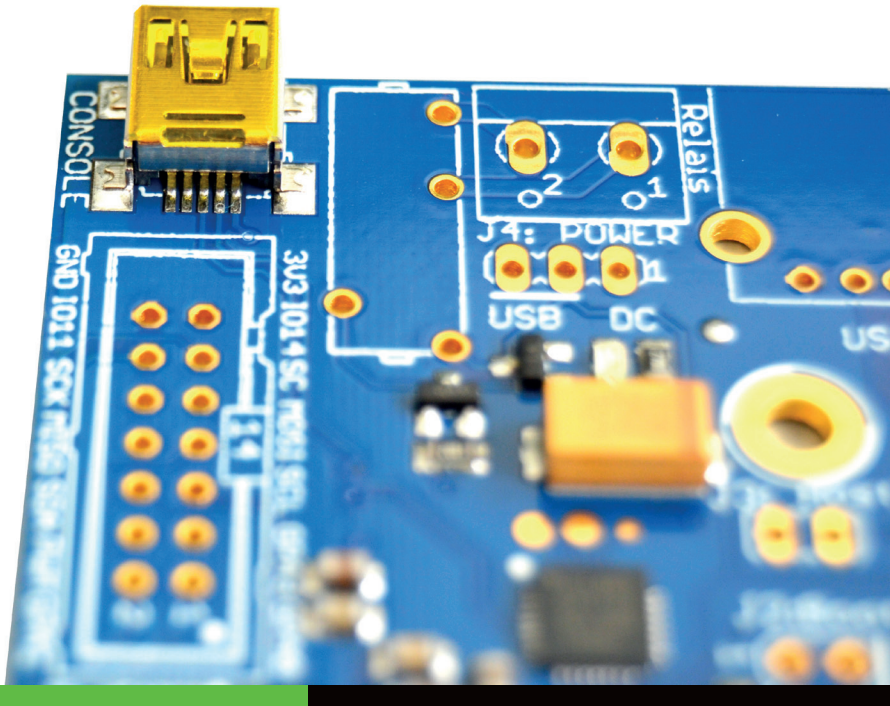
embedded projects JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Eine Open-Source Zeitschrift zum Mitmachen!

artikel zum nachbauen

[PROJECTS]

- Raspberry-Webradio
- High-Speed capture mit ATmega Timer
- Retrocomputing auf FPGA
- Batteriemonitor mit Dual Slope Wandler
- Vorankündigung USBprog 5.0



Einleitung

Ausgabe 03/2014

Embedded Projects Journal - Ausgabe No. 22

Einleitung

Viel Spaß beim Lesen wünscht Euch das embedded projects Team!

Diesmal etwas in eigener Sache:
Wir suchen Verstärkung in Augsburg. Eventuell kennt Ihr jemanden, der/die auf der Suche nach einer neuen Herausforderung ist?

- PHP Entwickler für unser Warenwirtschaftssystem in Augsburg

http://www.wawision.de/stellen/

Nächste Journal Ausgabe:
Aktuell könnten wir noch 1-2 Artikel für die Weihnnachstausgabe gebrauchen. Gerne könnt Ihr uns anschreiben.

Benedikt Sauter

sauter@embedded-projects.net

wawision.embedded-projects.net

waWision -
die Steuerzentrale für Ihre Firma

Verwal-tung	Plug & Play	Waren-eingang	Marke-ting
FiBu	Produk-tion	automa-tisches Lager	Online-Shops

EIN SYSTEM AUS EINER HAND

- keine Installation
- Betriebssystem unabhängig
- Standardhardware Plug & Play
- mitwachsend

DEMOVERSION

weitere Infos finden Sie auf unserer Internetseite

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

embedded projects GmbH
HARDWARE FOR PROJECTS

EURO CIRCUITS

Die europäische Referenz für PCB Prototypen und Kleinserien

Besuchen Sie uns zur Electronica in München, vom 11.-14. November, Halle A5 Stand 502

Tools zur Markteinführung Ihres Projekts - rechtzeitig und nach Budget

- Smartes Auftrags-Pooling senkt die Kosten
- Keine Einrichtkosten. Bestellung ab einer Leiterplatte.
- Sofortige Online-Bestellung - ohne bürokratische Verzögerungen
- Datenüberprüfung vor der Bestellung - keine Liefer-Verzögerungen
- Direkte Anzeige von DRC- und DFM-Fehlern - sofort lösbar
- Nutzen setzen und individuellen Text direkt am Bildschirm hinzufügen

