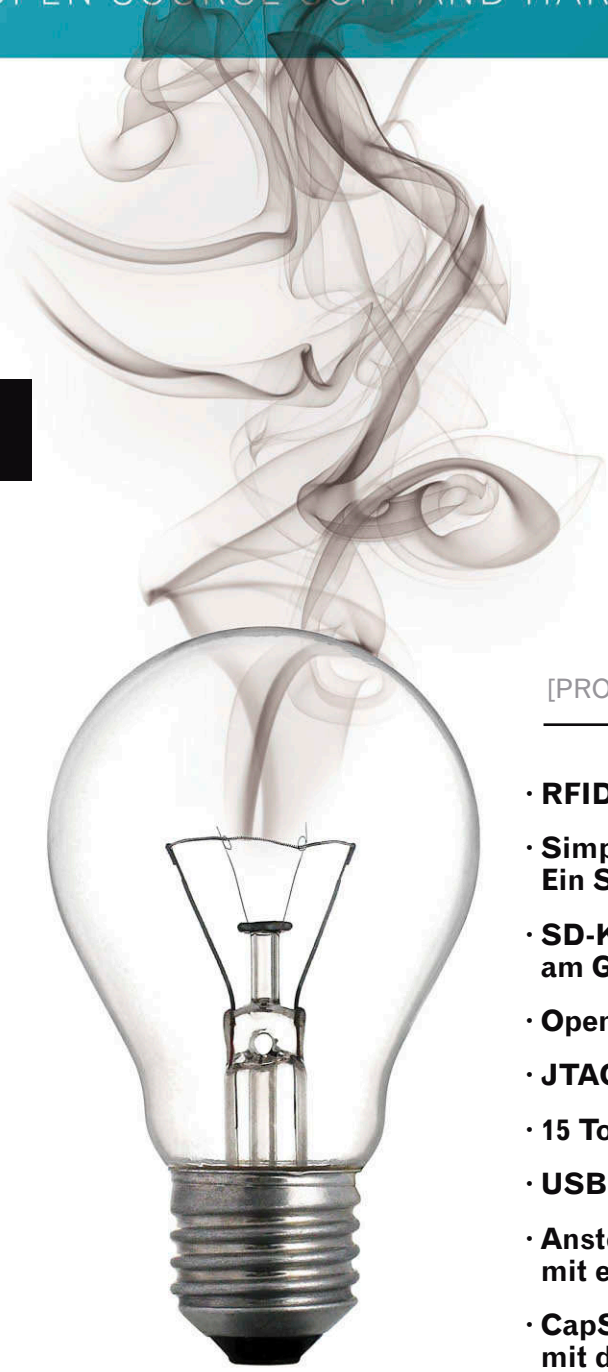
embedded-projects.net
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS



[EDITORAL]

NO IDEAS ?

Neue Projekte zum nachbauen und weiterentwickeln

[PROJECTS]

- **RFID Türmodul**
- **SimpleCPU Ein Selbstbau-Mini-Prozessor**
- **SD-Kartenleser am Grasshopper (AVR32)**
- **Open-Source Mikrocontroller Kurs**
- **JTAGICE-mkII-Klon**
- **15 Ton Melodiegenerator**
- **USB-Stacks für Microcontroller**
- **Ansteuerung Touchscreen mit einem AVR**
- **CapSense-Anwendungen mit dem CapToolKit**

WELCOME

Ausgabe 2 von Embedded-Projects Journal

Wir sind eine Eintagsfliege !?!

Mit dieser zweiten Ausgabe ist der Beweis erbracht: Wir sind eine Eintagsfliege!

Die Eintagsfliege (gr. Ephemeroptera) blickt – wie ihr Name bereits vermuten lässt - einer sehr kurzen Lebenserwartung entgegen. Im besten Fall hat sie laut Volksmund insgesamt eine Woche zu leben, da sie beflügelt bereits nach ein paar Stunden oder spätestens nach einem Tag stirbt. Doch nach dem Motto, alles hat Sinn – ist es bei der Eintagsfliege ganz genauso. Mutter Natur sorgt wie immer dafür, dass ein Gleichgewicht zwischen all der Vielfalt an Leben auf der Welt herrscht und die Besten und Stärksten überleben. Auch die Eintagsfliege, die es schon seit über 200 Millionen Jahren gibt, gehört dazu:

Die kleine so bekannte und doch anscheinend so unbekannte Eintagsfliege führt ein vollständig anderes Leben, wie so oft angenommen. Betrachtet man den Werdegang einer Ephemeroptera, so beginnt er mit dem Schlüpfen als Larve aus dem Ei. Dies geschieht in einem Zeitraum von 10 Tagen bis zu einigen Monaten. Dass auch wir bereits erfolgreich geschlüpft sind, haben wir mit der ersten Ausgabe im Juni bewiesen. Im Anschluss daran beginnt die Teenager-Zeit der kleinen Larven - welche auch ganz elegant als Nymphen bezeichnet werden. In dieser Zeit gibt es verschiedene Arten von Eintagsfliegen, die sich entweder schwimmend, grabend oder kriechend fortbewegen. Nur durch diese Vielfalt kann gesichert werden, dass das Gleichgewicht in der Natur erhalten bleibt. So hoffen auch wir, ein gutes Spektrum an Vielfalt anbieten zu können. Unglaubliche ein bis vier Jahre – für Insekten geradezu eine Ewigkeit - kann diese Hauptphase des Lebens andauern. In dieser Zeit häutet sich die Nymphe bis zu 50 mal, denn durch das Wachstum wird regelmäßig ihr Gewand zu eng und auch wir können stolz verkünden, unsere erste Häutung vollzogen zu haben, da die Seitenanzahl der ersten Ausgabe bereits jetzt gesprengt wurde.
Erst zum Lebensende hin entsteht durch Metamorphose die uns bekannte Eintagsfliege.

Die Perfektion der Ephemeroptera ist soweit ausgeprägt, dass sie sich instinktiv kurz vor ihrer Verwandlung gegen Flussrichtung des Gewässers bewegt. Und jeder, der etwas großes erreichen will, muss schon seit Urzeiten gegen den Strom „schwimmen".

Durch die Metamorphose in ein Wesen mit Flügeln ist es der kleinen Fliege endlich möglich, zum Olymp der Freiheit aufzusteigen. Tausende Tiere starten synchron mit ihrer Verwandlung und bilden gemeinsam mit vielen anderen Fliegen Scharen und ganze Wolken, um ihren Hochzeitstanz vorzuführen und ihre Weibchen zu begatten. Sie geben in diesem Akt der Vermehrung ihren genetischen Bauplan mit ALLEN Informationen verpackt in einem kleinem Ei weiter an die Nachwelt, so wie wir an Euch mit unserem Heft - frei, offen und einfach zugänglich.

Ist die Energie verbraucht, fallen die Eintagsfliegen tot zu Boden. Und genau dieser Prozess, die Vollendung des Lebens, der Beginn einer neuen Generation, ist es, der den meisten Menschen bekannt sein dürfte.

Dass wir uns nun schwimmend, grabend oder kriechend der nächsten Ausgabe entgegen bewegen können, verdanken wir vor allem Lutz Vollbracht, dem Geschäftsführer der Firma „IBV - Echtzeit- und Embedded GmbH & Co. KG" der als Hauptsponsor die Hälfte der Kosten für Druck und Porto übernommen hat!
An dieser Stelle bedanken wir uns ebenso bei den anderen Sponsoren.

Wir freuen uns auf die nächste Ausgabe und wünschen viel Spaß beim Lesen.

Benedikt Sauter
sauter@embedded-projects.net

und die stetig wachsende Community rund um dieses Open-Source Projekt Zeitschrift

[1] http://de.wikipedia.org/wiki/Eintagsfliegen
[2] http://de.wikipedia.org/wiki/OpenSource
[3] http://www.ibv-augsburg.net/

shop.embedded-projects.net

THE ONLINESHOP - HARDWARE FOR YOUR PROJECTS

ARM Boards

ARM USB Debugger (ARM-USB-Tiny)

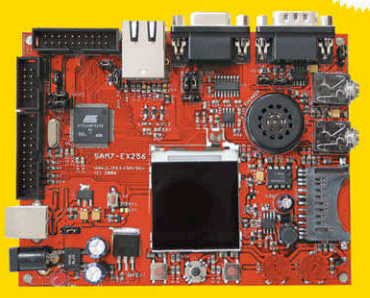
ARM7/ARM9/Cortex-M3 und XScale Debuginterface für OpenOCD und CrossWorks

ARM USB Debugger
44,90 €
embedded-projects SHOP

AT91SAM7X256+TFT+Ethernet (SAM7-EX256)

Bestückt mit einem 32 Bit ARM-Mikrocontroller mit 256 kB Flash, 64 kB RAM, 55 MHz, Ethernet 10/100, USB 2.0, RS232, CAN, MMC/SD-Card Slot und TFT-Display

AT91 SAM7X256 TFT+ Ethernet
119,90 €
embedded-projects SHOP



MSP430 Boards

MSP430F1611 Adapterplatine (MSP430-H1611)

MSP430F1611 mit 48K Bytes Programm Flash, 256 Bytes Daten Flash, 10K Bytes RAM

MSP430F1611 Adapterplatine
24,90 €
embedded-projects SHOP

MSP430 USB JTAG Adapter (MSP430-JTAG-TINY)

Programmierung/ Debugging für alle MSP430Fxxx Flash-Microcontroller

MSP430 USB JTAG Adapter
64,90 €
embedded-projects SHOP

AVR Boards

Atmel AVRISP mkII (USB)

Original AVRISP mkII In-System Programmer von Atmel.

Atmel AVRISP mkII
39,90 €
embedded-projects SHOP

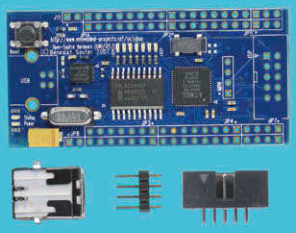
AVR Starterkit (inkl. USBprog, Netzteil und ATMega8)

AVR-Starterkit bestehend aus USBprog, AVR-Starterplatine, ATMega8 und Steckernetzteil.

SET AVR Starterkit
59,00 €
embedded-projects SHOP

Octopus USB

Octopus bietet viele bekannte Schnittstellen aus der Mikrocontrollerwelt über ein einfaches USB Gerät an.

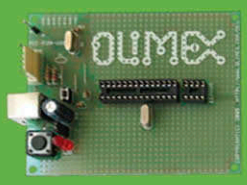


Octopus USB
39,00 €
embedded-projects SHOP

PIC Boards

PIC Entwicklungsplatine (PIC-P28-USB)

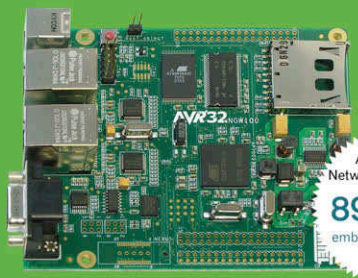
PIC Mikrocontroller Entwicklungsboard für 28-polige ICs + USB RS232.



PIC Entwicklungsplatine (PIC-P28-USB)
24,90 €
embedded-projects SHOP

AVR32 Boards

ATNGW100 Network Gateway Kit

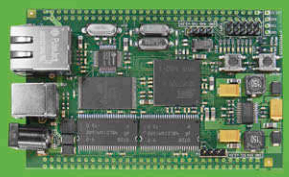


ATNGW100 Network Gateway Kit
89,90 €
embedded-projects SHOP

Grasshopper AVR32-Board

*Open-Source-Version (ohne CD+Kabel)

freie Plattform für die AVR32 Entwicklung



Grasshopper AVR32-Board*
ab 85,00 €
embedded-projects SHOP

Jetzt schneller zum Shop:
www.eproo.net

- großes Sortiment an Evaluations- und Testboards
- bekannte Open Source Projekte
- übernommener Artikelbestand von www. mikrocontroller.net
- faire Preise
- Produkte direkt vom Hersteller
- bequeme Zahlungabwicklung und schneller Versand

Jetzt Neu:
Versand weltweit
3,95 Euro

RFID Türmodul

Stefan Seegel <post@seegel-systeme.de>

ARTIKEL WETTBEWERB PLATZ 4

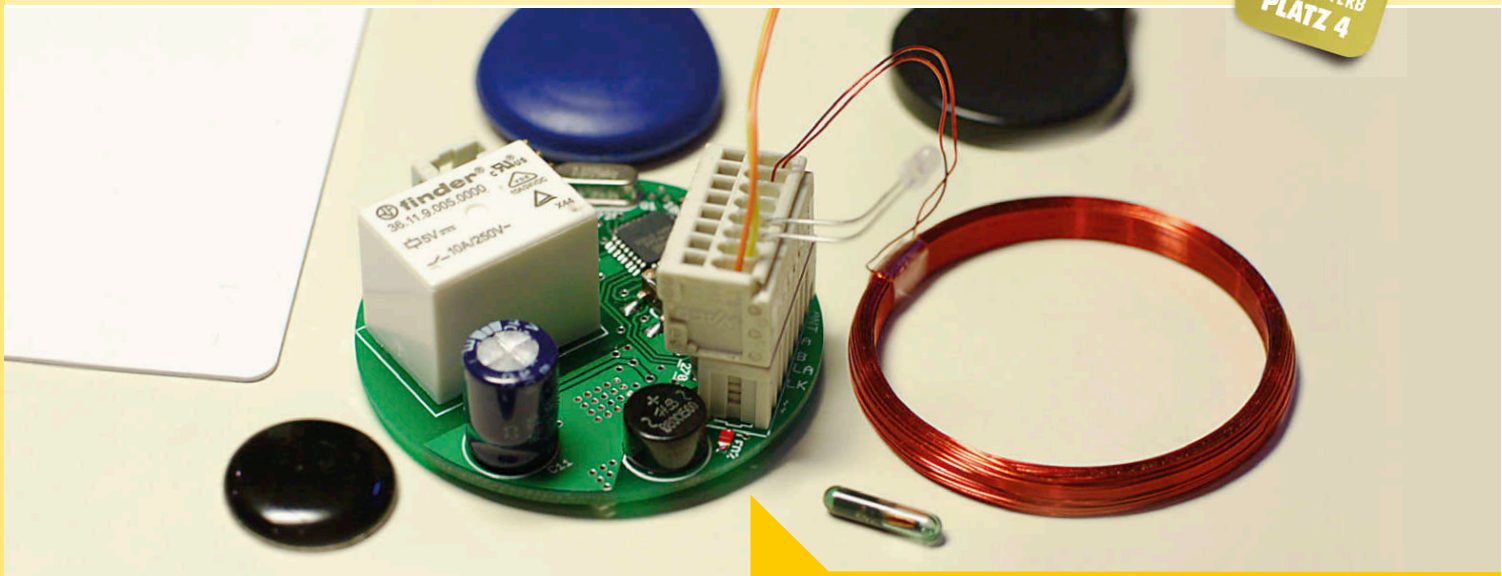


Abbildung 1: Aufgebauter RFID-Reader mit diversen Tags

Einleitung

RFID (Radio Frequency Identification) ist ein Verfahren, um draht- und berührungslos z. B. Gegenstände oder Tiere zu identifizieren oder zu orten. Eine Variante der RFID Technik ist die LF-Ausführung (Low Frequency). Diese ist sehr preisgünstig und die LF-Transponder benötigen keine eigene Stromversorgung, da diese über das Feld des Lesers mit Energie versorgt werden. Daraus ergibt sich ein breites Anwendungsgebiet, z. B. Zutrittssysteme, Zeiterfassung, Logistik, u. v. m.. Im folgenden Artikel wird ein RFID Leser für passive 125 kHz Transponder vorgestellt.

Projekt

Das RFID Türmodul (Abb. 1) eignet sich aufgrund seiner Abmessung (Durchmesser 52 mm) für den direkten Einbau in eine Unterputzdose. Die Unterputzdose sollte am besten an der inneren Seite der Tür angebracht werden. An der Außenseite kann z. B. eine Acrylplatte, welche die Antenne und optional eine (Duo-) LED beherbergt, befestigt werden. Das Modul kann z. B. direkt an einem Klingeltrafo 8 V~ betrieben werden, welcher auch gleich einen elektrischen Türöffner versorgen kann. Ein Relais um den elektrischen Türöffner zu steuern befindet sich mit auf der Platine. Optional kann die Platine mit einem RS485 IC bestückt werden, um z. B. eine Vernetzung oder erweiterte Steuerung zu ermöglichen. Weiterhin ist auf der Platine Platz für ein I2C EEPROM, um auch eine größere Anzahl an Transpondern speichern zu können. Schaltplan und Software stehen unter [1] zur Verfügung.

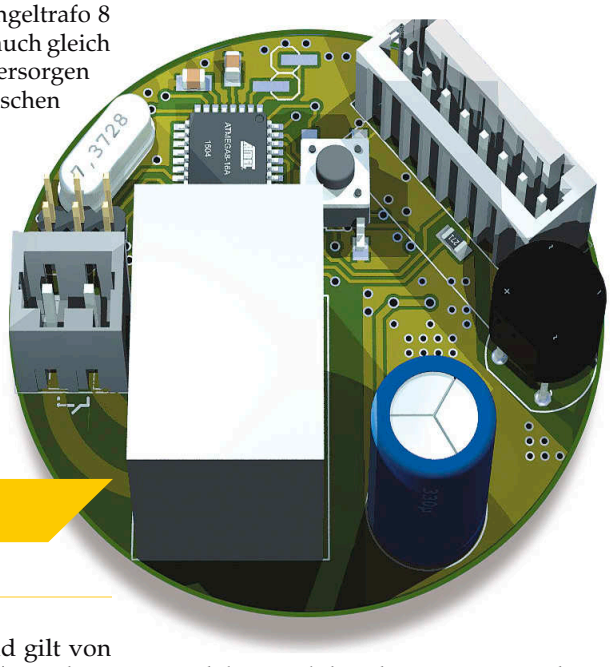


Abb. 2: RFID Türmodul Oberseite

Ausstattung im Überblick

- µC ATMega8
- Brückengleichrichter und Spannungsregler 78M05 on board
- RFID Reader IC EM4095
- Wagoklemme für den Anschluss von Stromversorgung, RFID Antenne, (Duo-) LED und Datenleitung (RS485)
- Wagoklemme für den Relaiskontakt, um z. B. einen Türöffner zu schalten (Relais on board)
- Kurzhubtaster, z. B. zum löschen von EEPROM
- 2 Pins des µC (PD3, PD4) mit Stiftleiste abgreifbar, um z. B. zusammen mit dem ISP Steckverbinder ein RFM12 Funkmodul anbinden zu können

Funktionsbeschreibung

Die Standardfirmware (Downloadlink unten) überprüft bei Erkennung eines Tags zunächst, ob bereits ein Tag im EEPROM abgespeichert ist. Falls nicht (EEPROM leer / Erstinbetriebnahme) wird dieser Tag im EEPROM abgelegt und gilt von nun an als „Administrator-Tag“. Wird ein gültiger Tag erkannt, zieht das Relais für ca. 2 s an, bevor es wieder abfällt um z. B. einen elektrischen Türöffner anzusteuern. Handelt es sich bei diesem Tag um den Administrator Tag, können weitere gültige Tags angelernt werden, wenn diese an die Antenne geführt werden solange das Relais

angezogen ist. Wird der Kurzhubtaster auf der Platine länger als zwei Sekunden gedrückt, werden alle gespeicherten Tags gelöscht. Der nach einem Löschvorgang als nächstes erkannte Tag wird wieder als Administratortag gespeichert. Über die RS485 Schnittstelle werden beim Lesen eines Tags dessen Daten nach folgendem Format ausgegeben: <STX>R<+/-><5 Bytes Tag hexadezimal><EOT>. Das ‚+' oder ‚-' gibt an, ob der Tag im EEPROM gefunden wurde und somit das Relais aktiviert wurde.

