterkonstanten direkt aus der normalisierten Form dieser Gleichung zu $b_0 = 2cos(\Omega_{Sin})$ und $b_1 = (-1)$ bestimmt werden können.

Die Impulsantwort dieses Filters lässt sich berechnen und unter Vernachlässigung der Dämpfung ergibt sich folgender Pseudocode:

```
Require:
  z0 ← α
  z1 ← 0
  loop
     y  ← z0 · b0 – z1
     z1 ← z0
     z0 ← y
  end loop
```

Für die Berechnung braucht es nur zwei Register, eine Multiplikation, eine Subtraktion und zwei Register-Kopieroperationen. Für einfache 8-bit-Mikrocontroller sind weitere Überlegungen erforderlich. Schnelle Lösungen ergeben sich entweder durch Umsetzung der Multiplikation mittels Shift-Operationen oder durch Verwendung einer Tabelle mit 256 Einträgen, bei der man Speicherplatz sparen kann, wenn die Periodendauer des Sinus größer als 256 ist. In C++ umgesetzt ergibt sich folgender Sourcecode:

```
1  int8_t y,z0 ,z1;
2  z0 = alpha ; z1 = 0;
3
4  for ( int n=0;n< Nvalues ;n ++)
5  {
6        y = (z0 << 1);
7        y -= (z0 >> Nshift );
8        y -= z1;
9        z1 = z0;
10       z0 = y;
11       result << ( int ) y << endl;
12 }
```

Um brauchbare Kombinationen von Initialwert α und N_{Shift} sowie die Eigenschaften des so generier-

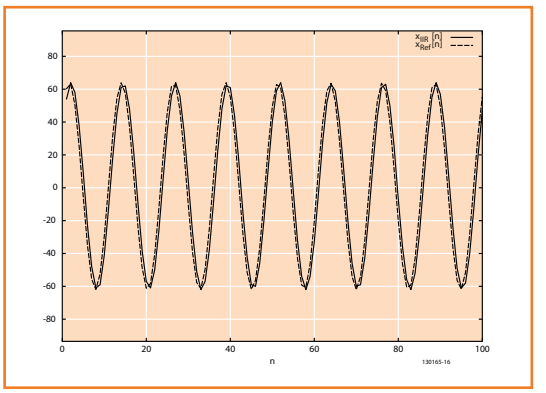


Bild 4.
N_{Shift} = 2, α = 61,
erwartet: 12,43,
gemessen: 12,42.
f_{sin}/f_T

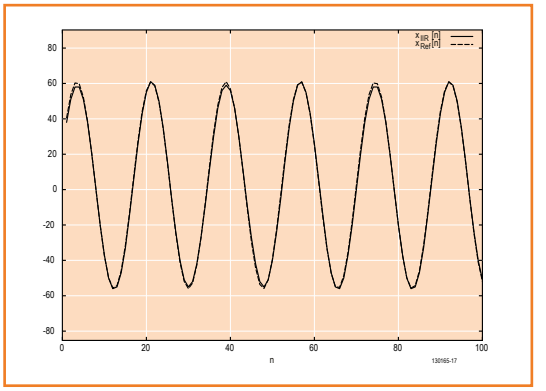


Bild 5.
N_{Shift} = 3, α = 40,
erwartet: 17,68,
gemessen: 17,75.
f_{sin}/f_T

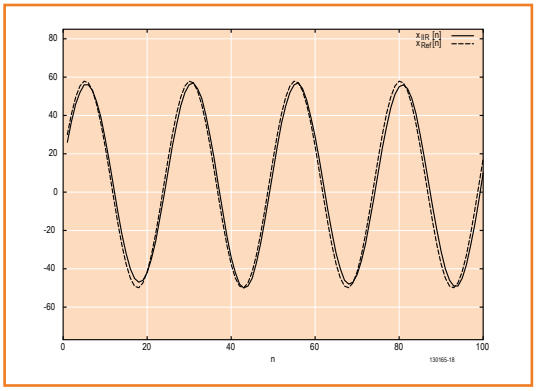


Bild 6.
N_{Shift} = 4, α = 26,
erwartet: 25,07,
gemessen: 25.
f_{sin}/f_T

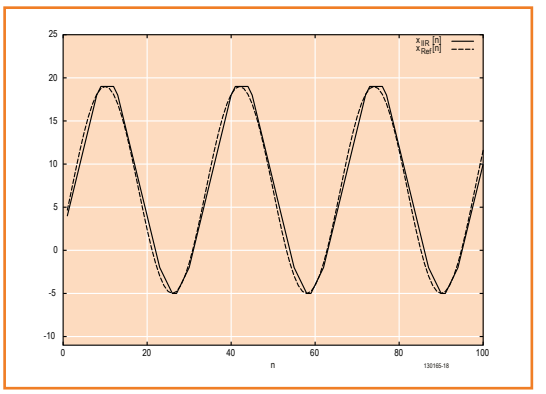


Bild 7.
N_{Shift} = 5, α = 4,
erwartet: 35,05,
gemessen: 32.
f_{sin}/f_T

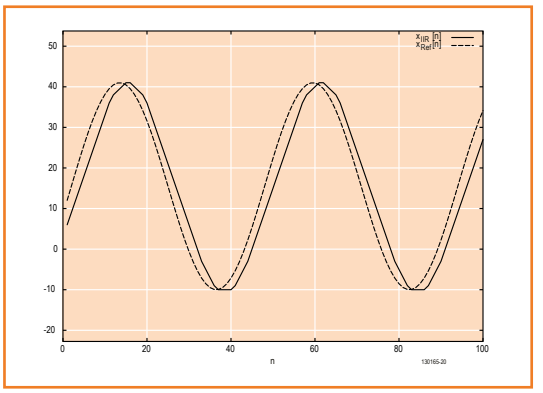


Bild 8.
N_{Shift} = 6, α = 6,
erwartet: 50,23,
gemessen: 46.
f_{sin}/f_T

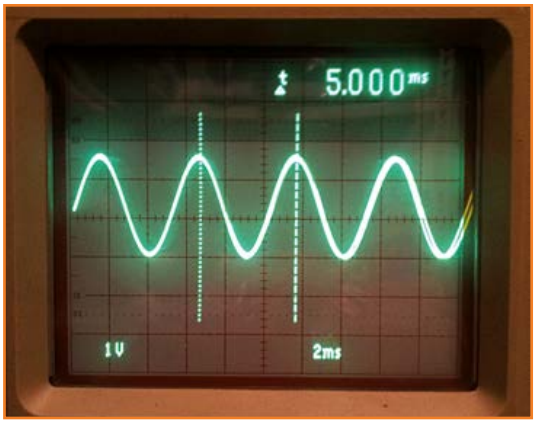


Bild 9.
Oszillogramm der Sinuskurve.

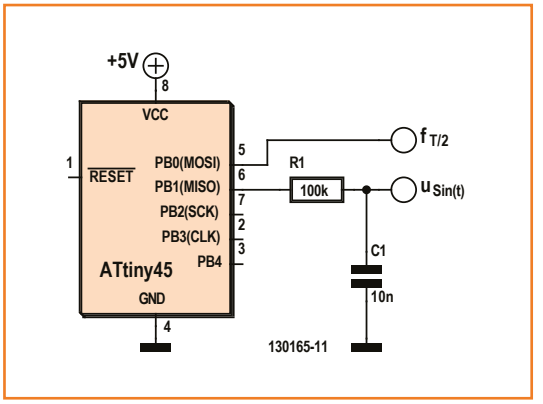


Bild 10.
Schaltplan des IIR-Sinusgenerators mit ATtiny45.

Tabelle 1. Numerische Analyse des generierten Sinus.

N_{Shift}	Initialwert α	f_{sin}/f_T erw.	η_{Sinus}	A	f_{sin}/f_T gem.	μ	Δ
2	61	1/12,43	0,98	63,00	1/12,42	1,00	10
3	40	1/17,68	1,00	58,50	1/17,75	2,50	2
4	26	1/25,07	0,99	54,00	1/25,00	4,00	2
5	4	1/35,50	0,99	12,00	1/32,00	7,00	1
6	6	1/50,23	0,98	25,50	1/46,00	15,50	1

ten Signals herauszufinden, wurden mit diesem Code *102.400* Ausgangswerte berechnet und mit einem Octave-Skript (ein freies, MatLab-ähnliches Programm) untersucht.
Die Ergebnisse mit den größten erzielten Amplituden sind in der **Tabelle 1** zusammengefasst. Die zugehörigen Signale sind in den **Bildern 4** bis **8** dargestellt.

Implementierung mit 8-bit-Mikrocontroller

Abschließend werden diese Vorüberlegungen mit einem ATtiny45-Mikrocontroller für den Fall N_{Shift} = *4* und α = *26* (siehe Tabelle 1) real umgesetzt. Der Controller wurde einfach mit dem internen Takt (8 MHz mit aktiviertem CKDIV8-Fuse = real 1 MHz) betrieben.
Timer0 des ATtiny45 wurde im CTC-Modus (siehe den Source-Code unter [4]) dazu verwendet, mit der Frequenz $f_T = f_{CPU}/200$ einen neuen Sinuswert zu generieren, welcher mit der schnellen 64-MHz-PWM von Timer1 am Ausgang OC1A (PB1) ausgegeben wird. Zur Kontrolle der Abtastfrequenz wird zusätzlich bei jeder Ausgabe eines Sinuswerts der Anschluss PB0 getoggelt.
Der an PB0 gemessene Flankenabstand lag bei 200 µs. Folglich gilt f_T = *5 kHz*. Das Oszillogramm von **Bild 9** bestätigt die Frequenz des generierten Sinus. Laut Tabelle 1 ist $f_{Sin} = f_T/25$ = 200 Hz bzw. beträgt die Periodendauer von TP = 5 ms.

Zur Unterdrückung von unerwünschten spektralen Wiederholungen und des PWM-Trägersignals genügte ein improvisierter Tiefpass mit der Grenzfrequenz f_g = *1 kHz* (R1 = 100 kΩ, C1 = 10 nF, siehe **Bild 10**).

(130165)

Literatur & Weblink

[1] N. S. Jayant, Peter Noll. *„Digital Coding of Waveforms“.* Prentice Hall, 1984.

[2] G. Cardamone, L. Lo Presti. *„A direct digital frequency synthesizer using an iir filter implemented with a dsp microprocessor“.* Band 6, S. 201–204. IEEE International Conference on Acoustics, Speech, and Signal Processing, 1994.