Online Chat Support –
Unsere Experten zur Beantwortung Ihrer Fragen

PCB proto

spezieller Niedrigstpreis Protoypen-Service für Entwickler

- 1, 2 oder 5 Leiterplatten in 2, 3, 5 oder 7 Arbeitstagen
- 2 und 4 Lagen
- Vollständige Ausführung, elektrisch getestet

NEU

PCB PIXture ein grafischer Druck auf Ihrer Leiterplatte

Mehr Info in unser BLOG

STANDARD pool

Europas große Auswahl an Pooling-Optionen

- 1 - 16 Lagen. Wählen Sie aus über 700 definierten Multilayer Aufbaue
- Layout-Technologie bis 90µm
- Ab zwei Arbeitstagen

RF pool

Alle Pooling-Vorteile mit HF-Material

- Isola I-TERA und Rogers 4350B Material
- 2 - 4 Lagen, bis 100µm Technologie
- Ab 3 Arbeitstagen

IMS pool

Alu-Leiterplatten für (LED) Anwendungen mit hoher Wärmeableitung

- Einlagige Aluminiumkern-Leiterplatten
- Lötstopplack / Best.druck weiß / schwarz oder umgekehrt
- Ab 3 Arbeitstagen

BINDI pool

Asiatische Preise - Europäische Qualität

- 1 – 4 Lagen. Bis zu 12,5 m^2 (1600 halbe Eurokarten)
- Produziert in unserem neuen indischen Werk
- Gleiche Materialien, Technologien und Qualitätsstandards wie in unseren europäischen Werken.

www.eurocircuits.de

PCB Visualizer PCB Checker PCB Configurator

The new EAGLE has landed!

Version 7

jetzt
erhältlich

CS
EAGLE

Weitere Informationen unter www.cadsoft.de

Raspberry-Webradio

Michael Schwarz

Wer kennt das nicht, man möchte einfach nur Musik hören und egal auf welchen Radio Sender man schaltet, entweder hört man Werbung, Nachrichten, unnötige Ansagen irgendwelcher Moderatoren oder die gefühlte 317te Wiederholung der Charts. Glücklicherweise gibt es eine bessere und obendrein kostenlose Alternative: Internet-Radio. Mit SHOUTcast gibt es dazu auch ein Verzeichnis, das annähernd 50.000 Online-Radiosender listet. Doch da ich nicht immer meinen PC einschalten möchte um Musik zu hören, musste ein Standalone Gerät her. Und da ich gerade meinen Raspberry Pi erhalten hatte, schien das ein gutes Projekt zu sein um den Mini-Computer sinnvoll einzusetzen.

Features

Hier ist eine Liste der Hauptfeatures des Raspberry Webradios

- Abspielen von MP3 Shoutcast Streams über WLAN oder Ethernet
- Abspielen von MP3s von USB-Medien mit Unterstützung von Playlists
- Anzeige der Song Informationen (sowohl von MP3s als auch von Streams)
- Vollständige Shoutcast Unterstützung (Suchen von Stations, Top Stations, Stationen nach Genre, Zufällige Stations)
- Lieblingsstationen und Verwaltung der Favoriten
- Android App zur Fernsteuerung
- Schlummerfunktion

Hardware

Das Hauptprogramm läuft auf einem Raspberry Pi (Model B) mit 256 MB RAM, getaktet auf 700MHz. Die Interaktion mit dem Benutzer erfolgt über ein Blau-Weiß Grafik-Display mit KS0108 Controller und einer Auflösung von 128x64 Pixel, sowie über einen Drehencoder und 6 Tasten.

Um den Raspberry Pi zu entlasten, und da dieser nicht besonders viele I/O Pins zur Verfügung stellt, stehen ihm zwei Mikrocontroller als „Co-Prozessoren“ zur Verfügung. Ein ATMega32 dient als Bridge zwischen Raspberry und GLCD und ein ATTiny2313 wird verwendet um den Drehencoder und die Tasten auszuwerten.

Funktion	Hardware
Prozessor	Raspberry Pi (Model B, 256MB RAM)
LCD Controller	ATMega32 @ 14,3181 MHz
I/O Controller	ATTiny2313 @ 8 MHz
Eingabe	Drehencoder und 6 Taster

Tabelle 1: Technische Daten

Stromversorgung

Der Raspberry Pi wird über den GPIO Pin Header mit 5V versorgt. Der ATMega32 und das Display werden ebenfalls mit 5V versorgt. Die 3,3V für den ATTiny2313 stellt der Raspberry Pi an seinen GPIO Ports zur Verfügung [1].