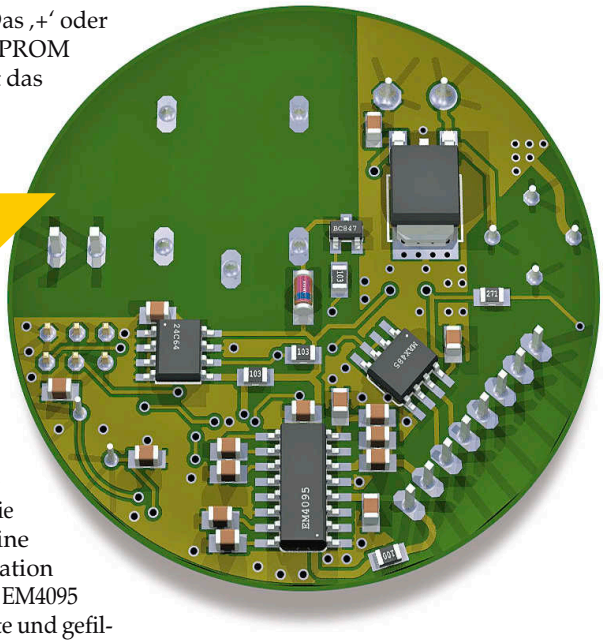
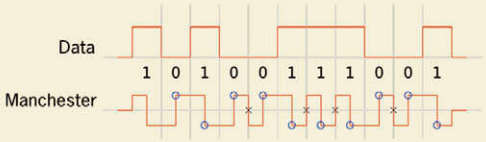


Abb. 3: RFID Türmodul Unterseite

Schaltungsbeschreibung

Der RFID IC EM4095 [2] gibt kontinuierlich ein 125 kHz Signal auf die Spulenantenne aus. Wird ein Tag in das Feld der Antenne gebracht, versorgt sich der auszulesende Tag darüber mit Energie. Gleichzeitig sendet der Tag seine Daten in dem er dem Leser unterschiedlich stark Energie entzieht, so dass eine Amplitudenmodulation entsteht. Das RFID IC EM4095 gibt das demodulierte und gefilterte Signal auf einen interruptfähigen Eingang des Mega8. Bei diesem Signal handelt es sich um ein Manchester-codiertes (siehe Abbildung 4) Signal mit einer Bitrate von einem 64tel der Trägerfrequenz. Ausgehend von 125 kHz Trägerfrequenz sind das knapp 2000 bit/s. Bei jedem Pegelwechsel des manchester-codierten Signals wird nun eine Interruptroutine im μC aufgerufen.

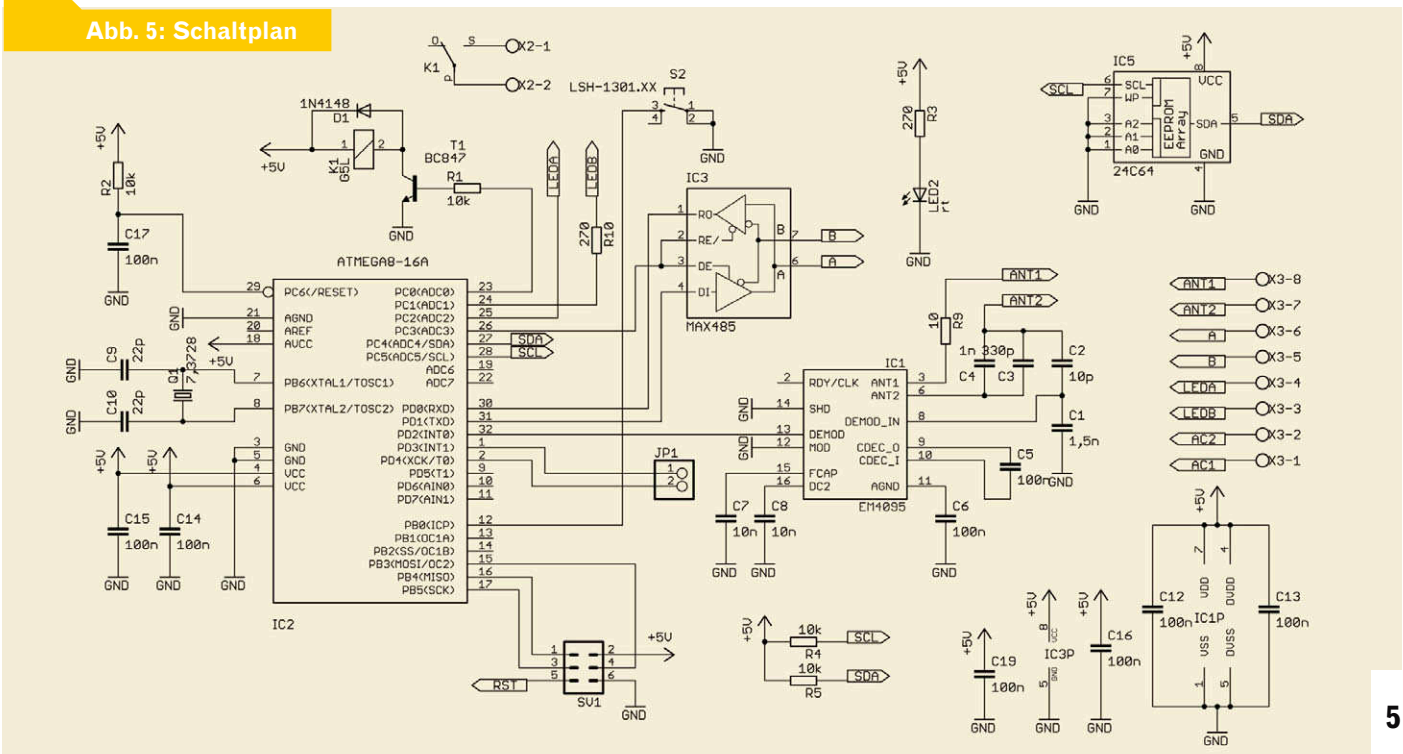
Abb. 4: Manchester-Codierung



Softwarebeschreibung

Trotz des kleinen μC wurde als Programmiersprache C++ verwendet, da sich damit eine sehr kompakte und übersichtliche Struktur erreichen lässt. Kernstück der Firmware bildet die Klasse „Rfid". Bei jedem Aufruf der INT0-Interruptroutine durch das Signal des RFID-ICs wird die vergangene Zeit zum vorherigen Flankenwechsel überprüft; ist die Zeit kürzer als 3/4 der Bitzeit, wird die Flanke ausgeblendet und die Routine verlassen. Andernfalls wird eine 16-bit Variable um ein Bit nach links geschoben und, falls der Pegel auf 0 liegt, mit 1 verodert. Nach jedem eingelesenen Bit wird nun dieses FIFO-Register auf das Vorhandensein eines Tagheaders (9x 1-Bits) überprüft, und gegebenfalls ein Bitzähler auf 0 gesetzt. Ist ein ganzes Byte voll, wird dieses der RFID Klasse übergeben. Nachdem 64 Bit eingelesen wurden, werden zunächst die Spalten- und Zeilenparitätsbits überprüft. Sind alle Paritätsbits korrekt, werden selbige ausgeblendet, so dass 40 „Nettodatenbits" verbleiben. Diese Transponderdaten werden nun der Klasse „Tagmanager" übergeben. Dort werden aus dem EEPROM immer 5 Byte ausgelesen und mit den dekodierten Daten verglichen, solange bis die Daten übereinstimmen oder das Speicherende (0xFFFFFFFFFF) erreicht wurde. Handelt es sich beim dekodierten Transponder um den ersten Speicherplatz im EEPROM, wird noch ein Administratorflag gesetzt, so dass weitere Transponder hinzugefügt werden können. Das 64 kbit große I2C-EEPROM kann bis zu 1638 Transponder speichern.

Abb. 5: Schaltplan



Datenaufbau der Transponder

Die verwendeten RFID-Transponder enthalten eine eindeutige, 64-Bit lange Bitfolge, die folgendermaßen aufgebaut ist: die ersten neun Bits sind alle auf 1 gesetzt und stellen den Header dar. Dieses Bitmuster kann aufgrund der eingeschobenen Paritätsbits nirgendwo sonst in den Daten auftauchen, und kann so zur Synchronisation auf den Datenanfang benutzt werden. Nach dem Header folgen 10 x 4 Datenbits, jeweils gefolgt von einem Paritätsbit das aus den jeweils letzten 4 Bits errechnet wird (Zeilenparität, ZP). Nach diesen zehn Gruppen schließen sich noch 4 weitere Paritätsbits an, welche aus den 4 Spalten der Datenbits errechnet werden (Spaltenparitäten, SP). Das letzte Bit ist das Stopbit und ist fest auf 0 gesetzt.

HEADER (9 x 1)					
	D0	D1	D2	D3	ZP
	D4	D5	D6	D7	ZP
	D8	D9	D10	D11	ZP
	D12	D13	D14	D15	ZP
	D16	D17	D18	D19	ZP
	D20	D21	D22	D23	ZP
	D24	D25	D26	D27	ZP
	D28	D29	D30	D31	ZP
	D32	D33	D34	D35	ZP
	D36	D37	D38	D39	ZP
	SP	SP	SP	SP	STOP

Die Antenne

Eine passende Leseantenne für das Modul kann sehr einfach selbst hergestellt werden. Man nehme z. B. 0,2 mm Kupferlackdraht, von dem ca. 100 Windungen auf einen runden Körper mit ca. 45 mm Durchmesser gewickelt werden. Die Induktivität der Spule lässt sich dann mit Hilfe eines L-Meters oder alternativ mit einem Funktionsgenerator und einem Oszilloskop herausfinden. Wer keine Antenne selber wickeln möchte, kann natürlich auch auf fertige Spulen zurückgreifen [3]. In jedem Fall müssen in der Schaltung C1-C4 und R9 diese an die Antenne angepasst werden. C3 und C4 bilden zusammen mit der Antennenspule einen Serienschwingkreis, so dass sich eine Resonanzfrequenz von ca. 125 kHz ergibt. C1 und C2 dienen als kapazitiver Spannungsteiler, um das relativ große Antennensignal auf ein für das IC verträgliche Maß zu verkleinern. R9 dient dazu, bei sehr niederohmigen Antennen, die Güte künstlich zu verkleinern. Auf der Herstellerseite des RFID ICs findet sich eine Excel-Tabelle, die die Berechnung der Bauteile etwas vereinfacht. Die Antenne kann in gewissen Grenzen mit einem Kabel vom Modul abgesetzt werden, ohne dass sich die Resonanzfrequenz dramatisch ändert. In einem Versuch mit 70 cm 0,5 mm² Doppellitze zwischen Antenne und Modul ging die Resonanzfrequenz um ca. 1 kHz nach unten.

Links

1. Artikel, Schaltplan und Software bei mikrocontroller.net http://www.mikrocontroller.net/articles/RFID_T%C3%BCrmodul
2. Datenblätter zum RFID IC 4095 und den Transpondern http://www.emmicroelectronic.com
3. Menting Mikroelektrik GmbH & Co KG http://www.spulen.com
4. Diskussion zum Artikel bei mikrocontroller.net http://www.mikrocontroller.net/topic/106299

Abb. 6: Bestückungsmodul unten

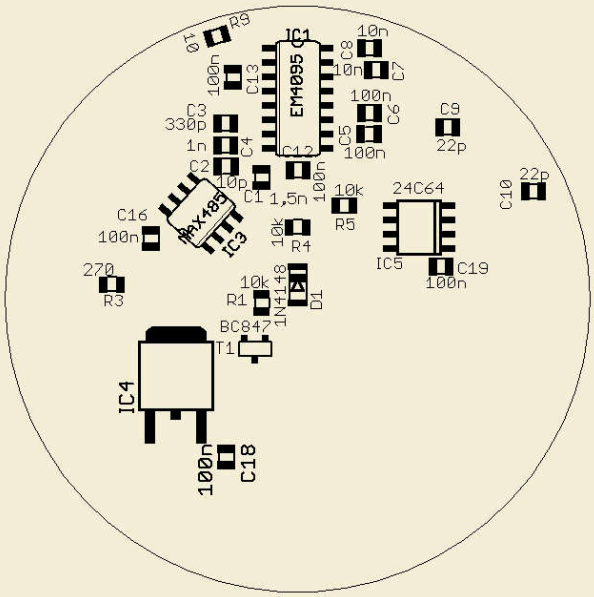
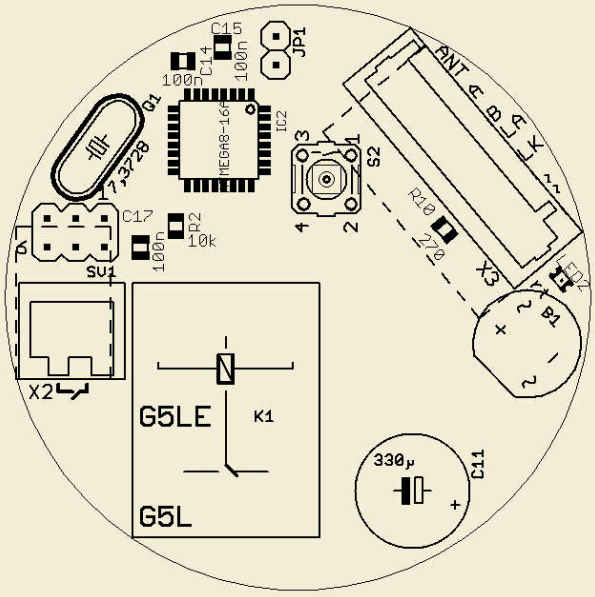


Abb. 7: Bestückungsmodul oben



SimpleCPU

Ein Selbstbau-Mini-Prozessor aus der Bastelstube

Michael Schäferling <m.schaeferling@t-online.de> | Benedikt Sauter <sauter@ixbat.de>

Einleitung

Einmal selbst einen kleinen Prozessor bauen, welcher schließlich auch in Form realer Hardware existieren soll und anhand dessen der Aufbau und die Funktionsweise eines minimalen Rechner-Systems veranschaulicht wird – dies war die initiale Idee zu diesem kleinen Projekt.

Zunächst soll der Leser kurz in den Aufbau und die Funktionsweise eines minimalen Rechnersystems eingeführt werden. Hierbei werden die benötigten Komponenten vorgestellt und deren entsprechende Realisierung in unserem Projekt. Im zweiten Schritt soll für unser System eine Plattform in Form eines Demo-Boards geschaffen werden. Das hieraus resultierende Design sollte übersichtlich und einfach aufgebaut sein, damit es mit möglichst wenig Aufwand und auch günstig nachgebaut werden kann.

Theorie: Was wird benötigt?

CPUs verrichten seit jeher ihre Arbeit als zentraler Bestandteil eines jeden PC, wobei die Leistungsfähigkeit eines Systems meist am (bzw. an den) verwendeten Prozessor(en) gemessen wird. Ganz allgemein ist eine CPU ein Bauteil, welches Programme ausführen kann. Um ein Programm ausführen zu können, werden von der CPU Informationen in Form von Daten und den darauf auszuführenden Befehlen benötigt. Diese sind in einem Datenspeicher permanent oder temporär gespeichert, in einem PC ist dieser Speicher durch die Speicherhierarchie realisiert: Festplatte, Arbeitsspeicher und Cache liefern hier die benötigten Daten. Um vernünftig mit einem Computer arbeiten zu können, finden natürlich auch Ein- und Ausgabegeräte wie Tastatur, Maus und Monitor ihren Einsatz bei gängigen Systemen. Doch welche der soeben vorgestellten Komponenten werden für unser minimales System tatsächlich benötigt?

Der zentrale Bestandteil ist selbstverständlich eine CPU. Zudem benötigen wir etwas Speicher, um Befehle und Daten halten zu können. Man könnte, sofern das System lediglich ein Programm ausführen soll und während des Betriebs keine Benutzerinteraktion erfordert, auf Ein- und Ausgabegeräte verzichten. Voraussetzung hierfür ist, dass vor dem Start des Systems die nötigen Informationen im Speicher bereitstehen. Hierfür ist eine Schnittstelle vom PC als Debug- und Programmierschnittstelle direkt zum Speicher des neuen Prozessors hin wünschenswert, da somit zu Beginn die Befehle und Daten geschrieben und nach Beendigung des Programms, die Ergebnisse aus dem Speicher gelesen werden können.

Umsetzung

Ein Ziel unseres Projektes ist es, einen Einblick in die Technik zu erfahren, die hinter den einzelnen Komponenten steckt. Es bietet sich also an, keine fertige CPU (wie z.B. einen AVR, ARM & Co.), sondern eine freie, offene (und womöglich in VHDL beschriebene) CPU zu verwenden, die eventuell auch noch erweitert werden kann. Für ein solches Vorhaben bietet sich konfigurierbare Logik als „Träger“ der CPU an. Falls der Leser noch keine Erfahrung mit CPLDs oder FPGAs sammeln konnte, möchten wir an dieser Stelle auf den Artikel „XSVF Player with USBprog“ der letzten Ausgabe dieses Journals verweisen. Auf die Frage, welche CPU genau wir denn nun in unserem Design arbeiten lassen, wollen wir in einem späteren Abschnitt dieses Artikels eingehen.

Abb. 1: CPLD Baustein – PLCC-Gehäuse

Ein weiteres Anliegen beim Design des Boards ist es, dass die Hardware mit wenig Aufwand selbst herzustellen ist. Aus diesem Grund entschieden wir uns für die Verwendung eines XC9572-CPLDs von Xilinx, da diese Bausteine in PLCC-Bauform erhältlich sind.

Diese ermöglicht einfaches Löten der IC-Fassung und leichtes Tauschen des CPLDs, falls doch mal etwas schief gehen sollte. Weiterhin entschieden wir uns hier für die 5Volt-Variante, da dies die Stromversorgung und die Interaktion mit dem verwendeten RAM-Baustein vereinfacht. Preislich schlägt diese Komponente mit etwa fünf Euro zu Buche.

Somit sind wir schon an der zweiten wichtigen Komponente angelangt, dem Speicher. Für Designs dieser Größe ist SRAM eine gute Wahl, da die Ansteuerung dieser Speicherart mit sehr wenig Aufwand verbunden ist. Jedoch liegt der Preis hier im Vergleich zu anderen Speichervarianten wesentlich höher. Für unsere Zwecke genügen jedoch 32KByte RAM, welches zur Zeit in DIL-Bauform etwas knapp über einen Euro kostet.