[3] Thomas Sikora Peter Noll. *„Signale und Systeme“.* TU Berlin.

[4] www.elektor-magazine.de/130165

Learn to program

Spielerisch programmieren lernen – ohne Tastatur

NEU!

Learn to program ist ein einfaches Lernspiel für Kinder und Schüler, das im Stil der Anfangsjahre des Computerzeitalters das Programmieren mit nur vier Tasten erlaubt. Auf dem Spielbrett befinden sich alle nötigen Ein-/Ausgabeeinheiten wie etwa eine LED-Ampel, ein Helligkeitssensor und ein Piezopiepser. Damit lassen sich verschiedene Programme entwickeln, wie zum Beispiel eine einfache Eieruhr, ein Wecker (der morgens wie ein Hahn „Kikeriki" ruft) oder auch einfache Reaktionsspiele. Die Batterieversorgung macht die Programmierung netzunabhängig.

Für viele Elektroniker ist es selbstverständlich, Programme zu schreiben, einfache Regelungen zu implementieren oder Steuerungen zu realisieren. Nur ist einem dabei oft nicht klar, was für ein langer Weg es war, um soweit zu kommen. Mit diesem edukativen Spiel kann man das Mysterium des Programmierens Kindern anschaulich erklären.

Eigenschaften des Boards:

• 16 verschiedene Befehle • 8 Eingabetaster • Tonausgabe (Piezopiepser) • LED-Ampel • 4 weitere LEDs • Lichtsensor

Inhalt der Lernbox:

• Bestückte Mikrocontrollerplatine (Spielbrett)
• 73-seitiges A6-Handbuch

€ 44,95
CHF 56,95

Weitere Infos & Bestellung unter www.elektor.de/learn-to-program

Elektronik-Praxis für Einsteiger

Projekte bauen mit oder ohne Löten

NEU!

Dieses neue Buch ist ideal für Sie,

• wenn Sie nachbausichere Schaltungen suchen, die mit wenigen elektronischen Bauteilen auskommen, problemlos funktionieren und Spaß machen. Hier finden Sie beispielsweise Radioempfänger für UKW, Lang- und Mittelwelle, eine akustische Illusion, ein Mini-Oszilloskop zur grafischen Anzeige von Spannungsverläufen, elektronische Orgeln und weitere funktionssichere Schaltungen.
• wenn Sie elektronische Schaltungen bauen möchten, vorerst aber lieber bei bedrahteten Bauteilen und Steckbrettern bleiben, anstatt Platinen herzustellen und Bauteile zu löten. Für alle Bauprojekte wird ein Steckbrett-Aufbau gezeigt und bei vielen zusätzlich ein einfacher Aufbau auf einer Universal-Leiterplatte.
• wenn Ihnen der reine Nachbau etwas zu wenig geworden ist und Sie anfangen möchten, Schaltungen selber zu entwerfen und zu berechnen. Von ganz einfach bis etwas schwerer erfahren Sie unter anderem, wie Sie Vorwiderstände berechnen, einen Schwingkreis oder eine Spule dimensionieren.
• wenn Sie nur wenig oder keine Erfahrung mit Elektronik haben und neu einsteigen möchten. Sie erfahren das Wichtigste über den praktischen Aufbau von Schaltungen, ob nun auf dem Steckbrett oder auf einer Leiterplatte.

254 Seiten (kart.) • Format 17 x 23,5 cm
ISBN 978-3-89576-278-9

€ 32,80
CHF 40,95

Weitere Infos & Bestellung unter www.elektor.de/elektronik-praxis-einsteiger

Richtig programmieren!

Leistungsfähige Schaltungen ohne Softwarefehler

Von **Clemens Valens**
(Elektor.Labs)

Oft wird viel Zeit und Energie auf eine elegante, gut durchdachte und robuste Hardware verwendet. Heutzutage steckt im Zentrum vieler dieser Schaltungen ein Mikrocontroller, die Software braucht, um zu funktionieren. Sollte man da nicht ein gut gestaltetes, richtig geschriebenes Programm erwarten? Doch oft hapert es an der Qualität der Software, was sehr teure bis tragische Folgen haben kann.

Es ist eine bekannte Tatsache: Bugs töten Menschen und das nicht selten. Jedes Jahr sterben Menschen wegen fehlerhafter Software. Einige tödliche Unfälle mit Flugzeugen, Hubschraubern und Automobilen sind auf Software-Probleme zurückzuführen. Medizinische Geräte versagen, Gebäude gehen in Flammen auf, Schiffe versinken im Meer und einige Leute hacken sich zu Tode, alles aufgrund schlecht geschriebener Software. Eine fehlerfreie Software, die eine sinnvolle Arbeit verrichtet, gibt es nicht. Laut Wikipedia ist es dem *NASA Software Assurance Technology Centre* gelungen, Software mit weniger als 0,1 Fehlern pro 1000 Zeilen Code zu entwickeln, ein Wert, der als extrem gut anzusehen ist. Kommerzielle Organisationen haben weder Zeit noch Geld, um ein solches Maß an Qualität zu erreichen. Microsoft strebt 0,5 Fehler pro 1000 Zeilen an, wenn es um ein neues Produkt geht. Das bedeutet zum Beispiel: Windows XP, das aus 45 Millionen Codezeilen kompiliert wurde, enthielt bei der ersten Veröffentlichung mehr als 22.500 Bugs! Es wird geschätzt, dass Programme für industrielle Zwecke Fehlerquoten von 5 bis 50 Fehlern pro 1000 Zeilen aufweisen.

Software defects, so die offizielle Bezeichnung für Fehler oder Bugs, können viele Ursachen haben, von schlechtem Verständnis über zu komplexe Probleme bis hin zu schlicht schlampiger Programmierung. Entgegen der landläufigen Meinung ist die Erstellung einer guten Software nicht einfach, da sie extreme Präzision und Sorgfalt verlangt. Für jede Codezeile, die geschrieben werden muss, sind mindestens drei Dinge erforderlich:

1. ein gutes Verständnis für das Problem, das die Code-Zeilen lösen sollen;
2. die Fähigkeit, diese Lösung vollständig in der Programmiersprache auszudrücken;
3. nicht irgendwelche Tippfehler während der Eingabe des Codes einzubauen.

Der erste Punkt ist oft der schwierigste Teil, vor allem bei großen Projekten. Deshalb sind geschickte System-Architekten und gute Spezifikationen für den Erfolg eines Projektes essentiell. Punkt 2 ist auch schwierig, da es die Wahl der Programmiersprache beinhaltet. Nicht jede Sprache ist für jede Situation passend – man kann Logik nicht immer fehlerfrei in einer nicht-natürlichen Sprache ausdrücken.
Der dritte Punkt scheint lächerlich, ist aber tatsächlich ziemlich schwer zu erfüllen. Es ist leicht geschehen, dass man eine falsche Klammer verwendet oder sie ganz vergisst oder zwei Symbole vertauscht oder ein Zeichen falsch eingibt. Zwar helfen Programmier-Tools dem Entwickler und weisen auf Syntax-Fehler, Kompatibilitätsprobleme und andere Konflikte zwischen Datenobjekten hin, hundertprozentig verlassen darf man sich aber nicht auf diese Tools.

Um die Gefahr von Defekten in den Codezeilen zu reduzieren, können wir zu der Liste hinzuzufügen:

4. Sicherstellen, dass der Code in einer klaren und verständlichen Weise geschrieben ist;
5. Kommentare hinzufügen (und immer aktuell halten), um den Sinn einer Codezeile zu erklären;
6. sich an einen Coding-Standard (Programmierstil) halten.

Windows XP bestand aus 45 Millionen Code-Zeilen — und enthielt bei seiner Veröffentlichung mehr als 22.500 Fehler

Punkt 4 bezieht sich auf die Verwendung von nachvollziehbaren Namen für Funktionen und Variablen. Obwohl es mehr Schreibarbeit ist, einen Variablen-Namen wie „Beschleunigung“ zu verwenden, ist das viel deutlicher als „Acc“ oder gar „a“.
Punkt 5 wird von vielen Programmierern ignoriert. Nicht, weil sie keine Kenntnis von der Kommentar-Funktion der verwendeten Programmiersprache hätten, sondern weil sie schlichtweg zu faul sind, um sie zu benutzen.
Der letzte Punkt ist sehr wichtig. Viele Amateur-Programmierer wissen nichts über vorgegebene Programmierstile, etliche professionelle Programmierer schon, kümmern sich aber nicht darum.

Was ist ein Coding-Standard?
Die meisten Software produzierenden Unternehmen haben eine „Hausordnung“: Konventionen, wie ihre Entwickler Code schreiben sollen. Im Internet finden Sie Coding-Standards für viele Open-Source-Projekte wie GNU [1] oder Linux. Allerdings beschränken sich diese Vorschriften meist auf die Formatierung des Quellcodes, ihr Ziel ist es, einheitlichen Quellcode zu produzieren, der leichter unterhalten werden kann. Da sie das Aussehen des Codes betreffen, ist es besser, von einem Coding-Style oder Programmierungs-Stil zu sprechen
Ein echter Coding-Standard ist nicht ein Satz von Regeln, der Ihr Programm hübsch aussehen lässt, es ist ein Satz von Regeln, um Fehler in der Programmierung zu reduzieren. Heute sind C und C++ die vorherrschenden Programmiersprachen, die meisten Coding-Standards betreffen deshalb C/C++. Der Vorteil eines Standards ist, dass man statische Analyse-Tools verwenden kann, um die Semantik des Codes statt nur die Syntax zu überprüfen. Das hilft, mögliche Probleme, die ein Compiler nicht finden kann, zu identifizieren. Also, welche Art von Regeln kann man von einem Coding Standard erwarten? Hier sind ein paar Beispiele:

Vermeiden Sie explizite Eigenschaften der Sprache, die Fehler verdecken können
In C und C++ ist es erlaubt, den Zuweisungsoperator „=“ in booleschen Ausdrücken zu verwenden. Zum Beispiel ist der folgende Ausdruck gültig:

```
if (sample=get_sample()) sample += 2;
```

Diese Codezeile fügt 2 zu `sample` hinzu, wenn `sample` nicht gleich 0 ist. Warum? Nun, zunächst wird die Funktion `get_sample ()` aufgerufen und der zurückgegebene Wert der Variablen `sample` zugeordnet. Nun prüft der boolesche `if`-Operator, ob der Ausdruck zwischen den Klammern wahr oder falsch ist. In C/C++ ist false gleich 0 und wahr entspricht nicht falsch. Wenn daher `sample` nicht Null ist, wird die Bedingung als wahr angesehen und die Addition ausgeführt wird.