Der Stromverbrauch des Raspberry liegt laut Spezifikation bei 700mA, das Gerät sollte also mit einem Netzteil versorgt werden das 1A liefern kann, vor allem wenn ein WLAN Stick verwendet wird. Mein Webradio wird durch ein USB Handy-Ladegerät (5V @ 1A) mit Strom versorgt.

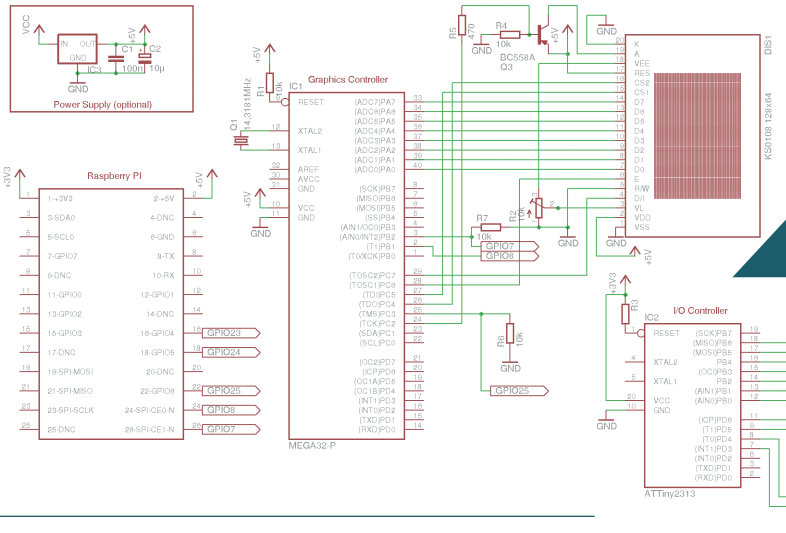


Abb. 1: Komplett-ter Schalt-plan mit RPi, LCD Controller und I/O Controller

GLCD Bridge

Da der Raspberry nur wenige Pins hat und diese nur 3,3V Pegel haben dient der ATMega32 sowohl als Pegel-Converter, Portexpander und Display Treiber. Die Stromversorgung erfolgt über die 5V mit welchen auch der Raspberry Pi gespeist wird. Die 3,3V der I/O Pins reichen für den ATMega um es als High-Pegel zu erkennen, eine Kommunikation in die andere Richtung findet nicht statt. Die einzige Aufgabe des Mikrocontrollers ist es, die vom Raspberry erhaltenen Daten an das Display weiterzuleiten. Die Kommunikation erfolgt über ein Clock und ein Data Signal. Bei jedem high-low Übergang wird das Bit vom Data-Pin übernommen und in einem Ringbuffer gespeichert, damit keine Daten verloren gehen. Der Inhalt des Ringbuffer wird dann regelmäßig an das Display gesendet. Ein vollständiges Bild hat eine Grö-

ße von 1024 Bytes (128x64 Bit). Danach werden die Daten für das nächste Bild gesendet. Überprüfungen finden nicht statt und es werden ausschließlich die Rohdaten des Bildes gesendet um den Overhead so gering wie möglich zu halten und damit eine hohe Bildaufbaugeschwindigkeit zu erhalten. Das Format der Daten kann in Tabelle 2 erkannt werden.

Jedes gesendete Byte enthält 8 vertikale Pixel, es werden also durch das Senden von 128 Bytes immer 8 Zeilen auf das Display geschrieben.

Bei einer Taktung des ATMegas mit 14,3181MHz wird damit eine maximale Refresh-Rate von 12 Bildern/Sekunde erreicht. Im Produktiveinsatz hat sich eine Refreshrate von ungefähr 6,5 Bildern/Sekunde als völlig ausreichend herausgestellt und sich als sehr stabil bewährt.

Zusätzlich zur Grafik steuert der ATMega auch noch die Hintergrundbeleuchtung des Displays. Diese ist so geschaltet, dass beim Einschalten des Gerätes die Beleuchtung aktiviert wird und bei Bedarf vom Raspberry deaktiviert werden kann.