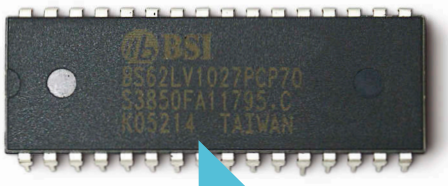


Abbildung 2: SRAM - DIP-Gehäuse

Die Wahl der CPU

Nachdem wir nun wussten, dass wir eine CPU bauen möchten, die auch noch in einen CPLD passt, machten wir uns zunächst einmal auf die Suche. Aufgrund der wenigen Logikgatter, die der gewählte CPLD zur Verfügung stellt (72 Makrozellen mit 1600 Gattern), ist die Auswahl, an bereits im Quellcode vorhandenen CPUs, leider nicht sehr groß. Wir stießen jedoch auf ein Projekt, das uns direkt begeistern konnte: Bei opencores.org wurde von Tim Böschke eine minimale CPU („MCPU – A minimal CPU for a CPLD“) veröffentlicht, die in einem CPLD mit 32 Makrozellen Platz findet [1]. Somit würde mit dieser Realisierung noch ausreichend Platz für den Speicher-Programmer bleiben, welcher ja ebenfalls im CPLD realisiert werden sollte. Des Weiteren ist die MCPU bereits ausführlich getestet (steht unter der GPL), ist sowohl als Verilog-, als auch als VHDL-Variante erhältlich und zudem ist das Projekt sehr gut dokumentiert.

Die Schaltung

Nachdem nun alle Anforderungen definiert sind, fehlt nur noch die passende Schaltung für den Versuchsaufbau. Der CPLD- und SRAM-Baustein bilden die zentralen Komponenten des Systems. Wie bereits erwähnt, verzichten wir nicht auf den Komfort einer Ein- und Ausgabeeinheit und spendieren daher dem Board drei Leuchtdioden für spätere Ausgaben. Getaktet wird unser Prozessor von einem einfachen Quarzoszillator, wobei dieser steckbar montiert wird, um den Prozessor schneller oder langsamer laufen lassen zu können. Die Maximalgeschwindigkeit wird später zum einem vom Design des Prozessors und zum anderen von der Leistungsfähigkeit des CPLDs abhängig sein.

Für das Programmieren des CPLD-Bausteins und das spätere Laden der Programme befindet sich ein 10-poliger Anschluss auf der Schaltung. Die Programmierung des CPLD-Bausteins kann auf diese Weise, wie im Artikel „XSVF Player with USBprog“ beschrieben, durchgeführt werden. Für das Laden von eigenen Programmen in die neue CPU wird es eine eigene Firmware für den freien Universalprogrammierer USBprog geben. Neben einem 10-poligen Anschluss befindet sich außerdem ein 34-poliger Wannenstecker auf dem Board. Hier können bequem alle Adress- und Datenleitungen abgegriffen werden. Der Taster S1 ist als Resetmöglichkeit für den Prozessor gedacht.

Bezeichnung	Wert
C1, C2, C3, C4, C5, C6, C8	100nF
C7	10uF
D1	Diode 1N4001
IC1	Xilinx XC9572-PC44 (5V)
IC2	Spannungsregler 78L05
LED1, LED2, LED3, LED4	Leuchtdioden
QG1	Quarz 1.000_MHz
R1, R2, R3, R4	270_Ohm
R5	470_Ohm
R6	10K_Ohm
S1	Taster
U1	SRAM 32Kx8 DIL28
X1	Hohlstecker-Buchse
X2	Wannenstecker 10-Pol.
X3	Wannenstecker 34-Pol.

Bestückungsliste „simpleCPU“

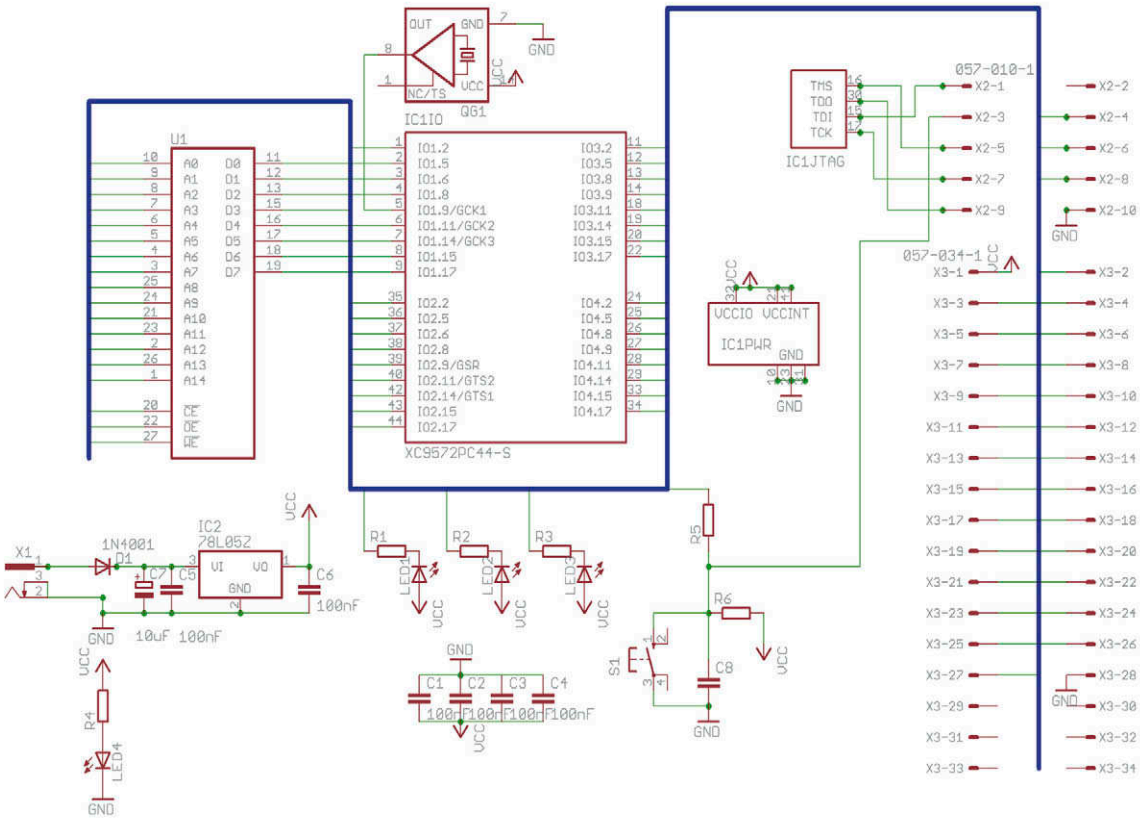


Abbildung 3: Schaltplan Prozessorplatine

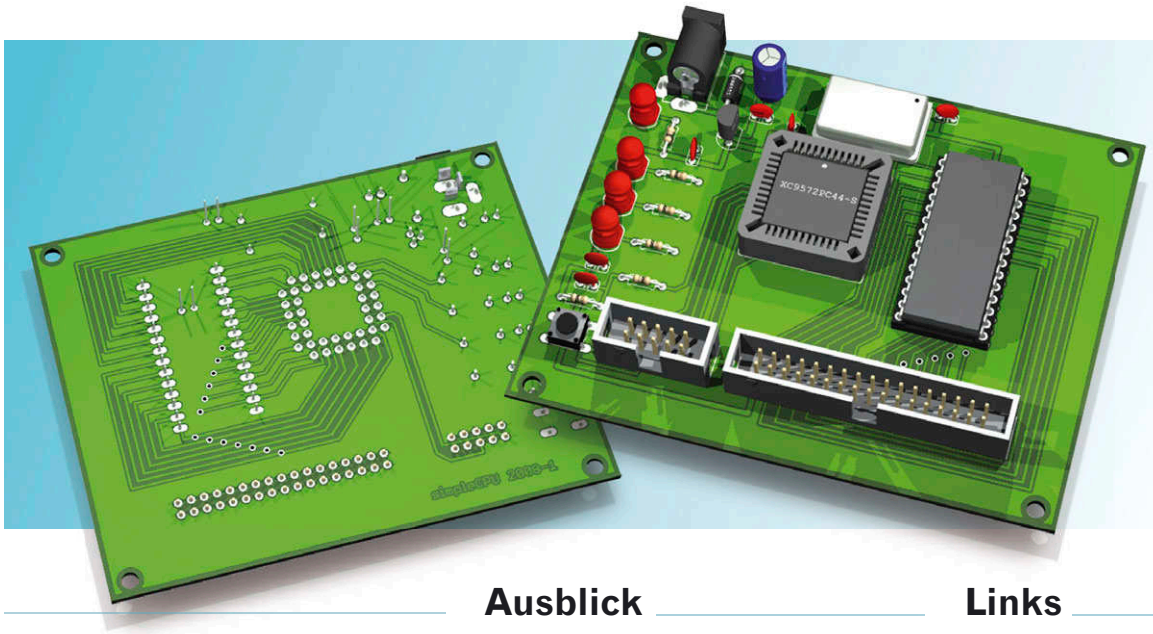


Abb. 4: Platine 3D Bilder Ober- und Unterseite

Zu guter Letzt fehlt noch die Stromversorgung der Schaltung, welche mit einem einfachen Spannungsregler vom Typ 7805 realisiert worden ist. Die vollendete Platine soll in etwa derart aussehen, wie in Abbildung 4 dargestellt.

Ausblick

Ziel dieses Artikels ist es, die Idee und alle wichtigen Komponenten des Projekts simpleCPU - die CPU, den Assembler und das Programmierinterface - vorzustellen. Bis zur nächsten Ausgabe gibt es für uns zwar noch eine Menge zu tun, aber wir sind noch ganz guter Dinge. Für Anregungen, Ideen oder sonstige Kommentare könnt Ihr euch gerne bei uns melden.

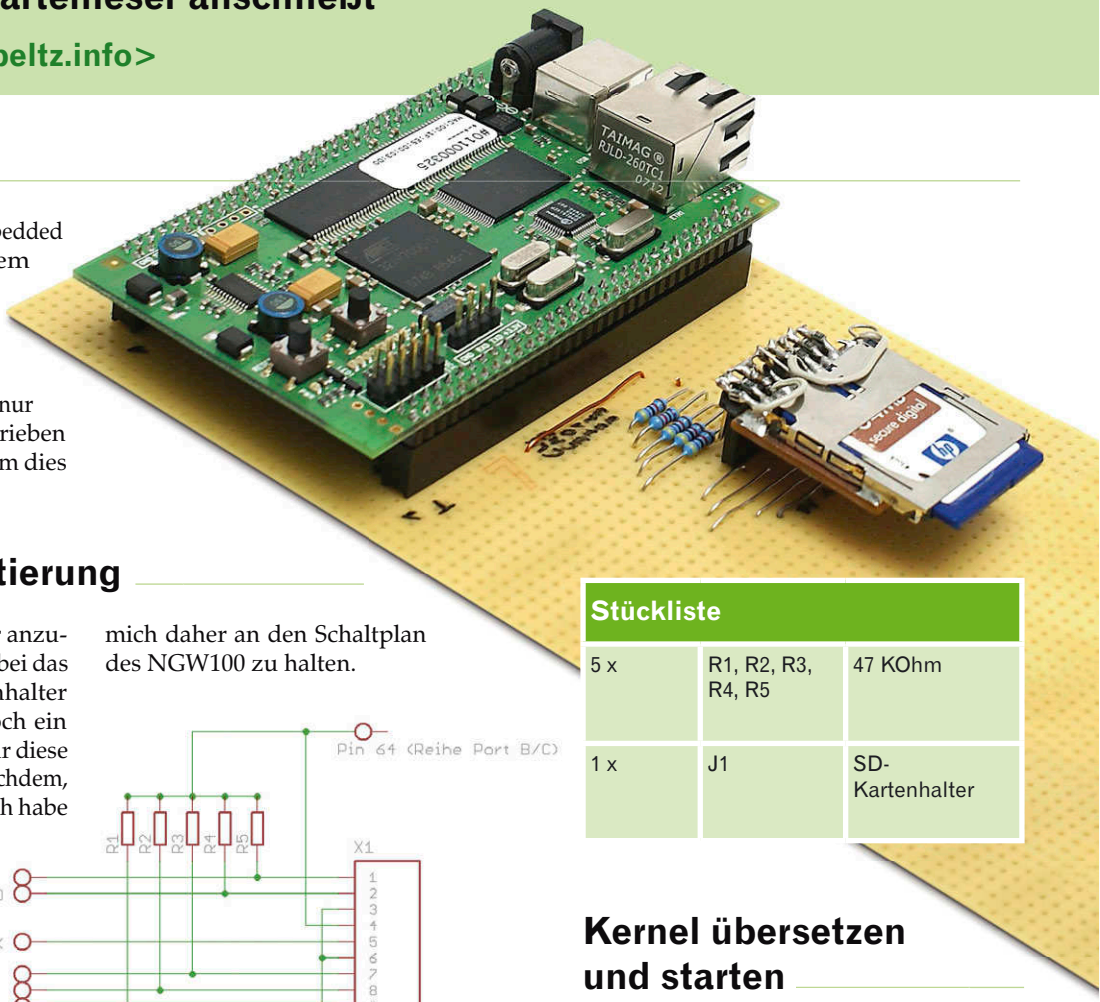
Links

1. MCPU – A minimal CPU for a CPLD
http://opencores.org/projects.cgi/web/mcpu/overview
2. USBprog Programmer
http://www.embedded-projects.net/usbprog

SD-Kartenleser am Grasshopper (AVR32)

Wie man einen SD-Kartenleser anschließt

Marcel Beltz <marcel@beltz.info>



Einleitung

Als ich in der letzten Ausgabe des „Embedded Projects Journal“ ein Bild mit dem dazugehörigen Artikel über das TFT-Display am Grasshopper (AVR32) gesehen habe, stand eindeutig fest, dass ich dies auch nachbauen muss. Da jedoch der Anschluss der SD-Karte nur im Foto sichtbar, jedoch nicht beschrieben war machte ich mich an die Arbeit um dies herauszusuchen und nachzubauen.

Hardware-Implementierung

Um eine SD-Karte am Grasshopper anzuschließen (siehe Abbildung 1), ist dabei das Schwierigste, sich einen SD-Kartenhalter zu besorgen. Daneben sind nur noch ein paar Widerstände nötig. Die Werte für diese Widerstände unterscheiden sich, je nachdem, wo man im Internet danach sucht. Ich habe mich daher an den Schaltplan des NGW100 zu halten.

Abb. 1: Schaltplan Grasshopper SD/MMC-Karte

Stückliste		
5 x	R1, R2, R3, R4, R5	47 KOhm
1 x	J1	SD-Kartenhalter

Treiber aktivieren

Die Aktivierung der notwendigen Treiber im Kernel ist kein Hexenwerk. In der folgenden Datei...

```
${GRASSHOPPER}/build_avr32/linux-2.6.24-icnova/arch/avr32/boards/
icnova/icnova_base.c
```

...muss nur die Struktur vor der „setup_board“ Funktion eingefügt werden:

```
static struct mci_platform_data __initdata mci0_data = {
        .detect_pin     = GPIO_PIN_NONE,
        .wp_pin         = GPIO_PIN_NONE,
};
```

Der „.detect_pin“ und „.wp_pin“ sind in dieser Variante nicht angeschlossen, und deshalb auf GPIO_PIN_NONE gelegt. Anschließend muss nur noch der Aufruf der Funktion

```
at32_add_device_mci(0, &mci0_data);
```

innerhalb der Hauptinitialisierungsfunktion icnova_init eingefügt werden. Jetzt ist alles notwendige im Kernel aktiviert, damit dieser eine Kommunikation mit der Karte im Kartenhalter aufbauen kann.

Kernel übersetzen und starten

Voraussetzungen zum Starten und für die ersten Experimente des Kernels sind in diesem Fall ein funktionierender TFTP- und NFS-Server zum Übertragen des Kernels und der Module. Im Internet findet man genug Quellen in denen beschrieben ist wie man sich solch Server installiert.

Nachdem die Quellen angepasst wurden, sind an der Kernel-Konfiguration nur noch zwei kleine Änderungen durchzuführen. Dazu muss in das Verzeichnis

```
${GRASSHOPPER}/build_avr32/
linux-2.6.23-icnova/
```

gewechselt und dort

```
make menuconfig
```

ausführt werden. An dieser Stelle wird der Support für VFAT aktiviert, was den Zugriff auf SD-Karten mit dem FAT-Dateisystem erlaubt. Die Schritte im Kernelmenü lauten wie folgt: „File Systems --->“, „DOS/FAT,NT Filesystems --->“ wechseln und das nötige Modul aktivieren (siehe Abbildung 2).

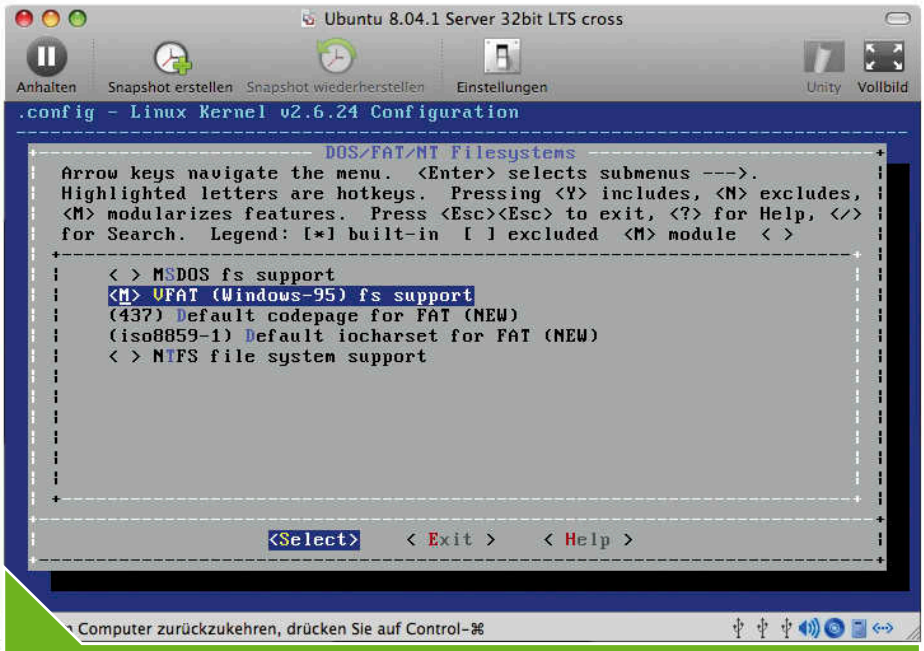


Abbildung 2: VFAT

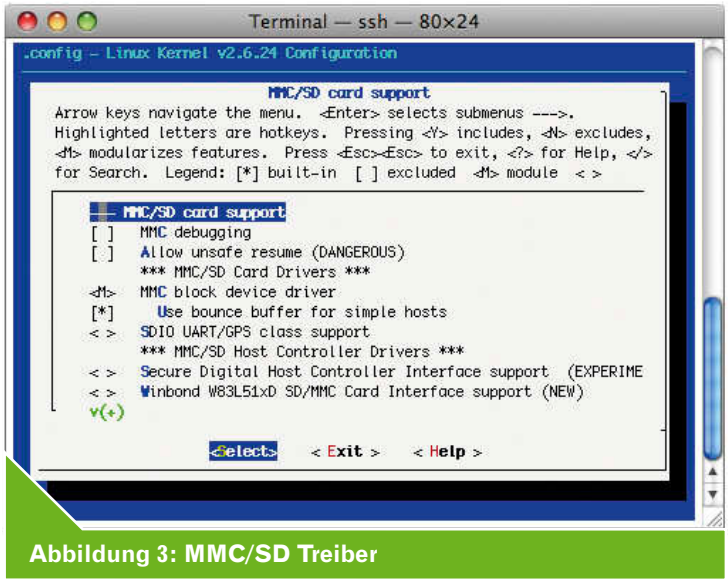


Abbildung 3: MMC/SD Treiber

Schließlich muss noch das Modul für die SD-Karten aktiviert werden: „Device Drivers --->“, „MMC/SD card support --->“ (siehe Abbildung 3).