Year 2000, Y2K oder Millennium-Bug (1999)

Da viele Programme die vierstellige Jahreszahl auf zwei Digits abkürzten, riskierten sie zur Jahrtausendwende die Berechnung falscher Daten.
Es wurden riesige Anstrengungen unternommen, um negative Folgen dieses Fehlers zu begrenzen. Es wurden weltweit geschätzt 425 Milliarden Dollar dafür ausgegeben, doch es (oder deswegen?) passierte beinahe nichts. Etwas Ähnliches steht uns am 19. Januar 2038 bevor, wenn der UNIX-Sekundenzähler überläuft.

Aber vielleicht hatte der Programmierer dieses im Sinn:

```
if (sample==get_sample()) sample += 2;
```

Der Unterschied, ein zusätzliches „=“-Zeichen, ist subtil. In C/C++ bedeutet die Sequenz „==“ „ist gleich“. So wird der Wert 2 zu `sample` nur dann addiert, wenn der Wert von `sample` gleich der Rückgabe der Funktion `get_sample ()` ist. Dies ist eindeutig nicht das gleiche Verhalten wie zuvor.
Ist das ein Programmierfehler oder war es so gewollt? Unmöglich zu sagen. Einige Compiler

Ariane 5 Flug 501 (1996)

Der vollständige Verlust der Orientierung und der Lageinformationen der Rakete und damit auch der Satelliten 30 Sekunden nach dem Abheben war die Folge eines Software-Fehlers im Inertial Reference System. Die Ursache des Absturzes war ein 64-bit-Floating-Point-Wert, der nicht in eine 16-Bit-Integer-Variable passte, was zu einem Überlauf führte. Ein unerwartet hoher Wert wurde durch einen Algorithmus berechnet, der aus der Software der alten Ariane 4 stammte. Der finanzielle Schaden wurde auf rund 400 Millionen Dollar geschätzt.

warnen bei diesem Problem, aber nur, wenn diese Warnfunktion auch aktiviert ist. Deshalb verbietet ein Coding-Standard diese Art von Ausdrücken.

Nur initialisierte Pointer-Ausdrücke verwenden

Dies ist ein klassischer Fallstrick und die Ursache unzähliger Bugs. Wenn ein Zeiger auf ein Datenobjekt nicht initialisiert wird (wilder Zeiger), kann er auf alles Mögliche hinweisen. Ein solcher Pointer führt zu einem undefinierten Verhalten des Programms. Auch hier können einige Compiler dieses Problem erkennen, aber nur bei aktivierter Warnung. Das Verbot nicht initialisierter Pointer ist die richtige Lösung. Da wilde Zeiger gefährlich sind, beschränken einige andere Programmiersprachen ihre Verwendung.
Wilde Pointer verursachen oft Puffer-Überläufe, die Nummer 3 in der CWE/SANS-Liste der 25 gefährlichsten Softwarefehler [2].

```
int* p_some_pointer;
p_some_pointer = address_of_data_object;
p_some_pointer[34] = 3;
```

Obwohl im obigen Code-Fragment der Zeiger vor der Verwendung initialisiert wird, bleibt eine Frage: Ist der Index von 34 gültig? Wenn nicht, führt das zu einem Pufferüberlauf-Fehler.

Strong Data Typing

In C/C++ können Variablen eines Typs Variablen eines anderen Typs zugeordnet werden, solange die anderen Daten eine gleiche oder eine höhere Genauigkeit besitzen. Zum Beispiel wird die Zuordnung einer Integer-Variablen zu einer Gleitkomma-Variablen akzeptiert. Passiert dies umgekehrt, erfolgt eine Warnmeldung. Um zu verhindern, dass Birnen und Äpfel addiert werden, verbietet die Standard-Codierung dem Programmierer, Datentypen zu mischen, sofern keine explizite Umwandlung eines Datentyps in einen anderen vorgesehen ist, wie folgt:

```
float a = 3.14;
int b = (int)a;
```

Unbenutzten Code eliminieren

Normalerweise haben alle Codezeilen eines Programms eine Funktion. Allerdings kann es passieren, dass aufgrund einer fehlerhaften Programmierung Teile des Programms nicht erreichbar sind, weil der Pfad zu diesem Code abgeschnitten ist. Nehmen Sie dieses Arduino-Listing:

```
void setup(void)
{
  int a = -1;
  unsigned int b = 1;
  if (a<b) a += 2;
  Serial.begin(115200);
  Serial.println(a);
}

void loop(void)
{
}
```

Intel Pentium Gleitkomma-Division (1993)

Ein Design-Fehler in Intels brandneuem Pentium-Prozessor verursachte kleine Fehler bei Gleitkommaberechnungen in einem bestimmten Wertebereich. Obwohl der Fehler nur für wenige Nutzer relevant war, wurde er zu einem Public-Relation-Alptraum und kostete Intel letztlich rund 650 Millionen Dollar.

Welchen Wert wird der serielle Monitor für die Variable a ausgeben? -1! Warum? Wenn in einem Vergleich einer der Werte vom Typ `unsigned` ist, wird der andere Wert automatisch zum `unsigned`-Typ „befördert“. Aber ein 16-bit-Integer weist im Arduino einen Wert von -1 auf (0xffff in Zweierkomplement, die Art und Weise, wie negative Werte von den meisten Prozessoren verstanden werden), dies ist bitweise identisch mit der Unsigned-Integer-Variablen 65.535 (0xffff). Da 65.535 größer 1 ist, ist die Bedingung immer falsch und die Addition wird nie ausgeführt, der entsprechende Code ist nicht erreichbar. Die Forderung, dass der Programmierer nicht erreichbaren Code entfernen soll, wird ihn entweder dazu zwingen, den Algorithmus zu überdenken oder das Programm zu vereinfachen. Denken Sie daran, dass weniger Code auch weniger Fehler bedeutet.
Im Beispiel kann der Code den Codierungs-Standard erfüllen, indem entweder die if- Anweisung entfällt oder eine strikte Datentypisierung der Variablen `b` durch eine temporäre Umwandlung in eine Signed-Integer-Variable vorgenommen wird:

```
if (a<(int)b) a += 2;
```

Begrenze die Komplexität!
Lange Funktionen sind in der Regel komplex und schwer zu verstehen und zu testen. Aus diesem Grund kann ein Codierungs-Standard verbieten, dass eine Funktion oder Methode mehr als sagen wir 200 Zeilen Code enthält.
Tools, die die Einhaltung des Programmierstils überwachen, können die Komplexität durchaus auch nach anderen Kriterien beurteilen. Ein Beispiel ist die zyklomatische oder bedingte Komplexität, die Anzahl unabhängiger Pfade einer Funktion. Je mehr Wege es gibt, desto höher ist die Komplexität. Diese Art von Komplexität ist nur schwierig einzuschätzen, besser, Sie lassen ein Tool für Sie arbeiten.

Stil
Obwohl der Zweck eines Coding-Standards nicht darin liegt, den Code am Bildschirm hübsch aussehen zu lassen, können Styling-Regeln dafür sorgen, dass der Quellcode leichter zu lesen ist, so dass die Identifizierung potenzieller Mängel einfacher ist. Beispiele für solche Regeln sind:

- Source-Zeilen sind auf eine Länge von 120 Zeichen beschränkt;
- Tabs sollten vermieden werden;
- Jede Anweisung steht in einer separaten Zeile;
- Alle Einrückungen werden durch mindestens zwei Leerzeichen vorgenommen, dies ist in der gleichen Quelldatei konsistent;
- Die Blöcke der Statements `if`, `else if`, `else`, `while`, `do-while` oder `for` werden immer in Klammern gesetzt, auch wenn die Klammern einen leeren Block bilden;
- Klammern („{}“), die einen Block einschließen, werden in der gleichen Spalte in getrennten Zeilen unmittelbar vor und nach dem Block gesetzt.

Populäre Coding-Standards

Die am weitesten verbreiteten Coding-Standards sind in der Reihenfolge ihrer Beliebtheit beschrieben (nicht aufgeführt sind Hausstandards):

MISRA C (und C++), erstellt von der Motor Industry Software Reliability Association, unterstützt Entwickler der Automobilindustrie bei der Erstellung sicherer und zuverlässiger Software im Fahrzeugbereich. MISRA gibt es gegen eine geringe Gebühr.

www.misra.org.uk/

CERT C++ (und C) diese Initiative aus dem Software Engineering Institute der Carnegie Mellon University ist bestrebt, unsichere Codierungspraktiken zu beseitigen, die von böswilligen Personen ausgenutzt werden können.

https://www.cert.org/
(beachten Sie die durchgängige Verwendung des sicheren https-Protokolls.)

HICPP - High Integrity C++, frei zur Verfügung gestellt von PRQA. Bietet Leitprinzipien für Wartung, Portabilität , Lesbarkeit und Sicherheit , indem sie den ISO C++-Sprachstandard beschränkt und die Flexibilität begrenzt.

www.codingstandard.com

JSF AV++ - der kostenlose Joint Strike Fighter Air Vehicle C++ Code Standard des Rüstungskonzerns Lockheed Martin soll dazu beitragen, dass Programmierer Code entwickeln, der keine Fehler enthält, die zu katastrophalen Ausfällen und erheblichen Schäden an Personen und/oder Geräten führen könnten. (Nicht zu verwechseln mit den erheblichen Schäden an Personen und/oder Geräten, die durch perfekt funktionierende militärische Software verursacht werden.)

www.jsf.mil/downloads/documents/JSF_AV_C++_Coding_Standards_Rev_C.doc

Eine lange Liste von statischen Analysetools für viele Programmiersprachen finden Sie hier:

http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

Fehler einer Patriot-Rakete (1991)

Während des ersten Golfkriegs scheiterte eine US-amerikanische Patriot-Rakete in Saudi-Arabien daran, eine irakische Scud-Rakete abfangen. 28 Soldaten wurden getötet und rund 100 verletzt. Die Ursache war ein Rundungsfehler in den Zeitberechnungen der Software, so dass einige der Ziele ignoriert wurden.

Software-Fehler sind tödlich!

Viele Programmierer verstoßen gegen diese Regeln, einige Programmeditoren tun dies auch, weil sie automatisch Tabulatoren einfügen. Warum Tabs so schlecht sind? Weil sie die Quellcode-Formatierung verderben, wenn verschiedene Personen unterschiedliche Tab-Distanzen verwenden.