I/O Bridge

Da der ATMega32 mit den Aufgaben als Grafik-Bridge schon ziemlich ausgelastet ist handhabt ein weiter Mikrocontroller, ein ATTiny2313, die Eingabe über den Drehencoder und die 6 Taster. Kommuniziert wird bei beim ATMega ebenfalls über ein Data- und Clock-Signal, diesmal jedoch in die andere Richtung. Der ATTiny agiert nur als Slave und sendet die Zustände der Taster und des Encoders an den Raspberry wenn dieser ein Clock-Signal anlegt.

Byte 0-127	Byte 128-255	...	Byte 895-1023
Bilddaten 1.-8. Zeile	Bilddaten 9.-16. Zeile	Bilddaten 17. - 120. Zeile	Bilddaten 121.-128. Zeile

Tabelle 2: Datenformat für LCD Controller

Bedeutung	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1: gedrückt	0	0	Taster 5	Taster 4	Taster 3	Taster 2	Taster 1	Taster 0
Byte 2: lange gedrückt	0	0	Taster 5	Taster 4	Taster 3	Taster 2	Taster 1	Taster 0
Byte 3: Drehencoder	Anzahl der Steps des Drehencoders (8-Bit singed char)							

Tabelle 3: Datenformat für LCD Controller

Eine vollständige Übertragung ist 3 Byte (24 Bit) lang und enthält für jeden Taster ein Bit ob er seit der letzten Abfrage gedrückt wurde, weiters für jeden Taster ein Bit ob er seit der letzten Abfrage für längere Zeit gedrückt wurde und schließlich noch ein Byte das die Anzahl der Steps des Drehencoders seit der letzten Abfrage enthält.

Erstes Byte, wenn das entsprechende Bit gesetzt ist, wurde der Taster seit der letzten Abfrage gedrückt, sonst nicht.

Zweites Byte, wenn das entsprechende Bit gesetzt ist, wurde der Taster seit der letzten Abfrage für längere Zeit gedrückt, sonst nicht.

Drittes Byte, die Anzahl der Steps die der Drehencoder seit der letzten Abfrage gedreht wurde. Positive Werte bedeuten eine Drehung nach rechts, negative eine Drehung nach links.

Die Entprellroutinen für die Taster und die Auswerte-Routine für den Drehencoder sind in diesem Controller implementiert, sodass sich der Raspberry Pi nicht mehr darum kümmern muss.

Internet

Für den Zugriff auf das Internet kann man entweder die Ethernet Buchse des Raspberry Pi verwenden oder einen WLAN Stick an einen der USB Ports anschließen. Hier ist der Netgear N150 Wireless USB Microadapter empfehlenswert, da er sowohl klein als auch recht günstig ist und es für den Raspberry Pi einen Treiber gibt. Eine Anleitung zur Installations des Treibers gibt es hier: http://pkern.at/1342/netgear-n150-wireless-usb-microadapter-auf-dem-raspberry-pi

Die Konfiguration des WLAN (Suche, Verbinden und Authentifizierung) kann vollständig über das Gerät erfolgen, sodass es auch möglich ist das Radio mitzunehmen und mit einem anderen WLAN Hotspot zu verbinden.

Software

Die Software des Webradios besteht aus mehreren Teilen und kann in folgende Kategorien eingeteilt werden:

Software

- Hauptprogramm das am Raspberry Pi läuft („Firmware“), programmiert in C
- Stream Player und USB-Player, programmiert in C
- Android Fernsteuerungsprogramm, programmiert in Java
- Installer zur erstmaligen Einrichtung, programmiert in Bash

Interfaces

- Kleine Hilfstools zur Kommunikation mit dem Betriebssystem, programmiert in Bash
- Firmware für die Bridges (I/O und GLCD), programmiert in C
- Mobile Webseite zur Fernsteuerung, programmiert in PHP und JavaScript

Developer Tools

- Simulator, programmiert in C++
- Bild-Konverter für Firmware, programmiert in Bash
- Font-Creator von F. Maximilian Thiele (apetech)
- Der Zusammenhang der wichtigsten Komponenten ist in Abbildung 2 zu sehen

Software Module

Graue Rechtecke sind Hardware-Komponenten (Display, Taster, Drehencoder), die Software-Hardware Bridges sind blau eingezeichnet, die gelben Elemente sind Software-Interfaces (Bash-Scripts), Software Komponenten sind weiß und in grün gehalten sind Betriebssystem-Anwendungen.