Nun mit „Exit“ vollständig aus der Konfiguration herausgehen und vor dem Beenden speichern. Der Build-Vorgang wird mit

```
make ARCH=avr32 CROSS_COMPILE=avr32-linux- uImage
```

gestartet. Da alle als Module ausgewählt worden sind, bleibt nichts anderes übrig, als noch die Module zu kompilieren:

```
make ARCH=avr32 CROSS_COMPILE=avr32-linux- INSTALL_MOD_PATH=${NFS}/module
modules_install
```

Dabei werden die Module in das - durch das NFS - freigegebene Verzeichnis kopiert. Der Kernel muss noch in das TFTP-Verzeichnis kopiert werden, mit:

```
cp arch/avr32/boot/images/uImage /tftpboot
```

[IMPRESSUM]

Embedded Projects:
Dipl.-Inf. (FH) Benedikt Sauter
Auf dem Kreuz 20
D-86152 Augsburg

Telefon: +49 821 31946-23
Telefax: +49 821 31946-24
Mail: journal@embedded-projects.net

Anzeigemöglichkeiten
und Preisliste auf Anfrage
via Mail.

Herausgeber:
Benedikt Sauter
Gestaltung/Layout/Satz:
Das-Medienkollektiv.de
Artikelfotos:
Claudia Lehndorfer

Veröffentlichung: 4x / Jahr
Ausgabeformate: PDF / Print
Auflage Print: 2500 Stk.
Einzelverkaufspreis: 1,00 EUR

Dies ist ein Open-Source Projekt.
Wie dieses Magazin abonniert
werden kann ist auf der Internetseite
beschrieben.

www.embedded-projects.net/journal

EPJ Fotonachweise
Titel: mehdi hama, spotlight-studios
Anzeige: Ramona Heim,Nicemonkey, dip

Jetzt den Grasshopper anschließen und booten. Im u-boot den Vorgang mit „SPACE“ abbrechen und folgende Kommandos eingeben:

```
ICnova> setenv serverip 192.168.x.x      #die IP Adresse des Servers
ICnova> setenv ipaddr 192.168.x.y        #die IP Adresse des grasshoppers
ICnova> tftpboot 11000000 uImage
ICnova> bootm
```

Wenn das System gebootet ist und man sich angemeldet hat, ist es erforderlich, die neuen Module - via NFS - in das System zu kopieren. Dazu:

```
Thank you for using In-Circuit ICNova

icnova login: root


BusyBox v1.5.0 (2008-04-24 11:31:03 CEST) Built-in shell (ash)
Enter ‚help‘ for a list of built-in commands.

# portmap
# mount -t nfs 192.168.x.x/${NFS} /mnt
# cp -a /mnt/module/* /
# modprobe atmel-mci
# modprobe mmc_block
# modprobe vfat
```

(Der portmap-Befehl ist dazu da, um sich auf den NFS-Server verbinden zu können) Jetzt sollte eigentlich alles funktionieren und man kann die SD-Karte mounten (siehe Abbildung 4).

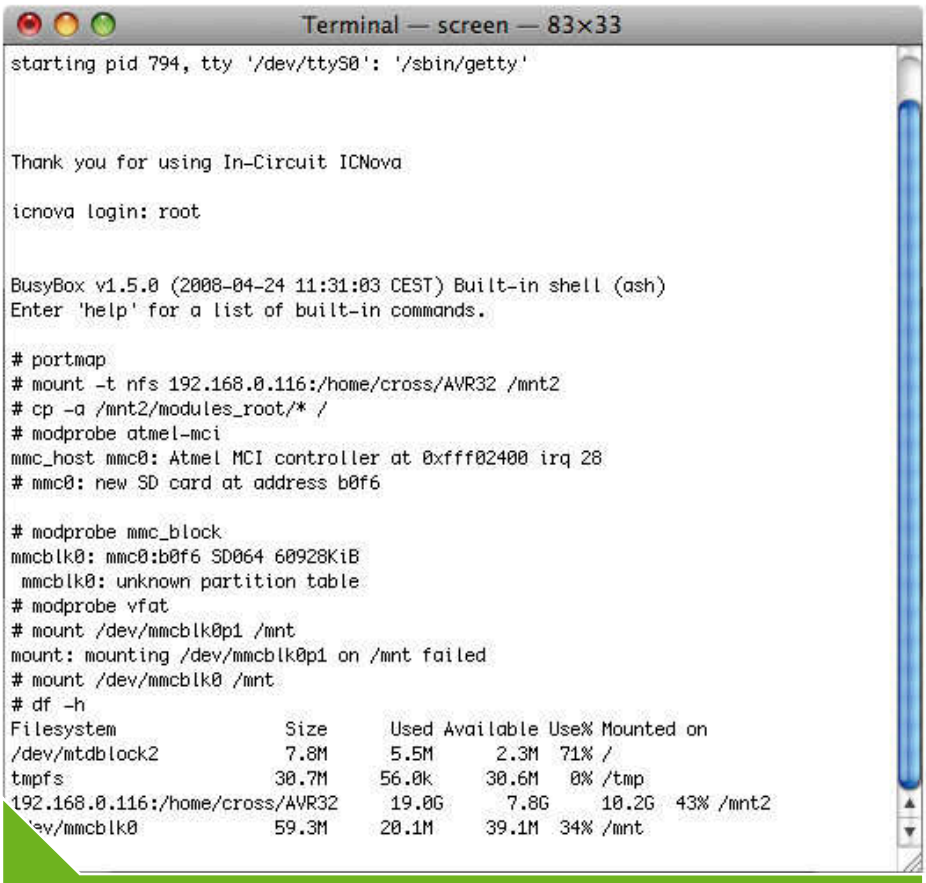


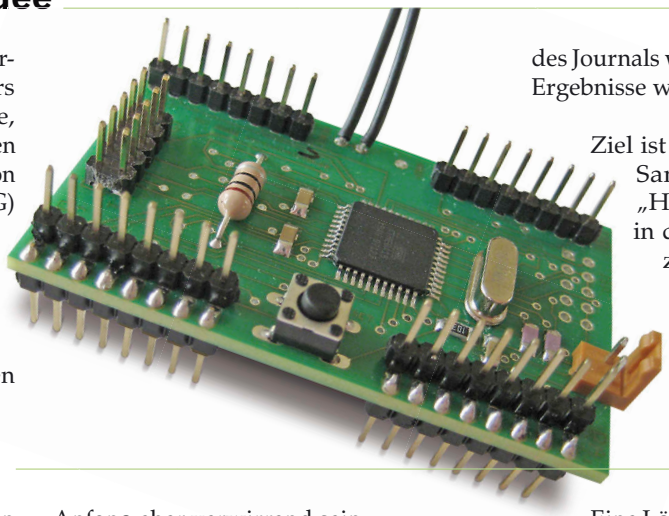
Abbildung 4: Beispielsession

Open-Source Mikrocontroller Kurs

Artikelreihe Mikrocontroller Projektgruppe (mcPG) des Augsburger Computerforums

Das Vorhaben und die Idee

Im Rahmen des Embedded-Projects Journals soll ein freier Mikrocontrollerkurs samt Unterlagen wie Beispielquelltexte, Schulungsunterlagen und einem passenden Hardwarepool enstehen. Der Kurs wird von der Mikrocontroller Projektgruppe (mcPG) des Augsburger Computerforums geleitet und hoffentlich finden sich beim Lesen des Artikels noch viele Interessenten, die dieses Projekt in irgendeiner Art und Weise unterstützen und daran mitwirken wollen. In den folgenden Ausgaben des Journals wird über das Projekt und die Ergebnisse weiterhin berichtet.



Ziel ist letztendlich, dass eine gute Sammlung samt passendem „Handbuch“ für den Einstieg in die Mikroconroller-Welt, frei zum Download angeboten werden kann.
Beheimatet ist das Projekt hier: http://www.mikrocontrollerkurs.de

Aller Anfang ist schwer

Oft ist der Zugang in die Welt der kleinen Helfer schwierig. Allein die große Auswahl an Anbietern und deren Entwicklungsboards ist für den Anfänger schwer zu durchschauen. So bieten nicht nur die Hersteller [1,2] ihre eigenen Lösungen an. Einige Hersteller haben sich speziell auf Entwicklungsboards spezialisiert [3]. Auch viele Elektronikhändler [4] haben eigene Schaltungen im Programm. Die meist proprietären Lösungen bieten oft komplexe Boards an. Sinnvolle Baugruppen auf einem Entwicklungsboard, die für zukünftige Ausbaustufen vorgesehen sind, können am Anfang eher verwirrend sein.

Grade der Einsteiger weiß aber gar nicht, was er genau braucht. Die richtige Auswahl im Dschungel der Angebote zu treffen, ist deshalb für ihn schwer. Bei seiner Auswahl kann man als Einsteiger kaum beurteilen, was wirklich sinnvoll ist. Es wäre also wünschenswert, dass man mit einem sehr einfachen Einstieg beginnt. Die notwendige Hardware sollte aber später auch zu größeren Schaltungen ausgebaut werden können.

Eine Lösung versucht die Mikrocontroller Projektgruppe (mcPG) des Augsburger Computerforums (ACF) [5] nun zu entwickeln. Ziel der Entwicklung sind standardisierte Funktionseinheiten als Module für Steckbretter [6]. Dabei wird zu jedem Hardware-Modul auch das passende Software-Modul entwickelt. Die Kombination erlaubt es dem Benutzer, seine Schaltung wie bei einem Lego-System schnell aufzubauen und mittels der Software Schritt-für-Schritt zu testen. Bei diesem iterativen Vorgehen können Fehler in der Hardware schnell isoliert und behoben werden.

Aktueller Stand des Projektes

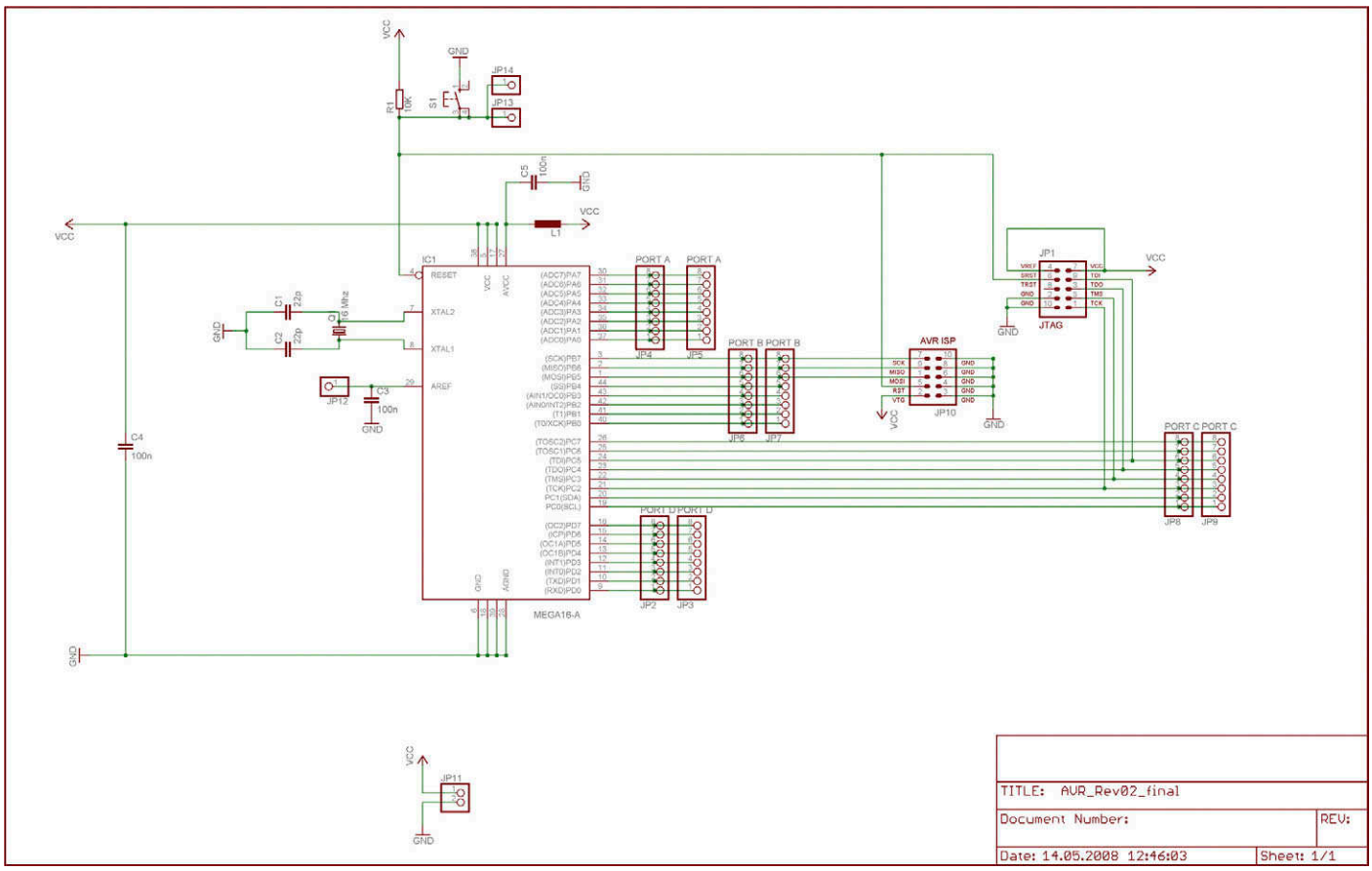
In diesem Artikel soll zunächst das Mikrocontroller-Modul näher vorgestellt werden. Die weiteren Module folgen in den nächsten Ausgaben des „Embedded-Projects Journals“.

Das CPU Modul wurde möglichst einfach aufgebaut (s. Bauteileliste). Um Platz zu sparen, wird ein AVR Atmega16 im TQFP44 Gehäuse verwendet. Bis zur endgültigen Verabschiedung des Moduls wird er unter Umständen gegen einen PIN-kompatiblen Atmega32 ausgetauscht. Die VCC und GND Pins sind mit einem Abblockkondensator (100nF) stabilisiert. Der externe Quarz taktet mit 16 MHz. Sämtliche Ports (IO Ausgänge) sind jeweils für das Steckbrett nach unten mittels Stiftleisten ausgeführt. Damit ist die Kontaktierung sicher gestellt und das Modul steht fest auf dem Untergrund.
Um die Messung und Verkabelung der I/O-Pins zu erleichtern, sind auf der Oberseite Buchsenleisten angebracht. Ein Reset

Bauteile	Wert	Beschreibung	Bauform
C1,C2	22p	Kondensator	SMD C1206
C3,C4,C5	100n	Kondensator	SMD C1206
IC1	AVR ATMEGA16	Atmel AVR	TQFP44
JP2-JP9	-	PINHEAD	1X08
JP11	-	PINHEAD	1X02
JP12-JP14	-	PINHEAD	1X01
JP1	-	AVR-JTAG-10 Pin	2X05
JP10	-	AVR-ISP-10 Pin	2X04
L1	10uH	Spule	2200-15.24
Q1	16 Mhz	CRYSTAL	HC49U-V
R1	10K	Widerstand	SMD R1206
S1	-	Taster	-

Schalter ermöglicht den Neustart der Atmel CPU. Die Entstörung des ADC (Analog Digital Wandler) wurde mit einer kostengünstigen Spule realisiert. Zur Programmierung (Flashen) ist eine ISP- und JTAG Schnittstelle vorhanden. Kleinteile wie Widerstände und Kondensatoren komplettieren den Aufbau. Mehr ist auf dem Modul nicht vorhanden.

Bis zur nächsten Ausgabe des „Embedded-Projects Journal“ im Dezember 2008 wird das Design noch weiter verfeinert. Für Verbesserungsvorschläge ist die Mikrocontroller-Gruppe immer dankbar.



Open-Source Online-Mikrocontroller-Kurs

Wie bereits erwähnt wird in Zusammenarbeit mit dem „Embedded-Projects Journal“ und dem ACF wird dieser Online-Kurs für den Einstieg in die Mikrocontroller-Welt entwickelt. Interessierte Teilnehmer und Leute die den Kurs mit gestalten wollen, können sich unter http://www.mikrocontrollerkurs.de melden. Arbeit gibt es genug :-).

Die notwendige Hardware wird später ebenfalls über die Internetseite angeboten. Durch den einfachen Aufbau kann der Preis niedrig gehalten werden, um auch Schülern und Studenten den Zugang nicht zu verbauen.

Quellen:
1 - http://www.microchip.com/
2 - http://www.atmel.com/products/AVR/
3 - http://www.olimex.com/
4 - http://www.pollin.de/ ATMEL Evaluations-Board Version 2.0.1 – Bausatz
5 - http://www.augusta.de/AG/micro/
6 - http://de.wikipedia.org/wiki/Breadboard
7 - http://www.mikrocontrollerkurs.de

JTAGICE-mkII-Klon

Zwischenbericht zur aktuellen Entwicklung

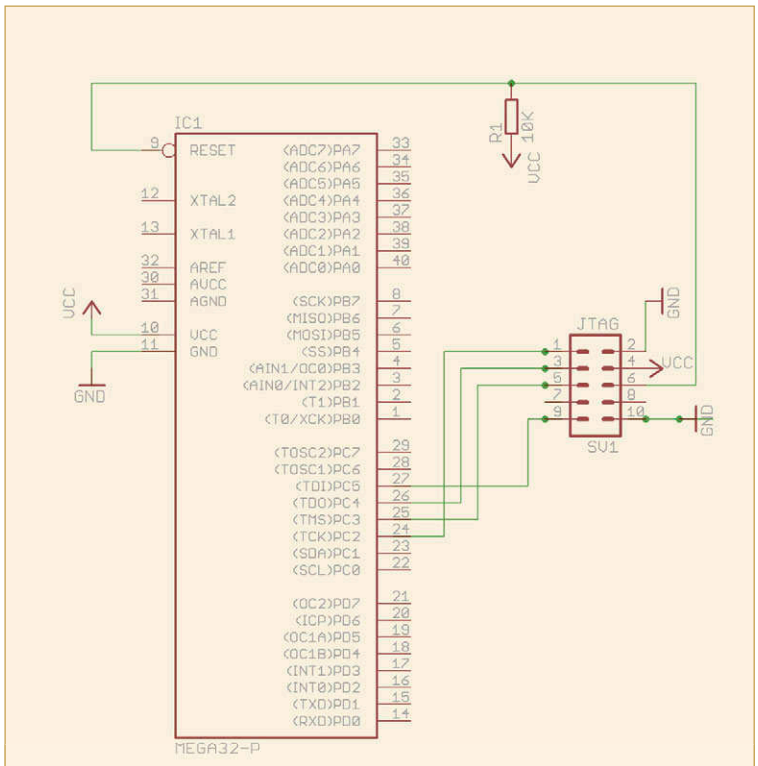
Martin Lang <martin.lang@rwth-aachen.de>