Schlechte Angewohnheiten

C/C++-Compiler verfügen meist über eine große Sammlung von Standard-Bibliotheken. Viele Programmierer verlassen sich auf die Verfügbarkeit dieser Bibliotheken und nutzen sie aus Gewohnheit. Leider können Funktionen in diesen Bibliotheken Plattform-spezifisches, nicht spezifiziertes, undefiniertes, Implementierungs-definiertes oder anderweitig schlecht definiertes Verhalten aufweisen. Hier sind ein paar Regeln, die die Verwendung von einigen beliebten Funktionen und Bibliotheken untersagen:

- Der Fehler-Indikator `errno` darf nicht verwendet werden;
- Die Bibliothek <locale.h> und die Funktion `setlocale` dürfen nicht verwendet werden;
- Die Signal-Behandlungsmöglichkeiten <signal.h> dürfen nicht verwendet werden;
- Die Eingabe/Ausgabe-Bibliothek <stdio.h> darf nicht verwendet werden;
- Die Funktionen `atof`, `atoi` und `atol` aus der Bibliothek <stdlib.h> dürfen nicht verwendet werden;
- Die Funktionen `abort`, `exit`, `getenv` und `system` aus der Bibliothek <stdlib.h> dürfen nicht verwendet werden;
- Die Timing-Funktionen der Bibliothek <time.h> dürfen nicht verwendet werden.

Beachten Sie das Verbot der Input/Output-Bibliothek <stdio.h>. Ja, Sie sollen Ihre eigene `printf`-Funktion schreiben.

Jetzt liegt es an Ihnen

Die oben aufgeführten Regeln, alle aus echten Coding-Standards, mögen es schwerer machen, aber sie sind nicht in Stein gemeißelt. Für viele Regeln der Coding-Standards sind Ausnahmen zugelassen. Andere Regeln sind umstritten und es liegt an Ihnen, zu entscheiden, ob Sie sie respektieren wollen oder nicht. Im Internet finden Sie eher philosophische Diskussionen über bestimmte Regeln, was beweist, dass diese durchaus interpretationsfähig sind. Denn selbst ein Coding-Standard kann Bugs haben!

(130271-I)

[1] GNU Coding Standard: www.gnu.org/prep/standards/standards.html

[2] Top 25 der gefährlichsten Softwarefehler: http://cwe.mitre.org/top25/

Tödliche Strahlentherapie (1985-1987, 2000)

Durch die fehlerhafte Software des medizinischen Strahlentherapie-Gerät Therac-25 wurden falsch berechnete Strahlendosen verabreicht. Einige Patienten erhielten bis zum 100-fachen der vorgesehenen Dosis, mindestens drei von ihnen wurden getötet. Ein ähnlicher Bug tauchte in Panama City im Jahr 2000 auf, wo die Therapie-Planungssoftware verschiedene Dosierungen in Abhängigkeit von der Reihenfolge berechnete, in der die Daten eingegeben wurden. Dieser Bug tötete mindestens fünf Patienten.

Alle Elektor-Artikel der Jahre 2000 bis 2009 auf DVD!

NEU!

Ein Muss für jeden Elektor-Leser!

ISBN 978-3-89576-292-5

€ 89,00 • CHF 110,40

Jetzt unter www.elektor.de/2000-2009 bestellen!

EveryCircuit

Die App für Elektronik-Enthusiasten

Von **Harry Baggen** (NL)

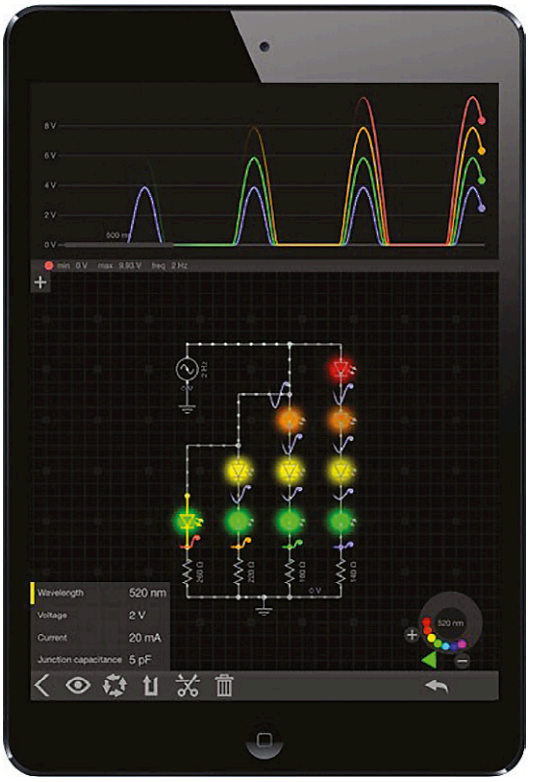


Bild 1.
Eine Simulation mit EveryCircuit auf einem iPad.

Wie anstrengend kann Elektronik sein? Wenn Sie mit EveryCircuit beginnen, ist sie tatsächlich ein Kinderspiel. Mit dieser interaktiven App simulieren Sie Ihre Schaltung, der Lötkolben hat Pause, er bleibt in der Werkzeugkiste. Auf Ihrem PC oder Tablet können Sie sogar die Wege der Elektronen verfolgen!

Das Entwerfen und Entwickeln elektronischer Systeme, seien sie simpel oder komplex, ist heute ohne Computer kaum mehr denkbar. Das Zeichnen der Schaltung, das Simulieren der Funktionen und das Layouten der Platine, viel Arbeit, die der Computer abnimmt. In den letzten Jahren sind Smartphone und Tablet hinzugekommen, oft dienen sie nur als Rechenhilfe, oder sie schaffen technische Informationen aus dem Web heran. Relativ lange hat es gedauert, bis Schaltungsdesign- und Simulationsprogramme auch in der mobilen Welt Fuß gefasst haben.

Doch EveryCircuit beweist, dass sich hier etwas bewegt. Wie sinnvoll der Einsatz komplexer Software auf den Minibildschirmen von Smartphones ist, das allerdings soll hier dahingestellt bleiben. EveryCircuit reiht sich zwar in die Programme ein, die elektronische Schaltungen simulieren, doch diese App ist erfrischend anders. EveryCircuit will dem Anwender klar und verständlich das Verhalten seines Entwurfs vor Augen führen, ohne ihn mit technischem Ballast zuzuschütten. Der Fluss von Signalen, Spannungen und Strömen wird „live“ dargestellt, die Schaltungseigenschaften lassen sich sogar während der Simulation verändern. EveryCircuit ist als App für Android und iOS verfügbar, es läuft aber auch auf Windows- und Linux-Systemen, wenn dort der Browser Chrome installiert ist. In unserem ersten Anlauf haben wir das Programm auf einem Tablet unter Android und auf einem PC unter Windows erprobt. Wir beziehen uns hier auf die Version, die im Browser Chrome läuft. Die Bedienung der Android- und iOS-Versionen ist nahezu identisch, nur sind dort die Finger anstelle der Maus in Aktion.

Eine Testversion von EveryCircuit steht im Web zum Download bereit [1], sie ist nützlich, um das Programm kennenzulernen. Allerdings ist das Fenster, das den Entwurf darstellt, stark eingeschränkt. Dort können lediglich sechs Komponenten platziert werden. Nach der Eingabe eines Lizenzschlüssels ist diese Einschränkung aufgehoben. Auch wird der Zugang zur Community geöffnet, und Entwürfe können in einer Cloud abgelegt werden. Der Zugriff auf die eigenen Schaltungen ist dann über mehrere Geräte möglich, beispielsweise über ein Tablet und einen PC. In die Cloud geladene Dokumente lassen sich für fremden Zugriff sperren, oder sie werden für die Community freigegeben. Der Bestand an freigegebenen Inhalten ist inzwischen beachtlich. Ein Lizenzschlüssel ist ein Jahr gültig, er ist im Elektor-Shop erhältlich [2].

Simulation

Nach dem Programmstart fällt zuerst auf, dass nur wenige Bedienelemente vorhanden sind. Da eine Dokumentation (noch) fehlt, ist hier intuitives Erkunden geboten. Auf einer horizontalen Menüleiste, die sich oben im Fenster befindet, sind häufig verwendete Schaltungskomponenten aufgereiht: Spannungs- und Stromquellen, Widerstände, Trafos, Transistoren, Schalter, Gatter, und sogar ein Timer des Typs 555. Eine Liste oder Bibliothek mit Typenbezeichnungen sucht der Anwender vergebens, stattdessen können die Eigenschaften der Komponenten festgelegt werden, nachdem sie im Zeichenfeld platziert sind.

Nach Anklicken einer Komponente wird sie in das Zeichenfeld übernommen, sie lässt sich dann mit der Maus an den ihr zugedachten Ort ziehen. Zwei Komponenten werden über Leitungen miteinander verbunden, indem die Anschlüsse nacheinander angeklickt werden. Wird eine Komponente selektiert, erscheinen in der Leiste unten im Fenster diverse Symbole, mit denen die Komponente gedreht, gespiegelt, gelöscht und in ihren Eigenschaften angepasst werden kann. Das Anklicken des Schlüsselsymbols führt zu den Einstellungen der Komponente. Mit dem virtuellen Drehknopf rechts unten im Fenster sind die Einstellungen modifizierbar. Über das Symbol, das wie ein Auge aussieht, ist die Wahl des Signalverlaufs möglich, der während der Simulation dargestellt werden soll. Wenn eine Komponente angeklickt wird, erscheint der Verlauf des Stroms, nach Anklicken einer Leitung wird der Spannungsverlauf angezeigt. Ein zweiter Klick auf eine selektierte Komponente oder auf eine leere Stelle im Zeichenfeld führt dazu, dass unten im Fenster zwei mit „t“ und „f“ gekennzeichnete Dreiecke erscheinen (das letzte nur in der registrierten Version). Die Buch-