Software

Die Software für die beiden Mikrocontroller ist in C geschrieben und kann mit einem Aufruf von

```
make hex
```

erstellt werden bzw. mit

```
make flash
```

erstellt und geflasht werden. Vor dem Flashen sollte man allerdings noch die Zeile mit dem Programmer im Makefile anpassen. Die Fuses können durch den Aufruf von

```
make fuses
```

richtig gesetzt werden. Für den ATMega32 bedeutet das: Taktfrequenz auf Quarz und JTAG deaktivieren (damit PORTC verwendet werden kann). Das Entspricht den Fuse-Werten Low-Fuse 0xde und High-Fuse 0xd9). Am ATTiny stellen die Fuses die interne Taktfrequenz auf 8MHz. Die Fuse-Werte sind Low-Fuse 0xe4, High-Fuse 0xdf und Extended-Fuse 0xff.

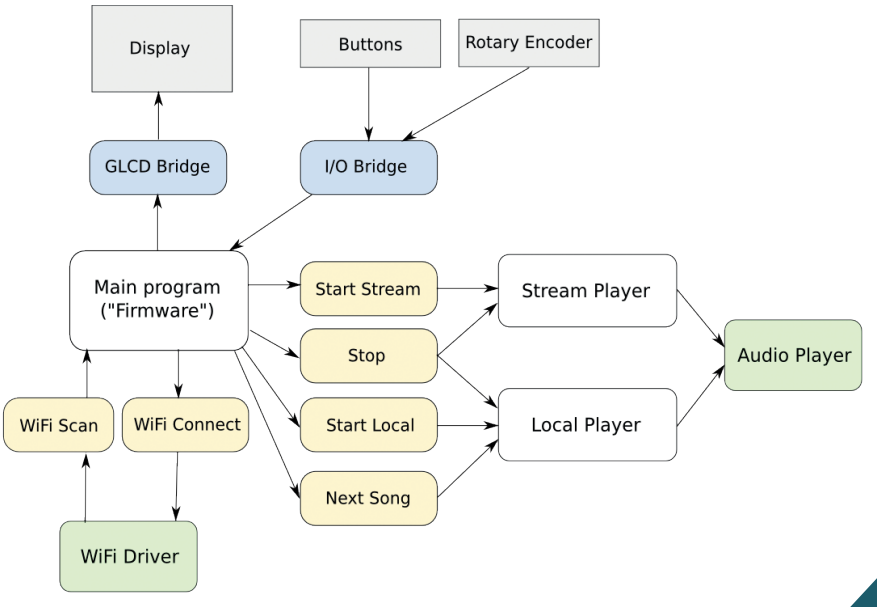


Abb. 2: Blockschaltbild des kompletten Hard- und Softwareaufbau

Die Taktfrequenzen der Mikrocontroller sind nicht kritisch, für den ATMega32 sollte alles im Bereich von 12MHz bis 16MHz funktionieren. Schlimmstenfalls muss der Wert GLCD_DELAY in der glcd.h erhöht werden. Die Firmware für den ATMega32 enthält auch das Boot-Logo, welches mit dem convert Script aus einem PNG erzeugt wurde. Das Bild kann durch ein beliebiges anderes Bild ausgetauscht werden, es muss dann nur der Aufruf zu GLCDD_XBMDraw_P(data, x, y, width, height) in der main.c angepasst werden (außer das neue Bild ist ebenfalls 59x56 Pixel groß).

Firmware

Die Firmware ist das Herzstück des Projekts. Sie ist zuständig für die Ein- und Ausgabe, das Starten und Stoppen von Musik und für die Verwaltung. In regelmäßigen Intervallen werden die Benutzereingaben von der I/O Bridge geholt, auf diese reagiert und der neue Display Inhalt an die Grafik-Bridge gesendet.

Möchte der User Musik abspielen, entscheidet die Firmware ob es ein lokales File oder ein Stream ist, und delegiert die Aufgabe dann entweder an den USB-Player oder den Stream-Player.

Alle Einstellungen die konfiguriert werden können sind in der default.conf zu finden. Die Optionen sollten selbsterklärend sein, es sollte jedoch nicht notwendig sein hier manuell etwas anzupassen (ausgenommen der Eintrag des Shoutcast API Keys).

Zum Abspielen von Musik gibt es zwei getrennte Applikationen: den Local-Player zum Abspielen von lokalen Dateien (von USB-Stick oder externen Festplatte) und den Stream-Player zum Wiedergeben von SHOUTcast Streams.

Die Konfigurationsdatei play.conf wird allerdings von beiden Programmen verwendet. In ihr kann der verwendete Player eingestellt werden (es funktionieren jedoch nur Player, welche die Musik Daten von einer Pipe verarbeiten können) sowie die Größe des Buffers den der Player verwendet.