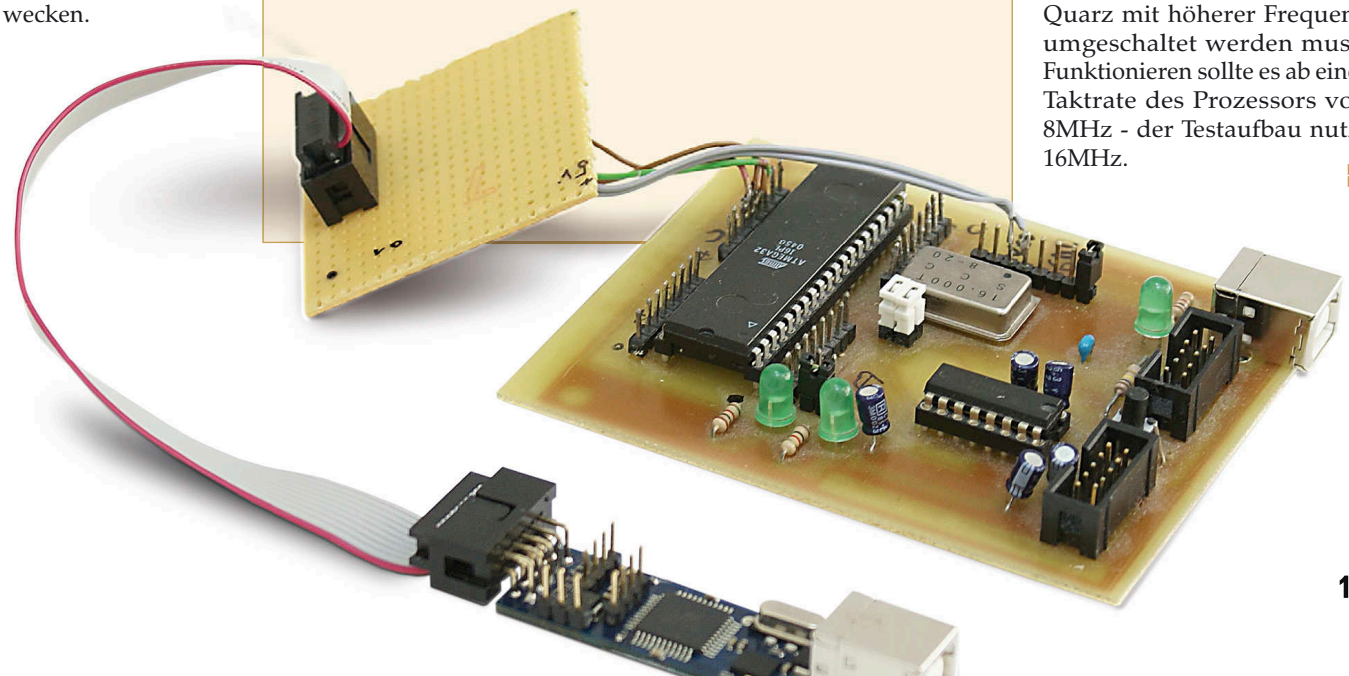
Die JTAGICE-mkII-Klon Firmware für den USBprog Adapter ermöglicht den Zugriff auf Mikrocontroller der AVR Familie über deren JTAG Schnittstelle. Während die meisten anderen Nachbauten der Atmel JTAG Adapter einen Nachbau der Hardware darstellen, der mit der original Atmel Firmware betrieben wird, ist dieser Ansatz für den USBprog nicht möglich.
Die entwickelte Firmware stellt somit eine komplette Eigenentwicklung dar. Da durch Atmel lediglich das Programmierinterface und nicht das Debugginginterface dokumentiert ist, stützt sich die Entwicklung der Debuggingfunktionen auf Reverse Engineering und die wenigen Online verfügbaren Informationen.
Von der Seite des PC aus gesehen, besteht das Ziel darin, das Verhalten eines JTAGICE-mkII nachzubauen, so dass letztlich eine Benutzung mit allen verbreiteten Werkzeugen für AVR Mikrocontroller möglich ist. Die Firmware befindet sich derzeit noch in einem Entwicklungszustand und wird wohl noch einige Zeit bis zu ihrer Fertigstellung benötigen. Wir haben uns trotzdem entschieden ein wenig darüber zu berichten, um etwas Interesse an dem Projekt zu wecken.

Zum jetzigen Zeitpunkt werden im Betastadium das Programmieren via JTAG, sowie grundlegende Debugfunktionen unterstützt. Dies umfasst das Starten und Stoppen des Controllers, die Nutzung der vier integrierten Hardwarebreakpoints als Programm- oder Datenhaltepunkt, sowie das Auslesen und Beschreiben von Registern, RAM und EEPROM. Darüber hinaus sind das Auslesen des Programmspeichers während des Betriebs und die Ausführung von einzelnen Instruktionen (Single-Step) implementiert. Die Möglichkeit zum Beschreiben des Flash während des Programmablaufs befindet sich noch in der Entwicklung. Somit sind noch keine Softwarebreakpoints möglich.

Die aktuelle Test- und Entwicklungsumgebung besteht aus einem einfachen Entwicklungsboard mit einem ATMega16 Controller. Die Verbindung des 10pol-Steckers am USBprog mit einem Mega16/32 wird in obiger Grafik gezeigt. Auf dem PC verwende ich Linux, sowie avrdude zum Test der Programmierfunktionalitäten und avarice zusammen mit avr-gdb zum Test der Debuggingfunktionen. Hierbei fungiert avarice als Remote-GDB, der die GDB-Befehle in das Protokoll des JTAGICE umsetzt.

In der Benutzung mit avrdude unterscheidet sich der JTAGICE-Klon kaum von anderen bekannten Programmieradaptern. Es ist lediglich im „-c“-Parameter „jtag2“ als Programmiergerät anzugeben. Ein Beispielaufruf von avrdude zum Flashen des Mega16 in meiner Testumgebung ist in Code 2 zu finden. Leider sind in der momentanen Entwicklungsphase noch keine Möglichkeiten geschaffen, die JTAG-Taktrate durch Parameter zu verringern, so dass ein fabrikneuer Mikrocontroller vorher auf andere Art und Weise auf einen externen Quarz mit höherer Frequenz umgeschaltet werden muss. Funktionieren sollte es ab einer Taktrate des Prozessors von 8MHz - der Testaufbau nutzt 16MHz.





Flashen

```
#include <avr/io.h>

unsigned char var;
const unsigned char __attribute__((__progmem__)) evar = 5;
void Testfunktion() {
        var++;
        var++;
        var++;
        var++;
        var++;
}
int main() {
        uint16_t test;
        uint8_t counter = 0;


        // initialisierung der LED's
        DDRD = (1<<5) | (1<<6) | (1<<7);
        PORTD = (1<<5);

        while (1) {
                // delay
                for (test = 0; test < 0xFFFF; ++test);
                for (test = 0; test < 0xFFFF; ++test);
                // led togglen
                PORTD ^= (1<<5);
                PORTD ^= (1<<6);

                Testfunktion();

                ++counter;
        }
        return 0;
}
```

Code 1: Programmausschnitt eines Testprogramms

```
$ avrdude -c jtag2 -P usb -p m16 -U flash:w:testprog.bin
avrdude: usb_open(): cannot read serial number „No error"
avrdude: usb_open(): cannot read product name „No error"

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.16s

avrdude: Device signature = 0x1e9403
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be per-
formed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file „testprog.bin"
avrdude: input file testprog.bin auto detected as raw binary
avrdude: writing flash (272 bytes):

Writing | ################################################## | 100% 0.60s

avrdude: 272 bytes of flash written
avrdude: verifying flash memory against testprog.bin:
avrdude: load data flash data from input file testprog.bin:
avrdude: input file testprog.bin auto detected as raw binary
avrdude: input file testprog.bin contains 272 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.33s

avrdude: verifying ...
avrdude: 272 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

Code 2: avrdude flasht unter Nutzung des JTAGICE

Soweit stellt das jedoch keine Überraschung dar - immerhin können wir die gleiche Funktionalität bei der Nutzung der ISP Schnittstelle mit dem AVRISPmkII-Klon erreichen.

Der Vorteil bei der Nutzung von JTAG im Vergleich zur ISP Schnittstelle liegt im OnChipDebug-Modus der AVR-MCUs. Um einen kleinen Ausblick zum Ziel der Firmware zu ermöglichen, möchte ich an

```
$ ./avarice -2 --jtag usb :4242
AVaRICE version 2.7.20071102, Mar 21 2008 15:51:47

Defaulting JTAG bitrate to 1 MHz. Make sure that the target
frequency is at least 4 MHz or you will likely encounter failures
controlling the target.

JTAG config starting.
Found a device: JTAGICEmkII
Serial number:  00:a0:00:00:0d:3f
Reported JTAG device ID: 0x9403
Configured for device ID: 0x9403 atmega16
JTAG config complete.

Preparing the target device for On Chip Debugging.

Disabling lock bits:
  LockBits -> 0xff

Enabling on-chip debugging:
  Extended Fuse byte -> 0xff
      High Fuse byte -> 0x19
       Low Fuse byte -> 0xff
Waiting for connection on port 4242.
Connection opened by host 127.0.0.1, port 58441.
```

```
$ avr-gdb testprog.elf
GNU gdb 6.7.1
Copyright (C) 2007 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type „show copying“
and „show warranty“ for details.
This GDB was configured as „--host=i686-pc-linux-gnu --target=avr“...
(gdb) target remote :4242
Remote debugging using :4242
0x00000000 in __vectors ()
(gdb) break test.c:28
Breakpoint 1 at 0x126: file test.c, line 28.
(gdb) break test.c:7
Breakpoint 2 at 0x9c: file test.c, line 7.
(gdb) c
Continuing.

Breakpoint 1, main () at test.c:28
28                      PORTD ^= (1<<5);
(gdb) c
Continuing.

Breakpoint 2, Testfunktion () at test.c:7
7               var++;
(gdb) c
Continuing.

Breakpoint 1, main () at test.c:28
28                      PORTD ^= (1<<5);
(gdb) c
Continuing.

Breakpoint 2, Testfunktion () at test.c:7
7               var++;
(gdb) print var
$1 = 191 ‚¿‘
(gdb) c
Continuing.

Breakpoint 1, main () at test.c:28
28                      PORTD ^= (1<<5);
(gdb) print counter
$2 = 2 ‚\002‘
(gdb) c
Continuing.

Breakpoint 2, Testfunktion () at test.c:7
7               var++;
(gdb) c
Continuing.

Breakpoint 1, main () at test.c:28
28                      PORTD ^= (1<<5);
(gdb) print counter
$3 = 3 ‚\003‘
(gdb) delete breakpoints
Delete all breakpoints? (y or n) y
(gdb) c
Continuing.

Program received signal SIGINT, Interrupt.
0x000000fa in main () at test.c:25
25                      for (test = 0; test < 0xFFFF; ++test);
(gdb)
```

dieser Stelle bereits einige grundlegende Funktionalitäten vorstellen, die ich erfolgreich testen konnte. Als ersten wichtigen Schritt muss man dazu das OCD-Fuse im Mikrocontroller setzen. Anschließend kann man mit avarice eine Verbindung zum JTAGICE aufbauen. Hierbei übergibt man avarice den Port, auf welchem es auf eingehende GDB Verbindungen warten soll, durch z. B. „:4242“. Nach dem Start von avarice hält der JTAGICE das laufende Programm an und resettet den Controller.

Debuggen

An diesem Punkt angelangt kann man nun den avr-gdb starten und das ELF-Binary mit Debuginformationen zum Programm laden. Durch Eingabe des Befehls „target remote :4242“ verbindet man nun den Debugger mit avarice, woraufhin sich diese synchronisieren. Der Debugger zeigt nun an, dass sich das Programm am Startvektor (Adresse 0) befindet. Zur Demonstration der Anwendung nutze ich die Gelegenheit und lege einen Haltepunkt innerhalb der Programmschleife sowie in der Testfunktion fest. Startet man nun das Programm, so zeigt der Debugger kurz darauf wieder an, dass es am Haltepunkt angehalten hat. Nun ist es möglich, sich die Werte der Variablen anzuschauen und im Einzelschrittmodus durch das Programm zu gehen. Ein kleiner Ausschnitt der Möglichkeiten ist hier abgedruckt. Sollte dies euer Interesse geweckt haben und ihr Lust bekommen habt, die weitere Entwicklung zu unterstützen oder wenn einfach nur Fragen stellen wollt, dann könnt ihr mich gerne kontaktieren.

Links

1. SVN Repository
http://svn.berlios.de/svnroot/repos/usbprog/trunk/jtagicemk2klon/

2. Firmware Binary
http://svn.berlios.de/svnroot/repos/usbprog/trunk/jtagicemk2klon/jtagice2.bin

3. Bootloader für USBprog
http://svn.berlios.de/svnroot/repos/usbprog/trunk/usbprog_base/firmware/usbprog_base.hex

4. Infos Rund um die Firmware
http://www.embedded-projects.net/index.php?page_id=163

15 Ton Melodiegenerator

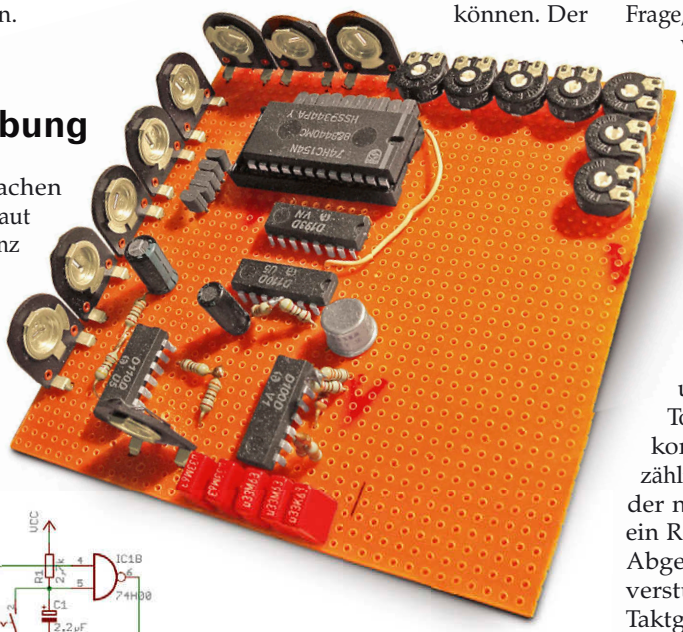
Julius Weber

Einleitung

Ich möchte in diesem Artikel eine Schaltung vorstellen, die besonders gut für Anfänger geeignet ist, da sie ohne Microcontroller auskommt. Der Melodiegenerator kann z.B. als Haustürklingel eingesetzt werden, wobei jedoch beachtet werden muss, dass die Leitung zum „Klingelknopf“ geschirmt sein sollte, um mögliche Störeinflüsse bei längeren Strecken zu vermeiden.

Schaltungs- und Funktionsbeschreibung

Wer schon einmal einen einfachen Tongenerator oder eine Sirene gebaut hat weiß, dass eine hörbare Frequenz erzeugt wird, die verstärkt und durch einen Lautsprecher ausgegeben wird. Die Frequenz ist durch einen Kondensator oder einen Widerstand veränderbar. In unserem Fall sind es Widerstände, die die Frequenz bestimmen. Ein einfacher Tongenerator mit einer festen Frequenz kann logischerweise nur einen Ton wiedergeben. Wir wollen aber 15 Töne nacheinander wiedergeben, also brauchen wir auch 15 Frequenzen. Um die Frequenzen zu ändern, würde man bei einem einfachen Tongenerator den Widerstand oder das Poti auswechseln, welches die Frequenz bestimmt. Wir nehmen also 15 Potis, damit wir die Frequenzen später bequem für jeden Ton einzeln einstellen können. Der Tongenerator selbst muss nicht weiter erklärt werden. Was man wissen muss, ist, dass er aus einem 7400 besteht und dass eine Sinus-Frequenz erzeugt wird. Der verwendete Tongenerator verfügt nur über einen Transistor der als Verstärker dient. Eine Lautstärkeregelung ist nicht vorgesehen. Sie ist aber für den fortgeschrittenen Anfänger leicht mit einem regelbaren OPV möglich. Da dies nun geklärt ist, stellt sich nun die Frage, wie die Töne nacheinander abgespielt werden bzw. die Potis ausgewechselt werden. Für dieses Problem kommt ein Zähler zum Einsatz. Da ein Binärzähler benutzt wird, müssen seine Ausgänge dekodiert werden. Für diese Aufgabe ist der 1 aus 16 Decoder 74154 angedacht. An seine Ausgänge sind die Potis angeschlossen. Es ist nun möglich verschiedene Töne auszugeben, da die Widerstandswerte durch den Zähler umgeschaltet werden und so verschiedene Frequenzen im Tongenerator entstehen. Doch woher kommt die Frequenz, die der Zähler zählt? Sie kommt von einem Taktgenerator, der nach Betätigen des Tasters beginnt, ein Rechtecksignal abzugeben und nach Abgeben des Stopsignals vom Decoder verstummt. Wie hoch die Frequenz des Taktgenerators ist, bestimmt später wie schnell die Melodie abgespielt wird. Die Geschwindigkeit kann mit dem 1kΩ Poti eingestellt werden. Wer die Schaltung betrachtet hat wird feststellen, dass der Decoder 74154 16 Ausgänge hat, wir aber nur 15 verschiedene Töne erzeugen. Wir könnten natürlich 16 Töne erzeugen, aber wir brauchen einen Ausgang, der das Stopsignal setzt, welches den Taktgenerator deaktiviert und somit den Prozess des Abspielens der Melodie beendet. Man will ja schließlich nicht, dass die Melodie unendlich lange abgespielt wird, sondern nur einmal durchläuft und dann verstummt. Wer mehr Töne haben möchte, kann natürlich den Zähler erweitern und einen 2 (oder mehr) Zähler vom Typ 74193 mit dem gleichen Decoder hinzufügen. Der Übertrag vom 1. 74193 wird hierzu einfach an den Takteingang des 2. Zählers gelegt. Das Prinzip ist immer dasselbe. Als Stromversorgung kann eine Batterie oder ein Transformator mit stabilisierter Gleichspannung benutzt werden. Abschließend ist noch zu sagen, dass um relativ gut klingende Töne zu erzeugen die Größe vom Kondensator C3 von 1-3μf zu variieren ist.



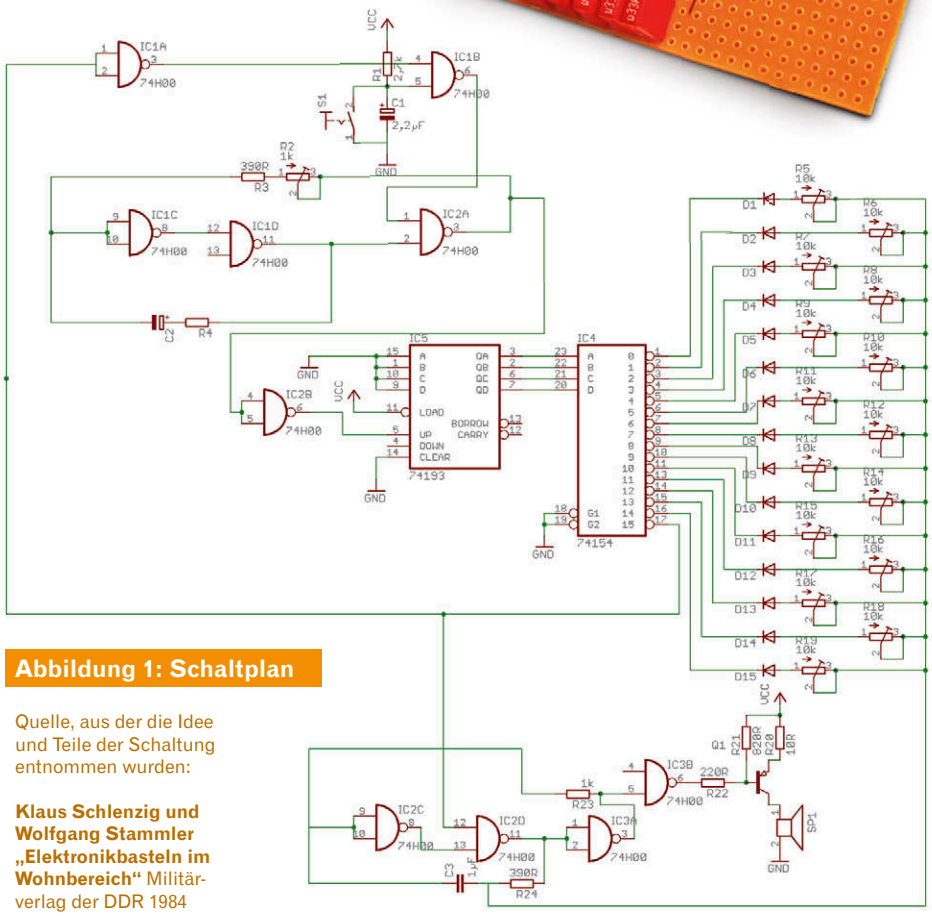


Abbildung 1: Schaltplan

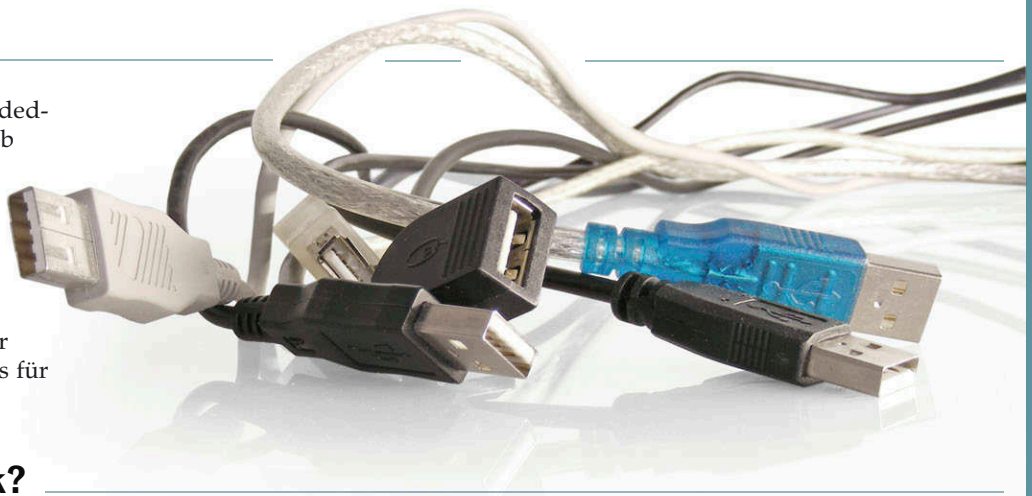
Quelle, aus der die Idee und Teile der Schaltung entnommen wurden:

Klaus Schlenzig und Wolfgang Stammler „Elektronikbasteln im Wohnbereich“ Militärverlag der DDR 1984

USB im praktischen Einsatz

Benedikt Sauter <sauter@ixbat.de>

Einleitung



In der ersten Augsgabe des Embedded-Projects Journals (Ausgabe 1/2008) gab es bereits einen USB-Grundkurs. Daraufhin fragten einige Leser nach, ob es vielleicht im nächsten Heft einen Artikel geben könnte, in dem konkret gezeigt wird, wie man sich eine USB-Schnittstelle programmiert. Aus diesem Grund möchte ich hier eine kleine Einführung in USB-Stacks für Microcontroller geben.

Was ist ein USB-Stack?

Für die Kommunikation zwischen Computer und USB-Gerät müssen diverse Zustände, Anfragen und Antworten verwaltet werden. Da die USB-Kommunikation wesentlich aufwändiger, als beispielsweise eine einfache UART Verbindung ist, werden daher auf der Treiberseite im Computer und in der Firmware auf einem USB-Gerät wesentlich mehr Verwaltungsaufgaben erledigt werden müssen. Im Computer ist dafür das USB-Subsystem zuständig und im USB-Gerät ein für die USB-Schnittstelle passender Stack.
Während der Enumeration müssen Anfragen im USB-Gerät abgearbeitet und Datenstrukturen für die Flusskontrolle sowie Datenübertragung verwaltet werden. Jedes USB-Gerät muss auf diese Weise die Aufgaben eines nach der USB-Spezifikation definierten Gerätes ordnungsgemäß erfüllen. Abhängig vom eingesetzten Baustein liefert dieser mehr oder weniger die USB-Intelligenz mit.
Meistens ist es jedoch so, dass die USB-Bausteine nur die unterste Protokollebene von USB bearbeiten können. Alles andere muss in einer Softwarebibliothek von einem Mikrocontroller erledigt werden. Diese Softwarebibliothek nennt man im Allgemeinen USB-Stack.

Wie bekommt man USB in das eigene Projekt?

Zuerst muss ein passender Baustein ausgewählt werden, der die USB-Schnittstelle anbietet. Im Wesentlichen gibt es hier zwei grundsätzliche Geschwindigkeitsunterschiede: Der Baustein kann mit Low- (1,5Mbit/s) und Fullspeed (12 Mbit/s) Geschwindigkeit arbeiten, oder er kann sogar zusätzlich Highspeed (480Mbit/s) Daten verwerten. Die zuletzt genannten sind jedoch für einfache Mikrocontrollersysteme weniger interessant und eher selten verfügbar, da bei 480 Mbit/s jeder einfache Mikrocontroller keine Chance hätte, mit dem Verarbeiten der Daten hinterher zu kommen.

Ist der Baustein ausgewählt, wird wie bereits erwähnt, ein passender USB-Stack benötigt. Oft liefern Hersteller selbst passende Bibliotheken. Für Open-Source Projekte fallen solche jedoch aus, da sich die Lizenzen meistens nicht sinnvoll mit freien Lizenzen verbinden lassen. An dieser Stelle ist noch kurz zu erwähnen, dass es darüber hinaus ein paar Firmen gibt, die ausschließlich von der Entwicklung und Vermarktung von USB-Stacks für Mikrocontroller leben. Im weiteren Verlauf werden wir aber nur offene und freie Stacks betrachten.

Vorstellung der einzelnen Stacks

Nun werden Stacks, die ich am interessantesten finde (siehe Tabelle 1), vorgestellt. Die Stacks sollen nicht direkt miteinander verglichen werden, sondern es sollen Stärken und Schwächen gezeigt werden. Um dennoch ein einheitliches Ergebnis zu erhalten, werden zu jedem Stack die folgenden Punkte betrachtet:

- Benötigte Dateien (bzw. Dateistruktur)
- Konfiguration eines eigenen Gerätes
- Einbindung in eine bestehende Software und Portierbarkeit auf andere Systeme

Baustein	Projekt	Funktionsweise
LPC2148	Lpcusb http://wiki.sikken.nl/index.php?title=LPCUSB	Mikrocontroller inkl. USB-Schnittstelle (ARM7)
USBN9604	Usbn2mc http://usbn2mc.berlios.de	USB-Bridge (ansteuerbar per SPI, Parallel-Bus und DMA)
AT90USB1287	AT90USB_firmware http://www.ssalewski.de/Misc.html.de	Mikrocontroller inkl. USB-Schnittstelle (AVR)
ATmega8	AVR-USB http://www.obdev.at/products/avrusb/index-de.html	USB-Stack basierend auf zwei IO-Ports des ATMega8

LPC2148 – Der König unter den USB-Stacks: lpcusb

Internetseite: http://wiki.sikken.nl/index.php?title=LPCUSB

Ohne parteiisch zu werden, kann ich gleich sagen, dass dieser Stack am meisten ausgereift und am benutzerfreundlichsten ist. Er ist sehr sauber und übersichtlich programmiert worden, zudem gibt es eine Menge fertiger Beispiele im Quelltextarchiv für verschiedenste USB-Geräte (virtuelle serielle Schnittstelle, Massenspeicher-Gerät, HID-Gerät, etc.). Der Stack kann genauso gut für eine „standalone"-Anwendung sowie als USB-Subsystem in einem Mikrocontroller Betriebssystem (z.B. FreeRTOS & Co.) verwendet werden.

Benötigte Dateien (bzw. Dateistruktur):
Die benötigten Dateien können einfach in den Übersetzungsprozess integriert werden. Die Struktur ist sehr einfach gehalten und daher müssen Abhängigkeiten kaum beachtet werden. Dadurch ist eine sehr leichte Integration in ein bestehendes Projekt möglich. In der folgenden Tabelle sind alle Dateien, welche für den Betrieb notwendig sind, aufgelistet und kurz beschrieben.

Datei	Beschreibung
type.h	Datentypen
usbcontrol.c	Transfer für Endpunkt 0 / Geräte Erkennung, etc.
usbinit.c	Initialisierung USB-Hardware und -Stack
usbhw_lpc.c / usbhw_lpc.h	Hardware Layer
usbstruct.h	Definition Datentypen für USB-Strukturen
lpc214x.h	Header-Datei für LPC2148
usbapi.h	USB-API für Anwendung
usbdebug.h	Debugausgaben Makro
usbstdreq.c	Standardanfragen aus Kapitel 9 der USB-Spezifikation
startup.c / startup.h	Initialisierung Prozessor, PLL, Interrupts