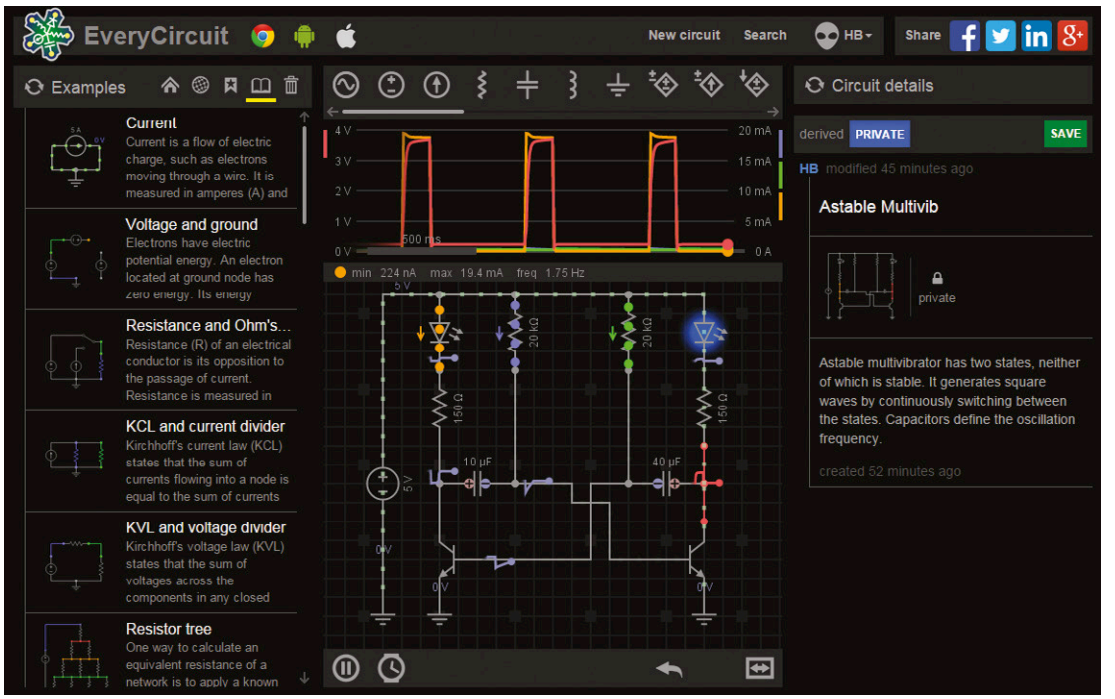


Bild 2.
Startseite von EveryCircuit im Browser Chrome. Links sind Anwendungen gelistet, in der Mitte der Arbeitsbereich mit den simulierten Signalen, rechts das aktuelle Projekt und die Beschreibung.

staben deuten die Simulationsarten „transient“ und „frequency“ an. Ein Klick auf ein Dreieck startet die Simulation. Bei der Frequenzanalyse wird nach dem Vorbild anderer Simulationsprogramme ein Bode-Diagramm eingeblendet. Beeindruckend ist insbesondere die Transienten-Analyse, da dann das Schaltungsverhalten und das Verhalten der Ströme gewissermaßen mitverfolgt werden kann. Wenn eine Spannungsquelle, ein Widerstand oder eine LED selektiert sind, werden der hindurchfließende Strom oder die abfallende Spannung angezeigt. Die meisten Komponenten verhalten sich auf dem Bildschirm wie in der Realität. LEDs leuchten nur auf, wenn genügend Strom fließt, und Schalter sind auch während der Simulation bedienbar. Diese Features können den Newcomer beim Start in die Welt der Elektronik effizient unterstützen. Allerdings sollten bereits einige Grundkenntnisse der Eigenschaften elektronischer Bauelemente vorhanden sein, hier bietet das Programm keine Hilfen an. Die von der Community bereitgestellte Sammlung enthält jedoch genügend Lehrstoff, ergänzt durch eine Liste mit Schaltungsbeispielen, die zum Vertiefen der Kenntnisse besonders geeignet sind.

Fazit

EveryCircuit ist eine App, mit der insbesondere kleine Schaltungen gut simulierbar sind. Der Umgang mit der App ist nicht kompliziert, wenn auch zunächst etwas gewöhnungsbedürftig. Die Darstellung von Spannungen und Strömen in „Echtzeit“ macht die Schaltungsfunktionen transparent, unterstützt von dem Feature, dass die Parameter der Komponenten während der Simulation variiert werden können. Der Newcomer in der Welt der Elektronik wird zu Lernerfolgen geführt, ohne Bauteile beschaffen und zum Lötkolben greifen zu müssen. Nur die Einstellungen der Halbleiter-Parameter dürften manchmal nicht sofort verständlich sein.

EveryCircuit ist kein Ersatz für ausgewachsene, etablierte Simulatoren. Trotzdem (oder gerade deshalb?) macht es Spaß, mit dieser anschaulich gestalteten, ballastarmen App zu arbeiten.

(140334)gd

Weblinks

[1] www.everycircuit.com

[2] Die Jahreslizenz für EveryCircuit ist im Elektor-Shop erhältlich, sie kostet 7,00 €. Mitglieder der Elektor-Community erhalten sie zum ermäßigten Preis, er beträgt nur 6,30 €. Gehen Sie auf www.elektor.de/everycircuit-app, dort können Sie die Jahreslizenz erwerben.

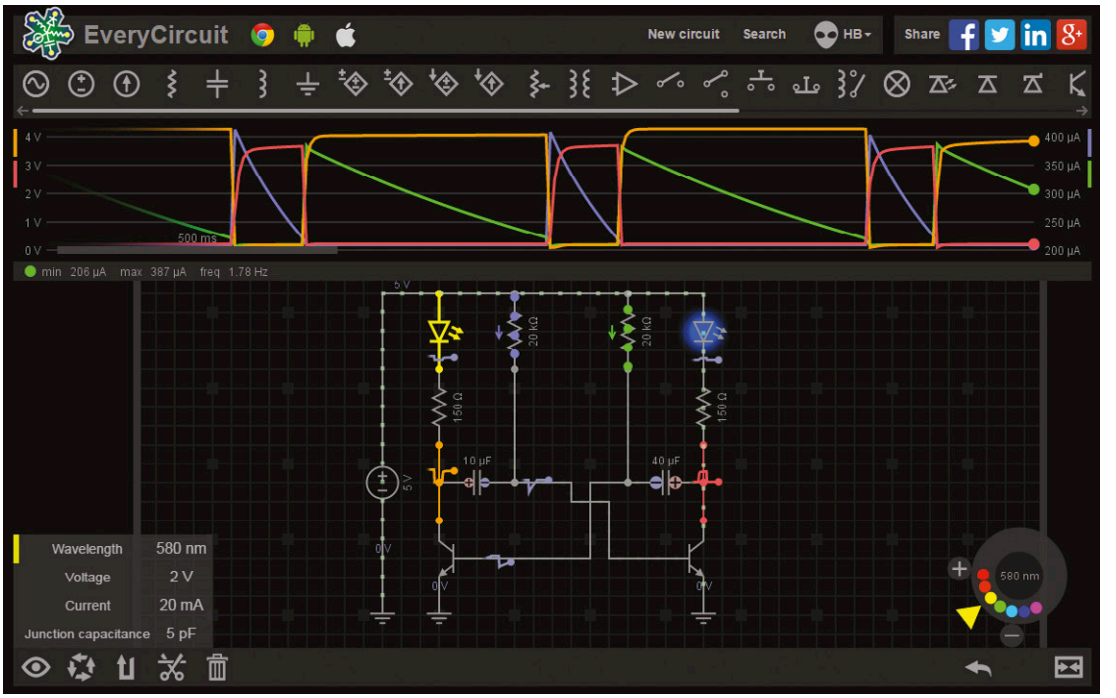


Bild 3.
Der linke und rechte Bereich der Startseite lassen sich ausblenden, so dass die Übersichtlichkeit steigt. Links unten erscheinen die Eigenschaften einer LED, rechts unten sind mit einem virtuellen Drehknopf diverse Parameter einstellbar.

Hexadoku Sudoku für Elektroniker

Auch in dieser Doppelausgabe finden Sie wieder Ihr vertrautes Hexadoku. Der Lötkolben kann ausgeschaltet bleiben, stattdessen wird hier mit Bleistift oder Kugelschreiber gearbeitet. Wie immer warten fünf Gutscheine auf diejenigen, die uns die Zahlen in den grauen Kästchen zuschicken (und dazu etwas Glück haben). Auf los geht's los!

Die Regeln dieses Rätsels sind ganz einfach zu verstehen: Bei einem Hexadoku werden die Hexadezimalzahlen 0 bis F verwendet, was für Elektroniker und Programmierer ja durchaus passend ist.
Füllen Sie das Diagramm mit seinen 16 x 16 Kästchen so aus, dass alle Hexadezimalzahlen von 0 bis F (also 0 bis 9 und A bis F) in jeder Reihe, jeder Spalte und in jedem Fach mit 4 x 4 Kästchen (markiert durch die dickeren schwarzen Linien) **genau einmal** vorkommen. Einige Zahlen sind bereits eingetragen, was die Ausgangssituation des Rätsels bestimmt.
Wer das Rätsel löst - sprich die Zahlen in den grauen Kästchen herausfindet - kann einen von fünf Buchgutscheinen im Wert von 50 Euro gewinnen!

Einsenden

Schicken Sie die Lösung (die Zahlen in den grauen Kästchen) per E-Mail, Fax oder Post an:
Elektor – Redaktion – Süsterfeldstr. 25 – 52072 Aachen
Fax: 0241 / 88 909-77 E-Mail: hexadoku@elektor.de
Als Betreff bitte nur die Ziffern der Lösung angeben!

Einsendeschluss ist der 28. Februar 2015!

Die Gewinner des Hexadokus aus der November-Ausgabe stehen fest!

Die richtige Lösung ist: **63D95**.
Einen Elektor-Buchgutschein über je 50 € haben gewonnen:
Andrej Marn, Gerrit van Leeuwen, Jozef Bouwen, Francois Jongbloet und Larry Burns.
Herzlichen Glückwunsch!