Simulator

Da es relativ umständlich ist, die Software jedesmal auf den Raspberry zu übertragen und zu testen existiert auch ein Simulator, der die komplette Hardware simuliert. Der Simulator ist in C++ geschrieben mit wxWidgets als GUI Framework. Der Simulator ist nur unter Linux getestet, da die Software auch unter Linux entwickelt wurde. Das liegt auch nahe, da auf dem Raspberry Pi ein modifiziertes Debian läuft und es nie geplant war, dass die Software auf einer anderen Plattform als auf dem Raspberry Pi läuft.

Um das Programm im Simulator zu testen muss der Quellcode nur mit dem Debugging Makefile kompiliert werden. Dieses setzt das Simulationsmakro und linkt die wiringPi Bibliothek nicht zum Programm. Weiter müssen noch die default.conf folgendermaßen angepasst werden:

Abb. 3: Simulator nach dem Starten

```
[hardware]

lcd=sim

io=sim
```

Die Hardware Zugriffe werden dadurch auf Datei-Operationen umgeleitet. Diese Dateien sind die Kommunikationsschnittstelle zum Simulator.

Beim Starten des Simulators kann man als Parameter den Pfad zur Software mitgeben, damit diese sofort geladen wird. Der Simulator selbst kann im Grunde nicht mehr als das Display, den Drehencoder und die Tasten zu simulieren, das ist aber ausreichend, um die komplette Funktionalität (bis auf die Hintergrundbeleuchtung) der Firmware zu testen. Die Oberfläche des Simulators ist zwar nicht schön, aber zweckmäßig.

Die Buttons in der ersten Zeile simulieren den Drehencoder (linke, Ok, rechts), die zweite und dritte Zeile enthalten die Buttons und die letzte Zeile ermöglicht das Simulieren des Gedrückthaltens der Tasten 1-4 für längere Zeit. Ein Klick auf den Screenshot Button speichert das aktuelle Bild im Ordner des Simulators als „lcd.png“.

Android App

Da der Raspberry Pi noch genügend Ressourcen hat, läuft im Hintergrund ein Webserver, der es erlaubt das Gerät auch über ein Android-Smartphone zu steuern. Die App dafür ist ebenfalls auf der Downloadseite des Projekts zu finden.

Das Smartphone muss natürlich im gleichen Netzwerk sein (WLAN), damit das Webradio gefunden werden kann. Um das Gerät zu finden, braucht man nicht einmal die IP-Adresse wissen, da in der App ein Zeroconf-Client implementiert wurde, der den Raspberry automatisch findet (der Zeroconf Name ist raspberrypi.local).

Wurde das Raspberry Webradio gefunden, kann man ihn der App sehen, welches Lied gerade abgespielt wird (inklusive Bild des Interpreten, falls vorhanden), Stationen aus der Favoriten-Liste und vom USB-Medium abspielen sowie neue SHOUTcast Stationen und SHOUTcast Streams hinzufügen.

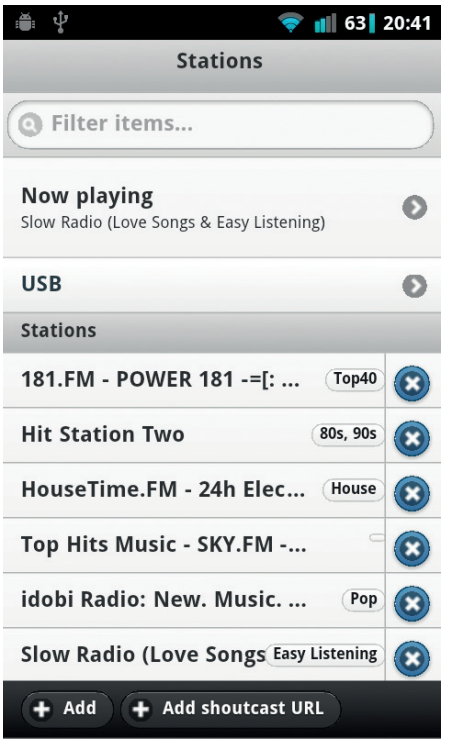


Abb. 4: Android App

Um das App zu installieren, kopiert man die apk Datei einfach auf das Smartphone und öffnet sie mit einem beliebigen Datei-Explorer auf dem Handy. Die einzigen Voraussetzungen für die App sind Android 2.2 und aufwärts sowie WLAN, da das Smartphone natürlich im gleichen Netzwerk sein muss wie das Webradio.