```
static const U8 abDescriptors [] =
{
  //  Device descriptor
  0x12,
  DESC_DEVICE,
  LE_WORD(0x0200),             // bcdUSB
  0x00,                                // bDeviceClass
  0x00,                                // bDeviceSubClass
  0x00,                                // bDeviceProtocol
  MAX_PACKET_SIZE0,            // bMaxPacketSize
  LE_WORD(0xFFFF),             // idVendor
  LE_WORD(0x0005),             // idProduct
  LE_WORD(0x0100),             // bcdDevice
  0x01,                                // iManufacturer
  0x02,                                // iProduct
  0x03,                                // iSerialNumber
  0x01,                                // bNumConfigurations

  //  Configuration descriptor
  0x09,
  DESC_CONFIGURATION,
  LE_WORD(31),                  // wTotalLength
  0x02,                                // bNumInterfaces
  0x01,                                // bConfigurationValue
  0x00,                                // iConfiguration
  0xC0,                                // bmAttributes
  0x32,                                // bMaxPower

  //  Data class interface descriptor

  0x09,
  DESC_INTERFACE,
  0x01,                                // bInterfaceNumber
  0x00,                                // bAlternateSetting
  0x02,                                // bNumEndPoints
  0x0A,                                // bInterfaceClass = data
  0x00,                                // bInterfaceSubClass
  0x00,                                // bInterfaceProtocol
  0x00,                                // iInterface

  //  Data EP OUT

  0x07,
  DESC_ENDPOINT,
  BULK_OUT_EP,                 // bEndpointAddress
  0x02,                                // bmAttributes = bulk
  LE_WORD(MAX_PACKET_SIZE), // wMaxPacketSize
  0x00,                                // bInterval

  //  Data EP in

  0x07,
  DESC_ENDPOINT,
  BULK_IN_EP,                  // bEndpointAddress
  0x02,                                // bmAttributes = bulk
  LE_WORD(MAX_PACKET_SIZE), // wMaxPacketSize
  0x00,                                // bInterval

  //  String descriptors

  0x04,
  DESC_STRING,
  LE_WORD(0x0409),
  0x10,
  DESC_STRING,
  ,O`, 0, ,t`, 0, ,o`, 0, , ,, 0, ,U`, 0, ,S`, 0, ,B`, 0,
  0x16,
  DESC_STRING,
  ,U`, 0, ,S`, 0, ,B`, 0,` ,,0, ,G`, 0, ,e`, 0, ,r`, 0, ,a`, 0, ,e`, 0, ,t`, 0
  0x12,
  DESC_STRING,
  ,2`, 0, ,0`, 0, ,0`, 0, ,8`, 0, ,0`, 0, ,8`, 0, ,0`, 0, ,6`, 0,
  //  Terminating zero
  0 };
```

Konfiguration des eigenen Gerätes:
Die Deskriptoren werden über einfache Arrays angelegt. Das ist bei der Definiton von aufwändigeren USB-Schnittstellen von Vorteil, jedoch bedeutet dies immer, dass man sich etwas in die USB-Spezifikation einarbeiten muss, um die Datenstrukturen (siehe Listing abDescriptor) erstellen zu können. Im ersten Absatz des folgenden Listings werden die gerätespezifischen Datenstrukturen und die für die internen USB-Datenstrukturen wichtige Informationen und Verknüpfungen angegeben. Man nennt diesen Bereich auch Gerätedeskriptor. Direkt im Anschluss findet man den Konfigurationsdeskriptor, welcher im Wesentlichen die Eigenschaften bzgl. der Stromversorgung enthält. Die wirkliche Kommunikation geschieht dann wie bereits im USB-Grundkurs ausführlich beschrieben über die Interfaces mit ihren „Endpunktbündeln". Ein Interface beschreibt die Anzahl der zugehörigen Endpunkte. Und ein Endpunkt beschreibt, mit welchem Übertragungsverfahren (Interrupt, Bulk oder Isochron) übertragen wird, und wie viele Daten aufgenommen werden können. Zu guter Letzt sind noch die Stringdeskriptoren angegeben, welche dem USB-Gerät ein Namen im Betriebssystem geben können. So kann jedes Betriebssystem - ohne das Gerät zu kennen - dem Benutzer sagen, welches Gerät angesteckt worden ist.

Ist die Schnittstelle über die „abDescriptor" Datenstruktur beschrieben worden, müssen nur noch die passenden Funktionsaufrufe für die Initialisierung durchgeführt werden.

```
usbRegisterHandlers ();
usbRegisterDescriptors (abDescriptors);
usbHardwareRegisterEPIntHandler (BULK_IN_EP, usbcomBulkIn);          //Callback für ausgehende Daten
usbHardwareRegisterEPIntHandler (BULK_OUT_EP, usbcomBulkOut);    //Callback für eingehende Daten
usbHardwareNakIntEnable (INACK_BI);
usbSetupInterruptHandler ();
usbHardwareConnect (TRUE);
```

```
static int usbcomBulkOut (U8 bEP, U8 bEPStatus __attribute__ ((unused)))
{
  int i;
  int iLen;
  portBASE_TYPE xTaskWokenByPost = pdFALSE;
  iLen = usbHardwareEndpointRead (bEP, abBulkBuf, sizeof (abBulkBuf));
  for (i = 0; i < iLen; i++)
    xTaskWokenByPost = xQueueSendFromISR (xUSBRXQueue, &abBulkBuf [i], xTaskWokenBy-
Post);
  return xTaskWokenByPost;
}
```

Einbindung in eine bestehende Softwarearchitektur:
Wenn Daten vom PC empfangen werden, wird automatisch die registrierte Funktion aufgerufen. Hier wurde usbcomBulkOut definiert. Intern in dieser Funktion müssen nur die Daten aus den USB-FIFO-Speicher ausgelesen werden.

Für ausgehende Daten sieht der Zugriff ähnlich aus. Jedesmal, wenn Daten vom PC aus empfangen werden können, wird die folgende Funktion auf Grund der Registrierung als Callback aufgerufen. Hierbei werden die zu versendenden Daten aus internen Datenstrukturen in die USB-FIFO-Speicher übertragen.

```
static int usbcomBulkIn (U8 bEP, U8 bEPStatus __attribute__ ((unused)))
{
  int i;
  portBASE_TYPE xTaskWoken = pdFALSE;
  for (i = 0; i < MAX_PACKET_SIZE; i++)
    if (!xQueueReceiveFromISR (xUSBTXQueue, &abBulkBuf [i], &xTaskWoken))
      break;
  if (i > 0)
    usbHardwareEndpointWrite (bEP, abBulkBuf, i);
  return xTaskWoken;
}
```

Portierbarkeit:
Dieser Stack wurde für den Mikrocontroller LPC2148 geschrieben. Eine Portierung auf einen anderen Mikrocontroller oder eine andere USB-Schnittstelle ist wohl nicht an einem Nachmittag erledigt.

USBN9604 – Die leider abgekündigte universelle USB-Bridge

Internetseite: http://usbn2mc.berlios.de/

Benötigte Dateien (bzw. Dateistruktur):

Datei	Beschreibung
usb11spec.h	USB Datenstrukturen
usbn960xreg.h	Register des USBN9604
usbn960x.c / usbn960x.h	Steuerfunktionen für USBN9604
usbnapi.c / usbnapi.h	USB Zugriffs-API für eigene Firmware
usbn2mc.c / usbn2mc.h	Hardware-Zugriff auf USBN9604

http://www.ssalewski.de/AT90USB_firmware.html.de

Dieser Baustein ist ideal für kleine 8- und 32-Bit Prozessoren ohne eigene USB-Schnittstelle. Angesteuert werden kann er über eine Parallele 8-Bit Schnittstelle per DMA oder sogar über SPI. Viel Bauteile braucht die Schaltung Rund um den USBN9604 nicht. Ein weiterer Pluspunkt ist außerdem, dass es den Baustein als SO Gehäuse (Abstand ca. 1 mm) gibt, welches die Lötarbeiten drastisch vereinfacht. Für den USBN9604 gibt es viele Bibliotheken im Internet. Eine davon ist USBN2MC. Hinter der Bibliothek steckt die Idee, USB ganz einfach „integrierbar“ in eigene Anwendungen zu machen. Von USBN2MC gibt es zwei Versionen. Einmal die sogenannte „main“-Version, welche ein sehr bequemes Konfigurieren des USB-Geräts erlaubt, jedoch mit dem Speicher entsprechend verschwenderisch umgeht. Und die Version „tiny“. Hierbei kann einiges an Speicher- und Codeplatz eingespart werden, dafür muss man sich aber intensiver mit USB auskennen, um ein Gerät erfolgreich definieren zu können. Leider ist seit ca. einem Jahr der Baustein als „nicht mehr Empfohlen für neue Schaltungen“ markiert. Der USB9604 Baustein wird aber dennoch produziert. Es wurde wohl nur die Entwicklungsabteilung für diesen Controller aufgelöst. Da er aber noch sehr gut verfügbar ist (ebenso bei Reichelt & Co.) ist er meiner Meinung nach keiner Schaltung zu schade.

Konfiguration des eigenen Gerätes:
In der „main“-Version werden für die USB-Bibliothek ca. 6KB Programmspeicher und 500 Byte Arbeitsspeicher benötigt.

Angabe des Hersteller- und Produkt-ID:

```
USBNDeviceVendorID(0x1234);
USBNDeviceProductID(0x9876);
```

Definition der Namen, mit welchen sich das Gerät am Betriebssystem anmelden soll:

```
USBNDeviceManufacture(„My little firm“);
USBNDeviceProduct(„My device“);
USBNSerialNumber(„rev 1.0“);
```

Hinzufügen einer Konfiguration:

```
conf = USBNAddConfiguration();
USBNConfigurationName(„xyz“);
USBNConfigurationPower(50); // Power in mA
```

Hinzufühen eines Interfaces:

```
interf = USBNAddInterface(conf,0);w
USBNInterfaceName(conf,interf,“usblp“);
```

Und zuletzt - Definiton der Endpunkte:

```
USBNAddInEndpoint(conf,interf,1,1,BULK,64,&TransferReady);
USBNAddOutEndpoint(conf,interf,1,2,BULK,64,&PrintOnLCD);
```

Der letzte Parameter gibt den Callback an, welcher aufgerufen wird, wenn Daten vom PC empfangen bzw. versendet werden.

Einbindung in eine bestehende Softwarearchitektur:
Die Integration in eine bestehende Softwarearchitektur kann entweder per Interrupt oder Pollingbetrieb geschehen. Ist die Interruptleitung des USBN9604 mit dem Zielprozessor, auf dem der Stack läuft, verbunden, so kann in der Interruptroutine einfach die Funktion USBNInterrupt() aufgerufen werden. Alles weitere wird dann vom Stack erledigt. Ist keine Interruptleitung verbunden, kann die Funktion ebenso in einer Endlosschleife regelmäßig aufgerufen werden.

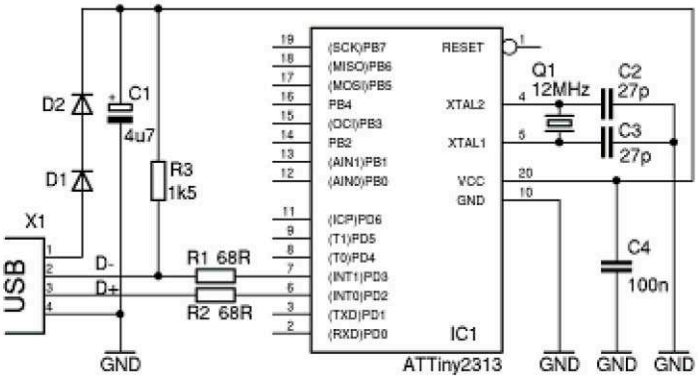
Portierbarkeit:
Der USB-Stack USBN2MC ist in reinem C geschrieben. Da die Anbindung entweder über einfache GPIO-Ports oder SPI erfolgt, kann USB an eine große Anzahl von Mikrocontrollern angebracht werden.

AT90USB1287 – Die Zukunft für AVR-Fans

Internetseite: http://www.ssalewski.de/AT90USB_firmware.html.de

Dieser Chip gehört zu einer kleinen neuen Produktfamilie von Atmel. Hier ist USB direkt als Einheit in den Prozessor integriert. Zwar bietet Atmel ebenso C-Beispiele für den Zugriff auf USB an, aber hier fehlt die freie Lizenz. Die AT90USB Firmware von Stefan Salewski hingegen ist dank GPL frei für jeden verfügbar und einsetzbar. Im Gesamten benötigt die USB-Logik ca. 130 KB Platz im Flashspeicher. Auf der Internetseite findet man ein fertiges funktionsfähiges Beispiel samt eigener Schaltung und Testprogramm für den PC.

AVR-USB – USB für Arme (Softwareimplentierung)



Internetseite: http://www.obdev.at/products/avrusb/index-de.html

Abb. 1: Schaltplan Low-Speed USB

An dieser Stelle ist folgende Lösung ebenfalls noch zu erwähnen. Es wird ein USB-Low-Speed Gerät mit einem einfachen AVR erzeugt. Dabei werden die Leitungen von USB direkt ausgewertet, was auch erklärt, wieso eine maximale Übertragungsgeschwindigkeit von 1,5 Mbit/s möglich ist. In Abbildung 1 sieht man, wie einfach die Schaltung ist. Für erste USB Tests ist dies sicherlich eine feine Schaltung. Ob man dieses Konstrukt jedoch für wichtigere Sachen einsetzen sollte, muss jeder selbst entscheiden.

Fazit

Dies war nur ein sehr kleiner Überblick über mögliche freie USB-Schnittstellen. Dennoch hoffe ich, dass der eine oder andere eine nützliche Information daraus ziehen kann.

Links

1. LPCUSB1 http://wiki.sikken.nl/index.php?title=LPCUSB
2. USBN2MC http://usbn2mc.berlios.de/
3. AT90USB1278 http://www.ssalewski.de/AT90USB_firmware.html.de
4 AVR-USB http://www.obdev.at/products/avrusb/index-de.html

Ansteuerung Touchscreen

mit einem AVR

Michael Hartmann <michael@speicherleck.de>

Motivation

Touchscreens werden schon seit vielen Jahren für die Steuerung von Geräten wie Karten- oder Spieleautomaten verwendet. Auch in PDAs und Tablet-PCs werden sie seit geraumer Zeit eingesetzt. Ihren großen Durchbruch verdanken sie aber Apples iPod Touch und dem iPhone.

Touchscreens bieten im Vergleich zu konventionellen Eingabegeräten deutliche Vorteile. Für den Benutzer ist die Steuerung per Finger intuitiver und einfacher als per Tasten. Aber auch der Entwickler profitiert von der Flexibilität eines Touchscreens. Will er nachträglich die Bedienung des Geräts verändern, so muss er nur die Software anpassen. Eine Anpassung der Hardware, wie sie z.B. beim Hinzufügen einer weiteren Taste notwendig wäre, entfällt. Zudem sind Touchscreens platzsparend, sehr stabil und besitzen je nach Qualität eine Lebensdauer von bis zu drei Millionen Berührungen.

Prinzip

Günstige Touchscreens sind in der Regel analog resistive Touchscreens mit 4-Wire-System. Diese Touchscreens bestehen aus zwei gegenüberliegenden, leitfähigen Schichten zwischen denen sich viele kleine Abstandshalter, so genannte Spacer-Dots befinden. Durch Berührung entsteht beim Druckpunkt ein Kontakt zwischen den beiden Schichten.

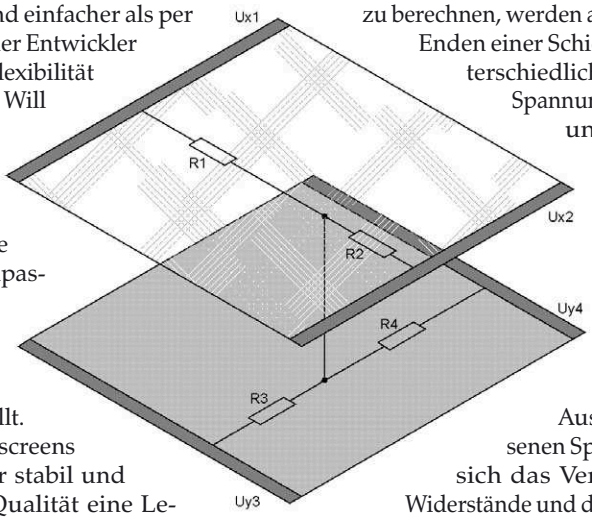


Abb.1:Schematischer Aufbau eines analog resistiven Touchscreens

Um eine Koordinate des Druckpunktes zu berechnen, werden an den beiden Enden einer Schicht zwei (unterschiedliche) bekannte Spannungen angelegt und an einem beliebigen Ende der anderen Schicht hochohmig die resultierende Spannung gemessen. Aus der gemessenen Spannung lässt sich das Verhältnis der Widerstände und damit die Position der Koordinate berechnen (siehe Abbildung 2 Rechenbeispiel).

Abb.2: Rechenbeispiel:

Legt man U_{x1} an eine Spannung von $+5V$ und an U_{x2} eine Spannung von $0V$ an, so ergibt sich für das Verhältnis der Widerstände:

$$\frac{R_1}{R_1+R_2} : U_{y3}=U_{y4}=U_{x2}+\frac{(U_{x1}-U_{x2})\cdot R_2}{R_1+R_2}$$

$$\rightarrow \frac{R_2}{R_1+R_2}=\frac{U_{y4}-U_{x2}}{U_{x1}-U_{x2}}$$

Misst man nun für U_{y4} einen Wert von $+2V$,ergibt sich daraus:

$$\frac{2V-0V}{5V-0V}=\frac{2}{5}=0{,}4$$

Das Verhältnis gibt die Position in x-Richtung an. Bei der Messung für die Position in y-Richtung wird an U_{y3} und U_{y4} Spannungen angelegt und an U_{x1} oder U_{x2} hochohmig gemessen.
Durch das Verhältnis der Widerstände entsteht an jedem Punkt eine unterschiedliche Spannung, die man zur Berechnung der x- und y-Koordinate nutzen kann.

Aufbau der Schaltung

Zur Steuerung wird ein Atmega32 verwendet. Die vier Kabel des Touchscreens werden mit den Pins ADC0 bis ADC3 verbunden. Dabei sind die PINs ADC0 und ADC1 mit der Schicht für die x-Koordinate, ADC2 und ADC3 mit der Schicht für die y-Koordinate verbunden. Im Datenblatt sollten sich für die vier Drähte des Touchscreens entsprechende Bezeichnungen wie YU (Y-Koordinate, oben), YD (Y-Koordinate, unten), XL (X-Koordinate, links) und XR (X-Koordinate, rechts) finden.

Gemessen wird an ADC0 und ADC2. Um einen definierten Wert zu messen, wenn keine Berührung vorliegt und daher keine Spannung anliegt, müssen diese beiden Pins noch hochohmig mit Masse verbunden werden. Ein Widerstand von je $1M\Omega$ sollte ausreichen, um in diesem Fall die Spannung auf $0V$ zu ziehen.

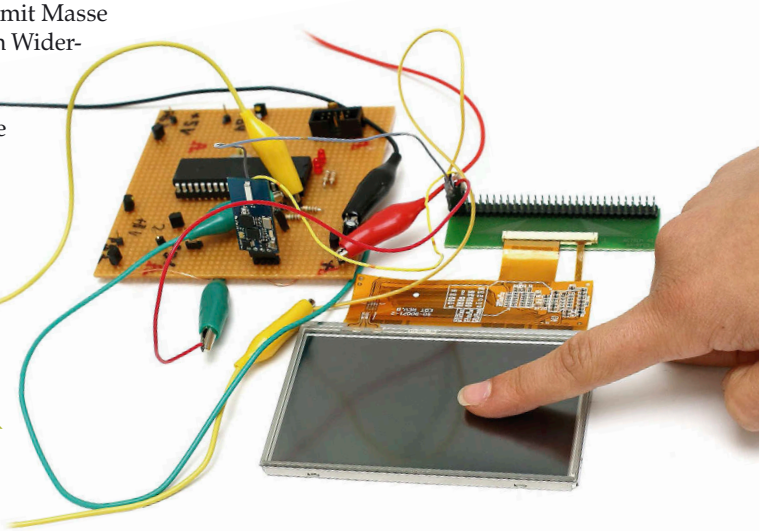
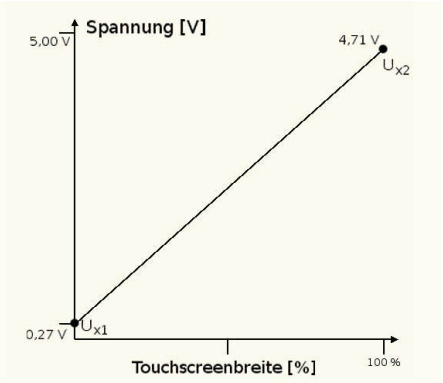


Abb.3 die schnell zusammengeschusterte Schaltung

Ansteuerung per Software

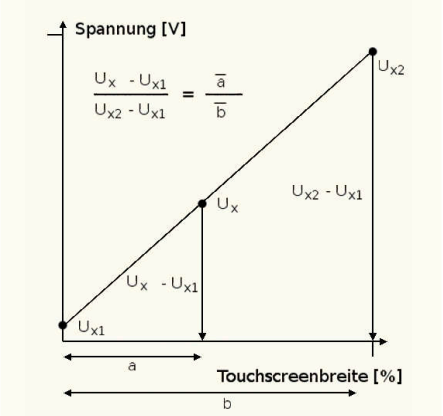
Per Software legt man nun an PD0 $+5V$ und PD1 $0V$ Spannung an und misst an PD2. Erweitert man Quelltext 1 noch um eine Debugging-Ausgabe und trägt die beiden Spannungen in Abhängigkeit von der Position an, erhält man für die x- und y-Koordinate je einen Graph (z.B. Graph 1 für die x-Koordinate).

Wie schon aus der Beispielrechnung, kann man aus den Graphen erkennen, dass die Spannung linear zum Abstand zunimmt. Aus den Graphen erkennt man nun auch die Extremwerte, demnach die maximale und minimale Spannung, die man für einen x-Wert (linker und rechter Rand; Ux1 und Ux2) und y-Wert (oberer und unterer Rand; Uy1 und Uy2) bekommt.



Die Spannung ist direkt proportionalo zum Abstand des Druckpunktes.

Der Quotient aus dem Unterschied zwischen der gemessenen Spannung und der Minimalspannung, und der Differenz zwischen Maximalspannung und Minimalspannung, entspricht dem Verhältnis vom Anfang zum Druckpunkt (a) und vom Anfang zum Ende des Touchpads (b). Dahinter verbirgt sich der zweite Strahlensatz. Multipliziert mit der Höhe bzw. Breite des Touchpads ergibt sich daraus die entsprechende Koordinate.



Mit ein bisschen Rechnen kommt man dann auch auf die richtige Koordinate.

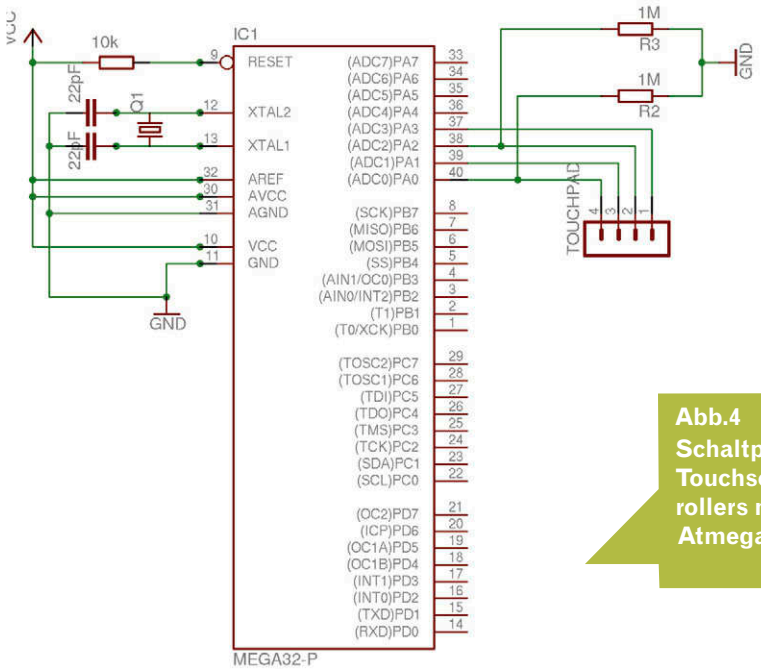


Abb.4
Schaltplan eines Touchscreencontrollers mit einem Atmega32.

Ausblick

Die Möglichkeiten sind hier natürlich längst nicht ausgeschöpft. Zum einen lässt sich die Software von Polling noch auf Interrupts umstellen, zum anderen kann man aufbauend auf der Erkennung einfacher Tastendrücke eine Bibliothek mit Funktionen für Buttons, Scrollbars, Drag `n Drop ... aufbauen. Den Vorstellungen sind kaum Grenzen gesetzt, nur mehrere Berührungen, wie sie Apples iPhone verarbeiten kann, unterstützen analog resistive Touchscreens leider nicht.

Damit dürfte dem eigenen Bastelvergnügen nichts mehr im Wege stehen.

Viel Spaß!

Quelltext 2:

```
[...]
#define HEIGHT 160
#define WIDTH  220

#define Uy1 0.78
#define Uy2 4.15
#define Ux1 0.27
#define Ux2 4.71

while(1)
{
  double Ux, Uy;
  int x,y;

  // Ux und Uy messen
  [...]

  x = (Ux-Ux1)/(Ux2-Ux1) * WIDTH ;
  y = (Uy-Uy1)/(Uy2-Uy1) * HEIGHT;

  _delay_ms(500);
}
```

Quelltext 1:

```
[...]
while(1)
{
  double Ux, Uy;

  /* x */
  DDRA  |= (1 << PA1); // PA0: +5V
  DDRA  |= (1 << PA0); // PA1:  0V
  PORTA |= (1 << PA0);
  _delay_ms(16);

  // adc(PORT) liefert gemessene Spannung in V
  Uy = adc(PA2);

  PORTA &= ~(1 << PA0); // PA0 und
  DDRA  &= ~(1 << PA0); // PA1 wieder
  DDRA  &= ~(1 << PA1); // Eingang

  /* y */
  DDRA  |= (1 << PA3); // PA2: +5V
  DDRA  |= (1 << PA2); // PA3:  0V
  PORTA |= (1 << PA2);
  _delay_ms(16);

  Ux = adc(PA0);

  PORTA &= ~(1 << PA2); // PA1 und
  DDRA  &= ~(1 << PA2); // PA2 wieder
  DDRA  &= ~(1 << PA3); // Eingang