	3			E	9		2		5	6			C	F	
9											F	2	3		6
			6					3	D	9		A	4	E	0
		8	C						E			9	D	5	
A				5	6				7	E				1	
0				F			C	4		A	6		5		D
						9			F		0	8	A		3
5					0		3	C		B			F		
		B			7		D	2		3					F
6		D	2	1		0			B						
8		7		2	3		9	D			C				1
	E				8	A				1	7				C
	6	5	A			1						C	9		
C	D	E	9		4	3	8					F			
4		1	7	B											E
	B	F			D	7		1		C	A			3	

6	8	2	4	D	B	3	5	C	E	1	9	7	0	F	A
C	5	0	7	A	E	F	2	B	6	D	3	1	8	4	9
1	9	3	A	0	6	4	7	F	2	5	8	B	C	D	E
B	D	E	F	C	8	9	1	0	7	4	A	2	3	5	6
D	2	F	3	5	0	6	C	A	9	7	1	8	E	B	4
0	1	5	9	3	4	D	A	2	8	E	B	F	6	7	C
4	7	6	E	B	F	8	9	D	C	3	5	0	1	A	2
8	A	B	C	1	2	7	E	4	F	0	6	3	D	9	5
2	B	1	5	9	C	0	8	E	D	6	F	A	4	3	7
7	C	8	D	E	A	1	3	5	0	9	4	6	F	2	B
F	E	9	6	2	D	B	4	1	3	A	7	C	5	8	0
A	3	4	0	6	7	5	F	8	B	2	C	D	9	E	1
E	0	A	1	F	9	C	B	3	5	8	2	4	7	6	D
3	F	7	2	4	5	E	D	6	1	B	0	9	A	C	8
9	6	D	B	8	3	A	0	7	4	C	E	5	2	1	F
5	4	C	8	7	1	2	6	9	A	F	D	E	B	0	3

NF-Generator Philips GM2308 x 2 (1950, 1964)

Schweben ist besser!

Von **Jan Buiting** (Chefredakteur Elektor-UK/US)

Im dunklen, prädigitalen Zeitalter war die Erzeugung stabiler, präziser und verzerrungsarmer Audiosignale mit vertretbarem Aufwand nicht gerade einfach. In den frühen 1950er Jahren gelang dies den Forschern im Physik-Labor von Philips in Holland mit Hilfe von etwas ungewöhnlichen, aber technisch überzeugenden Methoden dennoch.

Zuhause habe ich ein kleines Labor, das fast komplett mit antiken Messgeräten von Philips ausgestattet ist. Ich kann nur vermuten, dass ich damit ganz ähnlich arbeite wie Elektroniker vor 40 bis 60 Jahren: Zuerst werden alle relevanten Geräte eingeschaltet, um auf Betriebstemperatur zu kommen. Bis es soweit ist, genehmige ich mir erst einmal einen Kaffee. Wie man auf der Titelseite meines Buches [1] sehen kann, sind alle Messgeräte mit Röhren bestückt. Daher der wohlbekannte Geruch und das typische leichte Glühen, wenn ich wieder in mein Labor eintrete. Meine Geräte brummen lange nicht so laut, wie das gerne vermutet wird. Man hört auch kein Plopp, wenn die Geräte eingeschaltet werden. Selbstverständlich ist alles gut abgesichert und isoliert.

Letzthin begrüßte mich beim Eintreten aber der verdächtige Geruch eines NF-Generators vom Typ GM2315. Irgendetwas musste hochgegangen sein. Ich war eh nicht so ganz zufrieden mit diesem Apparat, und er stand schon auf meiner To-do-Liste zur Inspektion und Überholung.

Ich erinnerte mich, dass ich noch irgendwo einen NF-Generator des Typs GM2308 eingelagert hatte. Also entschied ich mich, diesen zunächst als Ersatz für den 2315 einzusetzen. Woran ich mich

Technische Daten:

- Frequenzbereich 0...16.000 Hz
- MiniWatt-Röhrenbestückung
- Ausgangsspannung 0...25 V
- Ausgang schaltbar symmetrisch/unsymmetrisch
- Genauer Abschwächer (Bereich 1:10^4)
- Nullabgleich der Frequenz mit magischem Auge
- Verstärkung externer Signale
- Wenig Brummen, Rauschen und Verzerrungen
- Geringe Einflüsse der Netzspannung
- Geeignet zum Einsatz in den Tropen
- Gewicht „nur“ 13 kg
- Leistungsaufnahme „nur“ 50 W

allerdings nicht erinnerte war, dass der Ersatz gleich doppelt so groß und schwer war. Der physikalische Austausch wurde also von Audiosignalen begleitet, die weder sinusförmig waren, noch hier abgedruckt werden können.

Geschichte & Einsatz

Sowohl der GM2315 als auch der GM2308 produzieren Sinussignale im Audiofrequenzbereich, die sich für Laboratorien und zur Gerätereparatur eignen. Sie unterscheiden sich in den Punkten Genauigkeit, Frequenzbereich und Preis. Während der Typ 2315 immerhin 20 kHz schafft ist beim 2308 schon bei 16 kHz Schluss. Diese Frequenzgrenzen waren in den frühen 1950ern nicht so fix, da es damals noch keine Abtastrate von 44,1 kHz etc. gab. HiFi war eher ein Konzept oder eine Art Entwicklungsziel. Als etwa um 1950 herum der Generator GM2308 entwickelt wurde, schienen 16 kHz genug für alle praktischen Zwecke der Elektroakustik.

Beim GM2308 ist ein viel genauerer und leistungsfähigerer Ausgang samt Abschwächer vorhanden als beim 2315. Es sind halbwegs genaue Pegel in vier Bereichen bis zu 25 V einstellbar. Beim 2315 ist die Einstellung von Pegel und Frequenz eher ein Glücksspiel.

Mit seinen zwei großen Drehknöpfen war das GM2308 ein prominentes Exemplar der GM-Serie von Philips. Es fiel allein schon wegen der schwarzen Front, den großen, geriffelten Knöpfen, dem grauen Stahlblechgehäuse und dem (zurückschnappenden) Trageriemen aus Leder auf. Es gibt auch einige Fotos davon, die im renovierten Eindhoven/Waalwijk-Physik-Labor in den Räumen der akustischen Forschung aufgenommen wurden.

Wie neu oder ramponiert?

Als ein Freund mitbekam, dass ich mein in grau und silber gehaltenes GM2308-Exemplar langsam mit Hilfe eines Stelltrafos hochfuhr, meinte er: „Ich habe auch so ein Ding mit zwei großen Drehknöpfen, aber das ist komplett schwarz.“ Er war sich aber sicher, dass es sich dabei ebenfalls um den Typ GM2308 handelte – wegen der schwarzen Front wohl deutlich älter als mein Generator. Schließlich wurde das „Ding“ aus seiner Garage ausgebuddelt und einer ersten Grobreinigung unterzogen. Der alte Apparat wurde wohl gut herumgestoßen, denn er hat Dellen und Kratzer sowie eine verbogene Einstellachse. Er wurde ebenfalls langsam hochgefahren und zeigte bis auf ein totes magisches Auge keine Probleme.

Schwebungen!

Für eine bessere Stabilität und Genauigkeit als die damals verwendeten RC- und Wien-Brücken-Oszillatoren setzte Philips auf das Prinzip der Interferenz zweier HF-Oszillatoren. Die entstehende Schwebung hatte die Frequenz der Differenz der beiden Oszillatoren. Wenn man z.B. zwei Frequenzen mit 99 kHz und 100 kHz mischt, erhält man eine Schwebung von 1,0 kHz. Das ist wie bei alten Mittelwellen-Radios, die bei einer Fehlabstimmung von 1 kHz ein Pfeifen hören lassen. Doch warum der ganze Aufwand mit zwei HF-Oszillatoren, einem Mischer, einem Tiefpass und mehr, wenn man 1 kHz auch mit ein paar Rs und Cs sowie einer Röhre erzeugen kann?

Die gewählte aufwendige Lösung bringt den Kenner zum Schmunzeln, da er weiß, dass LC-Oszillatoren damals sehr viel weniger Verzerrungen als Oszillatoren mit RC-Bestückung aufwiesen. Und ein L für z.B. 100 Hz ist ein Monster, mit dem kaum weniger als 10 % Ungenauigkeit möglich gewesen wären, was einen unmöglich großen Drehkondensator zur Folge gehabt hätte. Da ist es einfacher, zwei LC-Oszillatoren für HF zu bauen. Einer davon war im Bereich 85...100 kHz und der andere im Bereich 100...101 kHz abstimmbar. Damit erreichte man die gewünschte Schwebung.

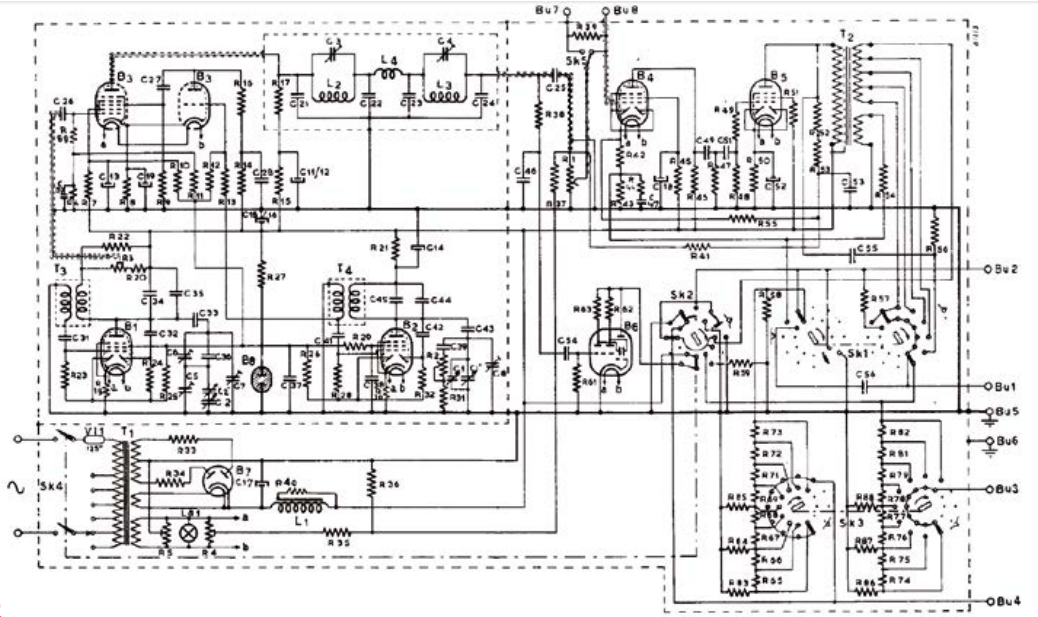

1

Mathematisch ergeben sich beim Mischen der beiden Frequenzen f_1 und f_2 mit einer Röhre folgende Produkte im Anodenstrom:

f_1, $2f_1$, $3f_1$ …
f_2, $2f_2$, $3f_2$, …
f_1+f_2, f_1-f_2, f_1+2f_2, f_1-2f_2, … etc.