SHOUTcast Unterstützung

Die Unterstützung für die SHOUTcast API kann alles was man braucht

- Suchen nach Stationen
- Anzeigen der Stationen nach Genre
- Liste der Top Stationen
- Zufällige Stationen (wenn man sich gerne überraschen lässt)

Leider benötigt man für die SHOUTcast API einen Developer Key der nicht so einfach zu bekommen ist (und den man auch nicht veröffentlichen darf).

Wer die oben erwähnten Features also verwenden möchte, kann entweder versuchen einen SHOUTcast Developer Key zu bekommen, oder man verwendet einfach einen der Keys die im Internet umherschwirren... Das Abspielen von SHOUTcast Streams funktioniert natürlich auch ohne Developer Key.

Der Developer Key muss in der Konfigurations-Datei default.conf überall eingetragen werden, wo [DevKey] vorkommt.

Installation

Die Firmware ist entwickelt und getestet auf Raspbian Wheezy. Die Installation von Raspbian wird auf der Raspberry Pi Seite beschrieben. Die einfachste Art das Raspberry Webradio aufzusetzen ist mit dem automatischen Installer. Hierfür kopiert man alle Ordner, die man aus dem SVN ausgecheckt hat auf den Raspberry Pi und führt den Installer wie unten beschrieben aus.

Eine andere, noch einfachere Möglichkeit ist, ein fertiges Image zu verwenden. Dieses braucht nur auf eine SD-Karte mit mindestens 2GB geschrieben werden. Das Image (Stand: 11.07.2013) ist zu finden auf misc0110.net/download/webradio-image.rar. Auf die SD-Karte schreiben kann man es (nach dem Entpacken) mit

```
sudo dd if=/pfad/zum/image.
img of=/dev/sdX bs=4M
```

wobei /dev/sdX durch den Pfad zur SD Karte ersetzt werden muss. Man kann statt dd auch das graphische Tool Image-Writer (sudo apt-get install usb-image-writer) verwenden.

Die Login Daten für das Image sind User: pi und Passwort: raspberry. Der Installer ist ein Bash-Script das alle erforderlichen Dateien auf dem Raspberry Pi installiert. Damit der Installer funktioniert, wird Internetzugang benötigt (am besten über die LAN Buchse) und er muss als root ausgeführt werden.

Folgende Schritte werden vom Installer durchgeführt

- Installation aller notwendigen Softwarepakete, falls diese noch nicht installiert sind (avahi-daemon, mpg123, id3v2, usbmount, lighttpd + php5, wiringPi, libcurl, libxml2)

- Einrichten des lighttpd Servers und der mobilen Webseite
- Kompilieren der Firmware, des USB-Players und des Streaming-Players
- Kopieren aller Dateien in die richtigen Ordner (/home/pi wird als Installationsordner verwendet) und setzen der Berechtigungen
- Standard Konfiguration einspielen

Der Installer wird mit einem Aufruf von

```
sudo ./installer.sh
```

gestartet.

Nach der Installation muss - sofern die Firmware auf der Hardware ausgeführt wird, und nicht im Simulator - die default.conf bearbeitet werden:

```
[hardware]
lcd=hardware
io=hardware
```

Sollte der Webserver nicht starten, kann dies durch ein

```
sudo chown pi:pi /var/log/lighttpd/
```

behoben werden.

Autostart und Autologin

Um die Firmware automatisch beim Starten des Raspberrys zu starten müssen zwei Dateien angepasst werden. Zuerst editiert man die Datei /etc/inittab (z.B. mit sudo nano /etc/inittab).

Man sucht die Zeile

```
1:2345:respawn:/sbin/getty
--noclear 38400 tty1
```

und kommentiert sie aus, indem man # davor setzt. Die Zeile sieht danach so aus:

```
#1:2345:respawn:/sbin/getty
--noclear 38400 tty1
```

Unter der geänderten Zeile fügt man folgende neue Zeile ein

```
1:2345:respawn:/bin/login -f
pi tty1 </dev/tty1 >/dev/tty1 2>&1
```

Dies sorgt dafür, dass man automatisch als User pi angemeldet wird wenn das Betriebssystem startet.