  _delay_ms(500);
}
```

CapSense-Anwendungen

Prototyping von CapSense-Anwendungen mit dem CapToolKit

Raphael Wimmer <raphael.wimmer@ifi.lmu.de >

Kapazitive Sensoren

Am Physikalisch-Technischen Institut St.Petersburg stellte der russische Physikprofessor Leon Theremin im Jahre 1920 ein merkwürdiges Musikinstrument vor - einen Kasten aus dem zwei Antennen ragten. Indem er seine Hände diesen beiden Antennen näherte, konnte er Höhe und Lautstärke des erzeugten Tons ändern. Als Musikinstrument führt das Theremin inzwischen ein (sehr langlebiges) Schattendasein. Das zugrunde liegende Prinzip findet sich aber vielfach in heutigen kapazitiven Sensoren wieder.
Ein kapazitiver Sensor misst die Kapazität zwischen einer Antenne und seiner Umwelt. Je näher man ein geerdetes, leitendes Objekt an die Antenne bringt, desto höher wird die Kapazität des so gebildeten Kondensators. Auf diese Weise kann man den Abstand zwischen Sensor und Objekt messen. Dies funktioniert mit metallenen Objekten, Flüssigkeiten und vielen anderen Materialien. Auch menschliche Körper oder Hände können mit kapazitiven Sensoren getrackt werden.
Obwohl das zugrunde liegende Prinzip sehr einfach ist, sind kaum brauchbare Sensoren verfügbar. Wenn nur Berührung erkannt werden soll, sind eventuell die QProx-Sensoren nützlich. Sensoren für größere Abstände sind aber nicht verfügbar bzw. haben eine sehr niedrige Empfindlichkeit. Aus diesem Grund habe ich im Rahmen meiner Diplomarbeit ein Toolkit entwickelt, das es ermöglicht, einfach und schnell kapazitive Sensoren in eigene Prototypen zu integrieren. Dieses Toolkit - sinnigerweise CapToolKit genannt - ist Open-Source und umfasst Hardware, Firmware, Software und Dokumentation. Ich entwickle es inzwischen seit zwei Jahren weiter und verwende es für eigene Projekte.
Kernstück des CapToolKits ist das CapBoard (siehe Abbildung 1). Dieses dient als Controller für bis zu 8 Sensoren. Ein PIC 18f2550 steuert die Sensoren (siehe Abbildung 2) an und liest deren Werte (freuqenz-enkodiert). Die Werte werden im PIC gefiltert und aufbereitet. Außerdem steuert er auch die 8 frei verwendbaren I/O-Pins an. Die Kommunikation mit dem Host-PC erfolgt über RS-232 (TTL) und ein Klartext-Protokoll. Die Sensoren basieren auf dem „Theremin Vision II"-Design von Terry Fritz [1]. Kernstück des Sensors ist

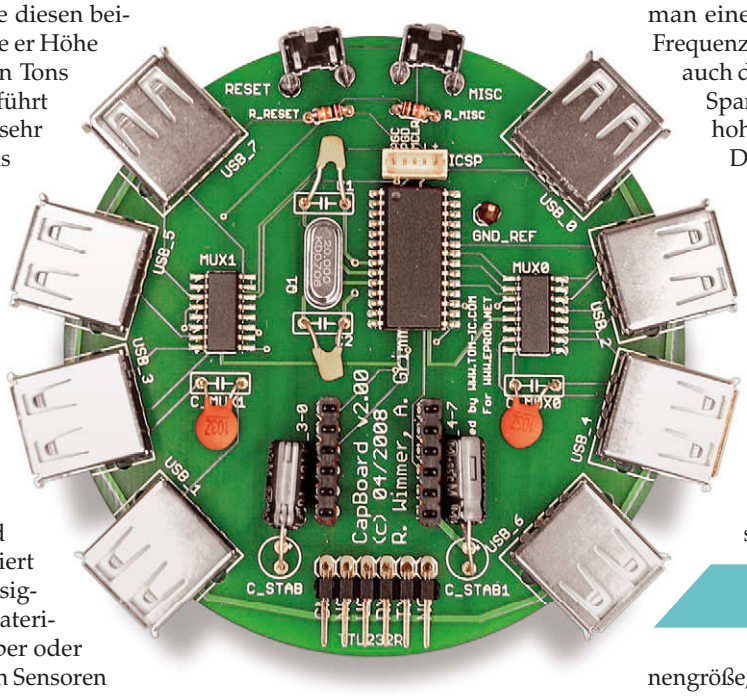
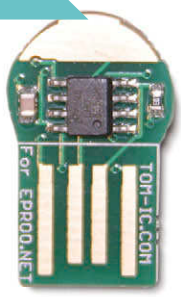


Abbildung 1: CapBoard

ein LMC555 Timer-IC. Dessen Frequenz hängt ab von der Kapazität, die die Antenne liefert. Der LMC555 wurde gewählt, weil die übrigen 555-Varianten nur bis 500 kHz

Abbildung 2: Sensor



gehen. Je höher aber die Frequenz, desto höher die Auflösung. Sie bewegt sich (je nach Abstand) zwischen einigen 100 kHz und 1,8 MHz. Die Frequenz wird durch einen Hardware-Counter im PIC gemessen. Ein generelles Problem von kapazitiven Sensoren ist ihre Empfindlichkeit gegenüber Störungen. Kapazitive Sensoren sind stark temperaturabhängig. Ein größeres Problem sind aber unerwünschte Kapazitäten, hervorgerufen z. B. durch vorbeigehende Personen oder große metallene Objekte. Hier ist eine Abschirmung des Sensors und der Antenne wichtig. Diese Abschirmung muss aber immer auf dem gleichen Spannungsniveau gehalten werden wie die Antenne, ohne dass diese elektrisch miteinander verbunden sind. Zu diesem Zweck verwendet man einen Spannungsfolger [2]. Da die Frequenz des Signals relativ hoch ist, muss auch der Operationsverstärker, der den Spannungsfolger implementiert, eine hohe Frequenz durchleiten. Um die Dämpfung des Signals möglichst niedrig zu halten wurde ein MAX4453 verwendet. Dieser hat sich in umfangreichen Tests bewährt. Einer der beiden integrierten Operationsverstärker treibt die Abschirmung der Antenne. Der andere Operationsverstärker schirmt die Signalleitung ab.

Welche Reichweite hat nun so ein Sensor? Dies ist schwer zu sagen, da diese von der Antennengröße, Umweltbedingungen und Messdauer abhängt. Standardmäßig misst das CapToolKit die Frequenz jedes Sensors 15 ms lang. Mit einer Antenne aus Weißblech von 40x40 mm lassen sich dann Bewegungen im Millimeter-Bereich auf ca. 20 cm Entfernung sicher erkennen. Aber auch ganz ohne zusätzliche Antenne lassen sich Bewegungen im Abstand von wenigen Zentimetern erkennen. Das Lötpad auf dem Sensor reicht hierfür als Antenne aus. Präzisere Richtwerte für die Reichweite der Sensoren werden in Kürze auf der CapToolKit-Webseite verfügbar sein.
Die 8 Sensor-Ports des CapBoard sind als USB-A-Buchsen ausgeführt. VCC und GND sind dabei wie bei USB belegt. Die anderen beiden Kontakte dienen dazu, den Sensor zu aktivieren, bzw. um seine Messwerte zu empfangen. Für die meisten Anwendungen empfiehlt es sich, ein Flachbandkabel an den Sensor zu löten und an dessen anderen Ende einen USB-A-Stecker anzubringen. Dadurch lässt sich der Sensor flexibel positionieren. Alternativ kann man auch einfach ein USB-Verlängerungskabel verwenden.

Einfacher Funktionstest

Zum Testen des CapBoard kann man wie folgt vorgehen:

- Einen oder mehrere Sensoren in beliebige Ports stecken.
- Das TTL232R-USB-Seriell-Kabel an die abgewinkelte Buchsenleiste des CapBoard stecken - schwarzer Draht (GND) links.
- Den USB-Stecker in den Rechner stecken, Gegebenenfalls die FTDI-Treiber installieren.
- Nun kann man z.B. die Python-API verwenden, um darauf zuzugreifen:

```
#!/usr/bin/python
import capboard
cb = capboard.CapBoard()
a = capboard.Analyzer(cb)
```

- Der Analyzer zeigt grafisch die Sensorwerte der einzelnen Kanäle an

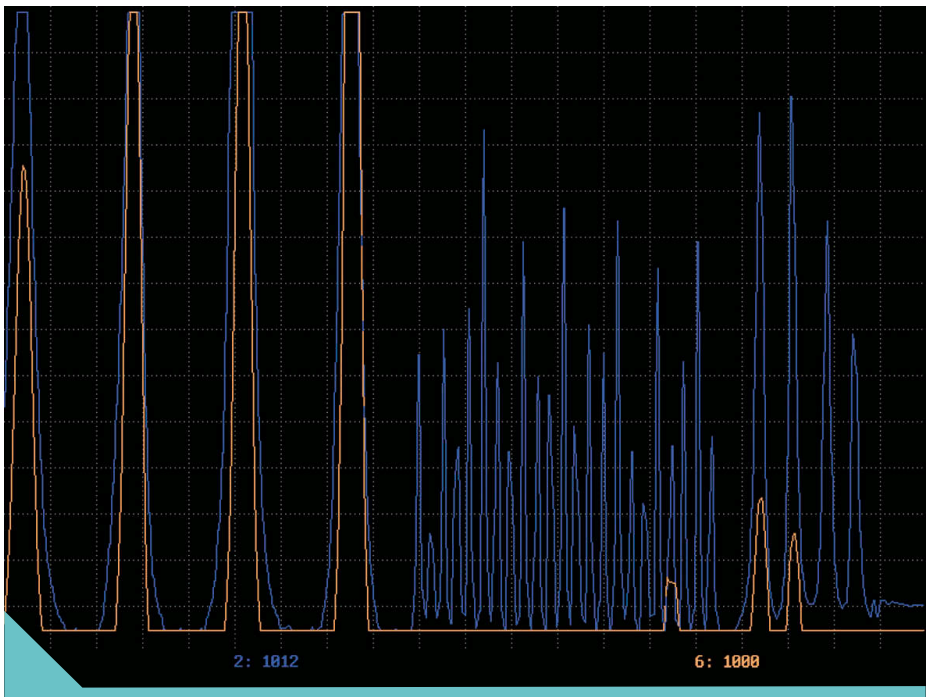


Abbildung 3: Analyzer zeigt grafisch die Sensorwerte der einzelnen Kanäle

Kommunikationsprotokoll

CapBoard erkennt beim Start automatisch die angeschlossenen Sensoren und gibt deren Messwerte aus. Die Ausgabe erfolgt im Format „VALUE, <device_id>,<channel>,<value>,<timestamp>“

Die einzelnen Komponenten:

`<device_id>`

dient dazu, mehrere angeschlossene CapBoards zu unterscheiden. Standardmäßig 0.

`<channel>`

gibt an, von welchem Sensor der Messwert kommt

`<value>`

ist der Messwert (16 bit).

`<timestamp>`

ist die verstrichene Zeit in Millisekunden seit Beginn der Messung.

Das CapBoard versteht auch einige einfache Befehle. Die komplette Dokumentation finden Sie auf der CTK-Website. Jeder Befehl wird durch eine kurze Statusmeldung bestätigt. Hier ein Beispiel:

```
   + 0 (aktiviere Sensor 0)
    REPLY,0,+,OK,80349
    - 1 (deaktiviere Sensor 1)
    REPLY,0,-,OK,82037
    F 3 (Filter 3, Lowpass)
    REPLY,0,F,OK,84766
    T 20 (setze Messdauer auf 20ms pro Sen-
sor)
    REPLY,0,T,OK,85928
    A (starte Messung)
    REPLY,0,A,OK,0
    VALUE,0,1,59705,8
    VALUE,0,1,59702,23
    VALUE,0,1,59701,36
    VALUE,0,1,59699,49
    ...
    Z (stoppe Messung)
    REPLY,0,Z,OK,2269
    S (speichere Einstellungen im EEPROM)
    REPLY,0,S,OK,22072
    ! (Reset)
```

APIs und Tools

Prinzipiell kann man auf das CapBoard mit jeder beliebigen Programmiersprache zugreifen - sofern diese von der seriellen Schnittstelle lesen kann. Wenn man nur schnell mal eine einfache Anwendung schreiben will, ist eine API aber ziemlich hilfreich. Für das CapToolKit gibt es eine Python-API und eine Java-/Processing-API. Diese kapseln die Kommunikation mit dem CapBoard und bieten einige Hilfsfunktionen. Mit der Python-API kann man sehr schnell z.B. einen Näherungssensor implementieren.

Zusätzlich gibt es z.B. Tool, das die Daten vom CapBoard über TCP/IP verschickt. Damit kann man auch von beliebigen Rechnern auf das CapBoard zugreifen.

```
#!/usr/bin/python
# simple proximity sensor using CapBoard
# see capsense.org
# Public Domain

import capboard, time

THRESHOLD = 5          # some value > 0

acb.reset()            # auto-detects sensors, starts CapBoard

baseline = cb.vals[0] # the current sensor value for channel 0
while True:
  if cb.vals[0] - baseline > THRESHOLD:
    print „sensor triggered!“
  else:
    print „“
  time.sleep(0.1)
```

Links & Bezug

Schaltpläne, Software, Dokumentation und weitere Informationen zum CapToolKit finden Sie auf CapSense.de [3].
Weiterführende Informationen zu den Grundprinzipien und Anwendungsmöglichkeiten von kapazitiven Sensoren finden Sie auf den Webseiten von Larry K. Baxter [4] und Joshua R. Smith [5]. Da schon bei den Vorgängerversionen des CapToolKit regelmäßig Anfragen kamen, ob ich denn auch fertige Hardware verkaufen würde, habe ich mich für CTK 2.0 mit Benedikt Sauter von Embedded-Projects.net zusammengetan. Ein Bausatz für das CapToolKit ist im Online-Shop von Embedded-Projects.net verfügbar.

Raphael Wimmer [6] beschäftigt sich seit 2005 mit kapazitiven Sensoren. Er promoviert am Lehrstuhl für Medieninformatik an der Ludwig-Maximilians-Universität München. In seinem Blog [7] schreibt er ab und zu auch über Embedded-Projekte.

1. ThereminVision II – Design von Terry Fritz http://thereminvision.com/version-2/TV-II-index.html
2. Spannungsfolger http://de.wikipedia.org/wiki/Spannungsfolger
3. Informationen zum CapToolKit http://capsense.de
4. Larry K. Baxter http://www.capsense.com/
5. Joshua R. Smith http://web.media.mit.edu/~jrs/
6. Raphael Wimmer http://www.medien.ifi.lmu.de/team/raphael.wimmer/
7. Blog von Raphael Wimmer http://my.opera.com/raphman/blog/

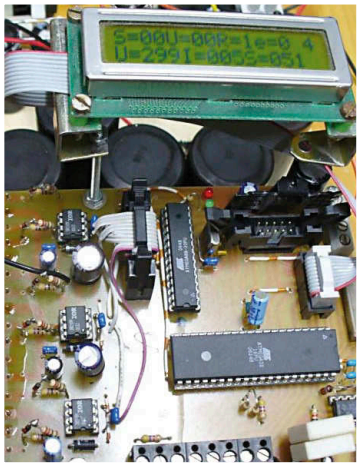
Artikelwettbewerb

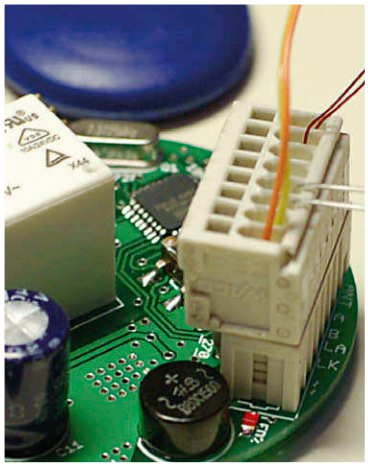
Mikrocontroller.net Artikelwettbewerb – die Ergebnisse

Andreas Schwarz <wettbewerb@mikrocontroller.net>

Der Wettbewerb

In der letzten Ausgabe des Embedded Projects Journal wurde der 1. Mikrocontroller.net Artikelwettbewerb angekündigt, jetzt ist er beendet. Abschließend kann man sagen, es war ein voller Erfolg – 26 fertige Artikel wurden eingereicht. Für die Gewinner standen Preise im Gesamtwert von über 800 € zur Verfügung. Die Ergebnisse des Wettbewerbs und die Artikel sind über http://www.mikrocontroller.net/wettbewerb erreichbar. Die 4 Hauptgewinner werden im Folgenden kurz vorgestellt.





Platz 1: USB IO-Expander

Der Gewinner des ersten Platzes und eines High-End-Multimeters von Voltcraft im Wert von 183 € ist **Michael Wittmann** mit seiner Beschreibung eines IO-Expanders für den USB-Port. Das Gerät basiert auf einem FTDI-Chip und einem AVR und besitzt eine Vielzahl von analogen und digitalen Ein- und A usgängen. Dazu gibt es eine in QT/C++ geschriebene PC-Bibliothek und GUI-Anwendung. Der Artikel wird in der nächsten Ausgabe des Embedded-Projects Journal abgedruckt.

Platz 2: Elektronischer Zauberwürfel

Von **André F.** stammt dieser elektronischer Nachbau des bekannten Rubik's Cube. Mit aufwendiger Holzkonstruktion, PC-Software, und zur Abwechslung mal einem HCS12-Mikrocontroller belegt dieses ausgefallene Projekt den zweiten Platz und gewinnt ein AVR32 Grasshopper-Board.

Platz 3: Frequenzumrichter mit Raumzeiger-modulation

Der dritte Platz geht an **Axel Jeromin** für die Beschreibung der Ansteuerung eines Drehstrom-Asynchronmotors. Von der Wellenformerzeugung per AVR bis zur Leistungselektronik ist alles Eigenbau und ausführlich beschrieben. Auch für diesen Platz gibt es ein AVR32 Grasshopper-Board.

Platz 4: RFID Türmodul

Mit einem AVR-basierten RFID-Türöffner belegt **Stefan Seegel** den dritten Platz und erhält einen Platinengutschein von Leiton im Wert von 50 €. Ein interessanter Punkt bei diesem Projekt ist, dass die AVR-Software in C++ geschrieben wurde.

[siehe Artikel Seite 4 - 6]

Und der Rest?

Neben den 4 Hauptgewinnern wurden noch viele weitere Artikel eingereicht, von denen 12 einen Preis gewonnen haben. Da dies der erste Artikelwettbewerb war, haben auch alle Nicht-Gewinner als Dankeschön für die Teilnahme einen 10 €-Platinengutschein erhalten. Danke an alle für die Teilnahme, und an alle Nicht-Teilnehmer oder Nicht-Gewinner, vielleicht klappt's ja beim nächsten Mal – denn eines ist sicher, das war nicht der letzte Wettbewerb.

TO OPEN YOUR SOURCE

JETZT VERÖFFENTLICHEN!

Dein Wissen - Dein Können - Dein Projekt

Dokumentiere deine Arbeit und veröffentliche im EP-JOURNAL!

DER BETRAG FÜR DIE COMMUNITY

embedded-projects.net
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Werdet aktiv!

Das Motto: Von der Community für die Community! Das Magazin ist ein Open-Source Projekt.

Falls Du Lust hast, Dich an der Zeitschrift durch einen Beitrag zu beteiligen, würden wir uns darüber sehr freuen. Schreibe Deine Idee an:
sauter@embedded-projects.net
Wir werden dann gemeinsam sehen, was wir daraus machen können.

Regelmäßig lesen!

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

1. Internetseite (Anmeldeformular gesponserte Abos)
http://www.embedded-projects.net/journal
2. Online-Shop für Journal (Preis 1 EUR + Versand)
http://www.eproo.de/journal

Sponsoren gesucht!

Damit wir weiterhin diese Zeitschrift für jeden frei bereitstellen können, suchen wir dringend Sponsoren für Werbe- und Stellenanzeigen. Bei Interesse meldet Euch bitte unter folgender Telefonnummer: 0821/5081581 oder sendet eine E-Mail an die oben genannte Adresse.

Bitte frei machen

Embedded Projects
Auf dem Kreuz 20
D-86152 Augsburg

Bitte in Druckbuchstaben gut lesbar ausfüllen, damit wir Ihre Bestellung bearbeiten können.

Name / Firma

Straße / Hausnummer

PLZ / Ort

E-Mail / Telefon / Fax

☐ **Bestellung des Embedded Projects Journal:**
Ich möchte jede zukünftige Ausgabe erhalten

☐ **Bestellung des Embedded Projects Journal für Hochschulen/ Ausbildungsbetriebe**
Wir möchten jede zukünftige Ausgabe zu Ausbildungszwecken bestellen
Anzahl der Hefte pro Ausgabe (zutreffendes bitte ankreuzen) ☐ 5 ☐ 10

☐ **Anforderung Infomaterial zur Anzeigenschaltung:**
Wir sind eine Firma und sind an Werbe- und Stellenanzeigen interessiert.
Bitte schicken Sie uns kostenlos und unverbindlich Infomaterial, Preisliste, ect. zur Anzeigenschaltung zu.

SPEZIALISTEN GESUCHT

Dein Spaß an High-tech
Deine Herausforderungen an Bits und Bytes
Deine Sendung mit der Maus - für Erwachsene

Du willst kreativ Entwicklungsprozesse mitgestalten -
werde Teil der Elektronikwelt.

Software-Ingenieure (m/w) Echtzeit und Embedded Systeme

Zur Verstärkung unseres Entwicklerteams in Königsbrunn
bei Augsburg suchen wir ab sofort deutschsprachige
Informatiker, Ingenieure oder Naturwissenschaftler mit
abgeschlossenem Hochschulstudium und Erfahrung
in der Entwicklung technischer Software.

Infos: http://www.ibv-augsburg.net/jobs

IBV - ECHTZEIT- UND EMBEDDED GMBH & CO. KG

Keltenstraße 2 D-86343 Königsbrunn
Fon +49 (0) 82 31.95 86 - 041 Fax +49 (0) 82 31.95 86 - 049
info@ibv-augsburg.net www.ibv-augsburg.net

ibv
Realtime is BLUE