Bei den gewählten Frequenzbereichen der beiden Oszillatoren sieht man auf Anhieb, dass bei Unterdrückung höherer Frequenzanteile wie $2f_x$, $3f_x$ etc. mit einem simplen Tiefpass, ein Signal im Audiofrequenzbereich von 0...16.000 Hz resultiert. Ganz schön clever!

Wenn man die 100 kHz als HF sieht, dann hat Philips mit der HF-Technik dank der Herstellung von Röhren-Radios und der dafür nötigen Fertigungstechnik für Spulen viel Erfahrung in der Massen-



2

produktion. So war es auch kein Wunder, dass die charakteristischen Trimmer beim GM2308 an einer servicefreundlichen Position oberhalb der großen Spulen verbaut waren. Außerdem waren die zentralen LC-Kreise mit einer Metallabschirmung versehen (**Bild 1**).

Magie

Die Schaltung des GM2308 in **Bild 2** zeigt wenig Überraschungen: Die Röhren B1 und B2 entsprechen den Oszillatoren mit 85...100 kHz und 100...101 kHz. C1 und C2 sind die beiden Abstimm-Kondensatoren, die mit den Frequenzknöpfen auf der Front verbunden sind. Mit R2 kann man eine Feinabstimmung des 100...101-kHz-Oszillators im Bereich weniger Hz vornehmen. Gemischt wird mit B3, worauf der erwähnte Tiefpass folgt. Bei B6 handelt es sich um das „magische Auge“ EM34, das direkt mit dem Tiefpass verbunden die Frequenzdifferenz von B1 und B2 anzeigt. Vor jeder Benutzung muss das Messgerät kalibriert werden. Hierzu stellt man die beiden großen Wählknöpfe auf ihre Ausgangsstellung und verstellt R2 mit Fingerspitzengefühl, bis das Auge langsam blinkt (siehe **Bild 3**). Auf diese Weise stimmen f_1 und f_2 bis auf weniger als 1 Hz überein. Dieser Abgleich driftet natürlich mit der Zeit und sollte laut Philips alle halbe Stunde wiederholt werden. Nicht schlecht! Selbst wenn die EM34 merkbar flackert, liegt die Differenz doch um die 1 Hz!

BU7 und BU8 erlauben die Einspeisung externer Signale in die Endstufe mit B4 und B5. Normalerweise liegt das NF-Signal des Tiefpasses an B4. Der Zweiröhrenverstärker ist der Spielverderber, denn seine Verzerrungen übertreffen sicher das sorgfältig aufbereitete NF-Signal (siehe **Tabelle 1**). Nach dem Abschwächer beträgt die maximale Ausgangsleistung normalerweise 625 mW. Wenn man stattdessen 1 W wählt, gibt es stärkere Verzerrungen.

Der Abschwächer ist mit dem Schalter Sk3 realisiert, der zwei Ebenen aufweist und einen Bereich von 40 dB (1:10.000) abdeckt. Sk2 erlaubt die Umschaltung zwischen symmetrischem und unsymmetrischem Ausgang.

Mit Sk1 wählt man via Ausgangstrafo T2 die Ausgangsimpedanzen 5/250/600/1.000 Ω. In Position „I-90V-ASYM“ wird die Spannung der Primärwicklung direkt auf den Ausgang gegeben. In diesem Fall werden 90 V an minimal 100 kΩ geboten.

Die als magisches Auge fungierenden Röhren haben einen schlechten Ruf was Lebensdauer und Verfügbarkeit angeht. Daher erhält B6 nur Spannung während des Null-Abgleichs. Aber bei Problemen hilft Technik: Es gibt nämlich LED-basierten Ersatz für die Typen EM34, EM4, EM1, EM35 [2].

Never change a winning team...

Das Foto von **Bild 4** zeigt das Innenleben beider GM2308-Chassis nebeneinander – links = älter und rechts = jünger. In den 10 Jahren die dazwi-

ESTD 2004

Retronik ist eine monatliche Rubrik, die antiker Elektronik und legendären Elektor-Schaltungen ihre Referenz erweist. Beiträge, Vorschläge und Anfragen telegrafieren Sie bitte an Jan Buiting (editor@elektor.com).

schen liegen gab es wohl keine großen Änderungen. Die neueren senfgelben Kondensatoren rechts deuten auf eine Fertigung um 1964 hin. Von daher „lebte“ der Typ GM2308 etwa 15 Jahre vom Konzept bis zum Ersatz (mit Transistoren). Verglichen mit der Versionsabfolge des iPhone ist das nahezu ewig. Interessant ist auch der Ring der Kondensatoren des älteren GM2308: Damit wird die „geerdete“ Seite bzw. die Außenseite des Dielektrikums angezeigt.

Beim neueren GM2308 gibt es einige Widerstände mit höherer Belastbarkeit, vermutlich um Widerstandsänderungen durch Feuchtigkeitsaufnahme, mechanische oder thermische Belastung zu reduzieren. Es gibt noch weitere kleine Unterschiede: Das neuere Exemplar ist mit einer Pope- und einer Tungsram-Röhre bestückt, während das ältere ausschließlich original MiniWatt-Röhren von Philips aufweist.

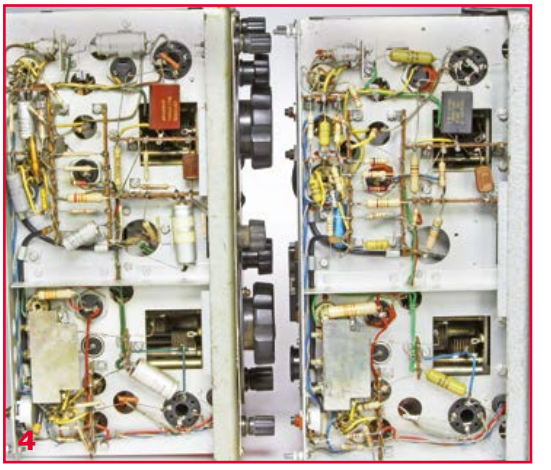
Der ältere Generator hat rückseitig eine Buchse, dank der man die Endröhre EL84 als Verstärker für externe Signale einsetzen kann. Ist ein 6,3-mm-Klinkenstecker eingesteckt, ist die Verbindung zwischen dem Generatorausgang und dem Eingangs-Gitter von B4 unterbrochen. Scheinbar entfiel dieses Feature in den frühen 1960ern. Kein Wunder, denn ein GM2308 verfügt weder über Frequenz-Sweeps noch Bursts. Der ältere Generator ist nicht einer aus der ältesten Produktion, denn er ist mit einer Dual-Diode-Pentode EBL21 in der Endstufe bestückt. In der ersten Serie steckte eine EL84.

Betrieb

Die Verwendung des GM2308 ist gewöhnungsbedürftig. Zwar ist der Nullabgleich mit der EM34 noch intuitiv, nicht aber die Einstellung von Frequenz und Ausgangspegel. Für die Frequenz muss man die Anzeigen beider Oszillatoren addieren. Das mündet in Aufgaben wie 14.400 + 750 ist gleich... In der Praxis stellt man den linken Knopf auf einen runden kHz-Wert ein und justiert rechts die Nachkommastellen, denn hier geht es ja von 0...1.000 Hz. Für den Ausgangspegel wird mehr als Adam Riese verlangt: Beispielsweise 3 x 10^{-x} am abgestuften Ausgang. In der Praxis wird man einfach dem untersuchten Radioapparat gelauscht haben und den Abschwächer so lange erhöhen, bis man etwas vernimmt. Damit verhindert man Überlastungen, auch wenn Röhren vieles verzeihen.

Tabelle 1. Verzerrungen

Frequenz	Max. Verzerrungen bei P_{out}			Note
	400 mW	625 mW	1 W	
30 – 200 Hz	2%	3%	4%	Verzerrungen und Brummen
200 – 16.000 Hz	0,75%	1%	2%	Verzerrungen



Mein GM2308 hält die originalen Daten zur Verzerrung heute noch ein. Es musste nur ein Poti minimal justiert werden, um den Frequenzbereich zu kalibrieren. Dies ist wohl den gelben Kondensatoren zu verdanken. Beim älteren GM2308 sieht es nicht so gut aus, aber immerhin hat nichts gequalmt. Es musste nur etwas runderneuert werden...

Da ich leicht älter als das neuere GM2308 bin, kann ich zwar die 16 kHz noch hören, aber nicht viel mehr. Beim Test mit Sinussignalen reicht mein GM2308 für alle meine Verstärker - ob transistorisiert oder mit Röhren. In Abwandlung eines berühmten Zitats von Ludwig Wittgenstein gilt: *Was man nicht hören kann, darüber soll man schweigen.*

(140368)

Literatur und Weblink:

[1] Retronik, 91 Geschichten elektronischer Antiquitäten, Elektor International Media, ISBN 978-3-89576-291-8

[2] www.pcvana-z.nl/ledogen.html

Mit Visual Basic in die analoge Welt

1 VB-Express und die Hardware

Visual Basic zählt nach wie vor zu den sehr weit verbreiteten Programmiersprachen. Seine Beliebtheit resultiert gerade für den Einsteiger aus der schnellen Erlernbarkeit und der einfachen Lesbarkeit des Programmcodes. Dieses Buch ist für Einsteiger in die Programmiersprache Visual Basic.NET gedacht, auch und gerade unter Berücksichtigung der Bedürfnisse von Elektronikern.

287 Seiten (kart.) • ISBN 978-3-89576-270-3
€ 36,80 • CHF 45,95

Grundlagen für Visual Basic-Einsteiger und VB6-Umsteiger

2 VB Express 2010

Dieses Buch unterstützt den Anwender bei den ersten Schritten mit Visual Basic, in dem es sich auf die Werkzeuge der Toolbox und deren Eigenschaften konzentriert, die zum Schreiben praktisch verwertbarer Programme notwendig sind. Zu jedem Thema findet der Leser ausführlich kommentierte Beispielprogramme, die er selbst ausprobieren kann und die sich auf das Mindeste beschränken, was zum Starten der Software notwendig ist.

284 Seiten (kart.) • ISBN 978-3-89576-269-7
€ 34,80 • CHF 43,95

45 Experimente mit Hard- und Software für Elektroniker

3 Raspberry Pi

Dieses Buch beschreibt 45 spannende und interessante Projekte mit Raspberry Pi, wie zum Beispiel ein Wechselblinklicht, eine Motorregelung, Erzeugen und Verarbeiten analoger Signale, ein digitales Thermometer, ein Lichtmesser. Aber auch kompliziertere Projekte wie eine Motor-Geschwindigkeitsregelung, ein Webserver mit CGI (Common Gateway Interface) und Client-Server-Programme werden vorgestellt. Sie können dieses Buch als Projektbuch verwenden und die Projekte nachbauen, um sie dann in der Praxis einzusetzen.

271 Seiten (kart.) • ISBN 978-3-89576-273-4
€ 39,80 • CHF 49,95

Arrays und serielle Daten

4 LabVIEW 2

Der zweite Band der LabVIEW-Lehrbuchreihe beschäftigt sich u.a. mit Arrays, Cluster und den seriellen VISA-Funktionen. Als Erstes werden vier neue zusammengesetzte Datentypen (Enum, Ring, Array, Cluster) vorgestellt und deren Verwendung wird anhand zahlreicher praktischer Beispiele und Übungen erläutert. Danach wird es praktisch: Ein 8051er-Mikrocontrollersystem dient dabei als Datenquelle und -senke für verschiedene LabVIEW-VIs.

248 Seiten (kart.) • ISBN 978-3-89576-274-1
€ 34,80 • CHF 43,95

Der professionelle Ratgeber für Funkempfangstechnik

5 Funkempfänger-kompendium

Wollten Sie schon immer wissen, wie sich die klassische Funkempfängertechnik fortentwickelt hat? Wie funktionieren professionelle Funkempfänger heute und was können sie leisten? Welche Empfangssysteme und Techniken stehen heute zur Verfügung? Möchten Sie

auch ausgefallene Anwendungen von Empfängern kennenlernen und wissen, wie ein Software Defined Radio (Digitalempfänger) nun wirklich funktioniert und was der letzte Stand der entsprechenden Technik kann? In diesem Buch findet man die Antworten!

397 Seiten (geb.) • ISBN 978-3-89576-276-5
€ 49,00 • CHF 61,95

Theorie und Praxis mit Multisim

6 Grundschaltungen der Elektronik

Dieses Buch ist ein Nachschlagewerk über Elektronik mit praxisorientierten Fakten und ausführlichen Erklärungen. Der Autor hat selbst für komplexe Vorgänge oder Formeln praktische kurze Erklärungen und Näherungsrechnungen entwickelt, ohne die Darstellungen zu simplifizieren. Als Ausgangspunkt wurde das Simulationsprogramm Multisim gewählt, das zahlreiche Bauelemente und umfangreiche Messinstrumente zur Verfügung stellt.

360 Seiten (kart.) • ISBN 978-3-89576-286-4
€ 44,00 • CHF 54,95

Schnell und einfach
mit Arduino und Elektor-Shield

7 Mikrocontroller verstehen und anwenden

Mit diesem Buch erweitert der Anfänger auf dem Gebiet der Mikrocontroller, der Arduino-User bzw. -Enthusiast seine Mikrocontroller-Kenntnisse auf Grund eigener Erfahrungen und Erfolgserlebnisse und wird dazu noch ganz nebenbei in die Welt des Arduino und seiner Entwicklungsumgebung eingeführt.

392 Seiten (kart.) • ISBN 978-3-89576-296-3
€ 42,00 • CHF 52,95

Schaltungsprojekte für Profis

8 Arduino

Für den großen Erfolg der Arduino-Plattform lassen sich zwei Ursachen finden. Zum einen wird durch das fertige Board der Einstieg in die Hardware enorm erleichtert; der zweite Erfolgsfaktor ist die kostenlos verfügbare Programmieroberfläche. Unterstützt wird der Arduino-Anwender durch eine Fülle von Software-Bibliotheken. Die täglich wachsende Flut von Libraries stellt den Einsteiger vor erste Probleme. Nach einfachen Einführungsbeispielen ist der weitere Weg nicht mehr klar erkennbar, weil oft detaillierte Projektbeschreibungen fehlen. Hier setzt dieses Buch an. Systematisch werden Projekte vorgestellt, die in verschiedene Themengebiete einführen. Dabei wird neben den erforderlichen theoretischen Grundlagen stets größter Wert auf eine praxisorientierte Ausrichtung gelegt.

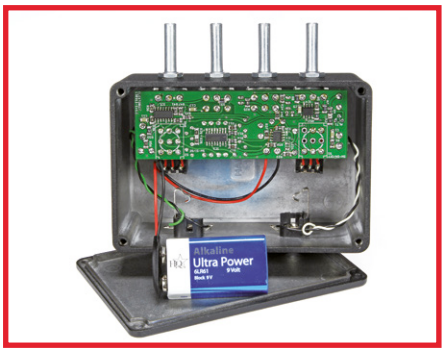
270 Seiten (kart.) • ISBN 978-3-89576-257-4
€ 39,80 • CHF 49,95

Weitere Informationen zu unseren Produkten sowie das gesamte Verlagssortiment finden Sie auf der Elektor-Website:

www.elektor.de

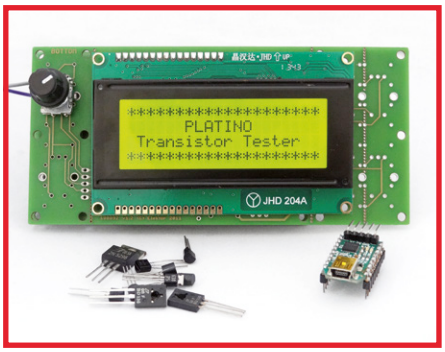
Elektor-Verlag GmbH
Süsterfeldstr. 25
52072 Aachen
Tel. +49 (0)241 88 909-0
Fax +49 (0)241 88 909-77
E-Mail: bestellung@elektor.de

Nächsten Monat in Elektor



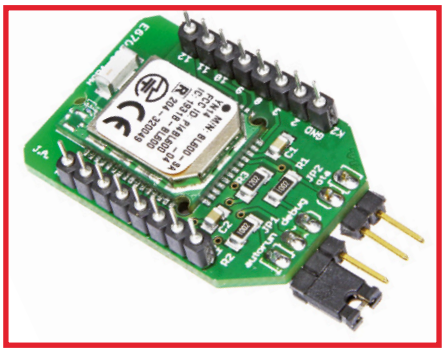
OTA-Overdrive

In der Musiker-Szene werden die Klänge elektrischer Gitarren häufig „veredelt“, indem die Signale spezielle Filter oder Verzerrer durchlaufen. Beim OTA-Overdrive sind selektierte Germaniumdioden und so genannte OTAs daran beteiligt, dem Sound eine individuelle Prägung zu verleihen. Der OTA-Overdrive (OTA = Operational Transconductance Amplifier) ist mit diskreten Komponenten aufgebaut.



Platino-Transistortester

Das zweite arbeitsplatztaugliche Gerät unserer Platino-Reihe, das wir bereits in dieser Ausgabe vorgestellt haben, war ein Funktionsgenerator. Im nächsten Monat setzen wir die Reihe mit einem vielseitigen Transistortester fort. Geprüft werden Transistoren fast jeden Typs, einschließlich JFETs und MOSFETs. Die Lage der Anschlüsse und die gemessenen Werte erscheinen auf einem vierzeiligen LC-Display.



BoB BL600

Breakout-Boards sind modulartige Platinen, die für den Einsatz in größeren Systemen oder auf Steckbrettern konzipiert sind. Elektors jüngster Spross ist das BoB BL600, eine Platine mit dem Bluetooth-Modul BL600-SA von Laird Technologies. Dieses Bluetooth-Modul arbeitet mit dem Bluetooth-Standard 4.0 Low Energy, sein auffälligstes Merkmal ist der ungewöhnlich niedrige Energiebedarf.

Änderungen vorbehalten. **Elektor März 2015 erscheint am 25. Februar 2015.** **Verkaufsstellen findet man unter www.pressekaufen.de.**

Lesen Sie 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.
- **Exklusiv:** Sie erhalten alle 2 Wochen per E-Mail ein neues Extra-Schaltungsprojekt (frisch aus dem Elektor-Labor).
- **Exklusiv:** Wir gewähren Ihnen bei jeder Online-Bestellung 10% Rabatt auf unsere Shop-Produkte – dauerhaft!
- **Exklusiv:** Der Online-Zugang zum 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:** Als Dankeschön gibt es einen Elektor-Gutschein in Höhe von 25 € 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!

Sie wollen 2D-Multi-Touch & 3D-Gestenerkennung in einer PC-Peripherie vereinen?

Microchips 3DTouchPad ist die erste Entwicklungsplattform für 2D/3D-Eingabeerfassung

Microchip präsentiert mit dem 3DTouchPad ein sofort einsatzfähiges Entwicklungskit und Referenzdesign, das 2D-Tracking von bis zu zehn Fingern und 3D-Gestenerkennung unterstützt. Damit lässt sich eine fortschrittliche Eingabeerfassung für PC-Peripherie und andere Anwendungen schnell entwickeln.

Auf der Basis von Microchips GestIC®-Technologie bietet die robuste Gestenerkennung des 3DTouchPad einen Erfassungsbereich von bis zu 10 cm. Die reaktionsschnelle 2D Projected-Capacitive Multi-Touch-Sensorik unterstützt bis zu 10 Berührungspunkte und Oberflächengesten mit mehreren Fingern.

Der neue integrierte High-Voltage Capacitive-Touchscreen-Leitungstreiber MTCH65X von Microchip sorgt für robuste Projected-Capacitive Touch-Performance und unterstützt größere Sensoren sowie dickere Oberflächenmaterialien, indem der Signal-Rauschabstand (SNR) erhöht wird.

Als Plug&Play PC-Peripherie wird das 3DTouchPad über ein USB-Kabel an den PC angeschlossen. Hinzu kommen eine kostenlose grafische Benutzeroberfläche (GUI), ein Softwareentwicklungskit (SDK) und ein Application Programming Interface (API). Sofort einsatzfähige, treiberlose Funktionen erhöhen das Benutzererlebnis für Windows® 7/8.X und MacOS®.

WESENTLICHE LEISTUNGSMERKMALE:

- 3DTouchPad-Entwicklungskit für 2D/3D-Eingabeerfassung: DM160225
- MTCH65X-Leitungstreiber für 2D Projected-Capacitive Touchscreens
- GestIC®-Technik für fortschrittliche 3D-Gestenerkennung

Weitere Informationen: www.microchip.com/get/eu3DTouchPad

MICROCHIP

Microcontrollers • Digital Signal Controllers • Analog • Memory • Wireless