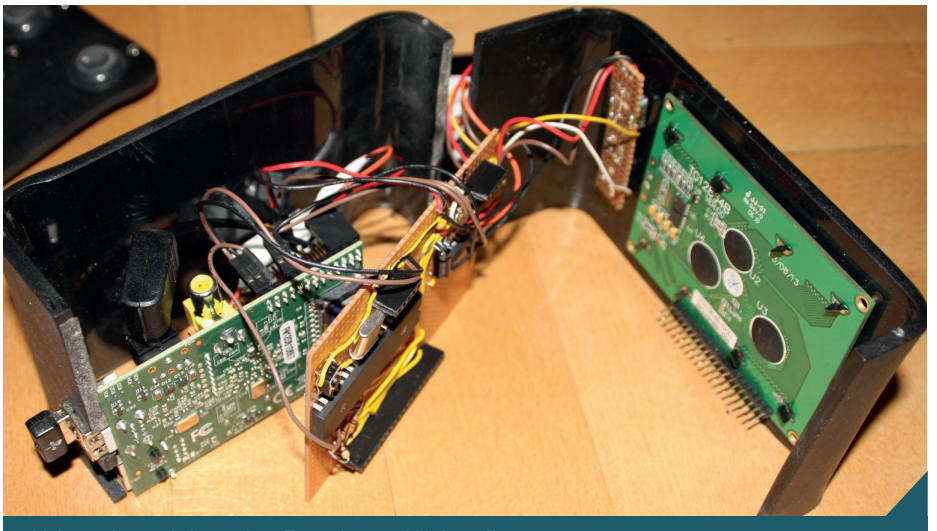


Abb. 5: Innenleben des Raspberry-Webradio

Danach muss noch die Datei /etc/profile geändert werden (z.B. mit sudo nano /etc/profile). Hier fügt man einfach am Ende der Datei folgende zwei Zeilen ein

```
cd /home/pi
./firmware
```

Dadurch startet die Firmware automatisch wenn man den Raspberry Pi hochfährt.

Troubleshooting

Ich kann keine Musik hören

Es kann sein, dass die Standard Audio Ausgabe auf dem HDMI Port erfolgt. Umschalten auf die Audio Buchse kann man mit

```
sudo amixer cset numid=3 1
```

Eine weitere Möglichkeit ist, dass entweder die Bash Scripts nicht dort liegen, wie es in der default.conf angegeben ist, oder die Bash-Scripts nicht ausführbar sind. Wenn die Software allerdings mit dem Installer Script installiert wurde sollte das nicht der Fall sein.

Es wird eine falsche Uhrzeit angezeigt

Das ist dann der Fall, wenn die Zeitzone nicht richtig eingestellt ist. Das Ändern der Zeitzone ist nicht schwer und sieht z.B. für Wien so aus

```
sudo mv /etc/localtime /etc/
localtime.old

sudo cp /usr/share/zoneinfo/
Europe/Vienna /etc/localtime
```

Der Drehencoder funktioniert nicht/ verhält sich seltsam

Möglicherweise sind die beiden Ausgänge vertauscht. Tauschen der Pins PD5 und PD6 sollte helfen (entweder in Software oder Hardware)

Im Simulator funktioniert das Abspielen von Streams/MP3s nicht

Die Firmware geht davon aus, das alle notwendigen Dateien in /home/pi legen. Abhilfe schafft entweder das Anlegen des Ordners /home/pi und hineinkopieren der relevanten Dateien in diesen Ordner oder das Ändern der Pfade in der default.conf

Das Display zeigt Streifen an/ist verschoben/unleserlich nachdem ich das Audio-Kabel ein/ausgesteckt habe

Das ist ein bekannter Bug, der manchmal auftritt, den ich aber noch nicht lokalisieren konnte. Beheben kann man ihn im Moment nur, indem man das Gerät herunterfährt (Settings -> Shutdown), vom Strom trennt und wieder mit Strom versorgt. Dadurch wird der Display Controller zurückgesetzt und man sollte wieder ein normales Bild haben.

Die Android App zeigt nur Loading... Please wait... an

Wenn ein Neustart der App nichts hilft (manchmal wird der Raspberry nicht gefunden), dann kann es daran liegen, dass das Handy nicht im gleichen Netzwerk ist (wenn es z.B. über 3G im Internet ist und nicht mit WLAN) oder dass der Raspberry im Moment keine aktive Internet Verbindung hat. Der zweite Fall kann dann auftreten, wenn der Raspberry über WLAN verbunden ist, und für längere keine Zeit kein Traffic verursacht wurde. Abhilfe schafft hier das manuelle Starten einer Internet Radio Station, danach sollte es auch über die Android App wieder funktionieren.

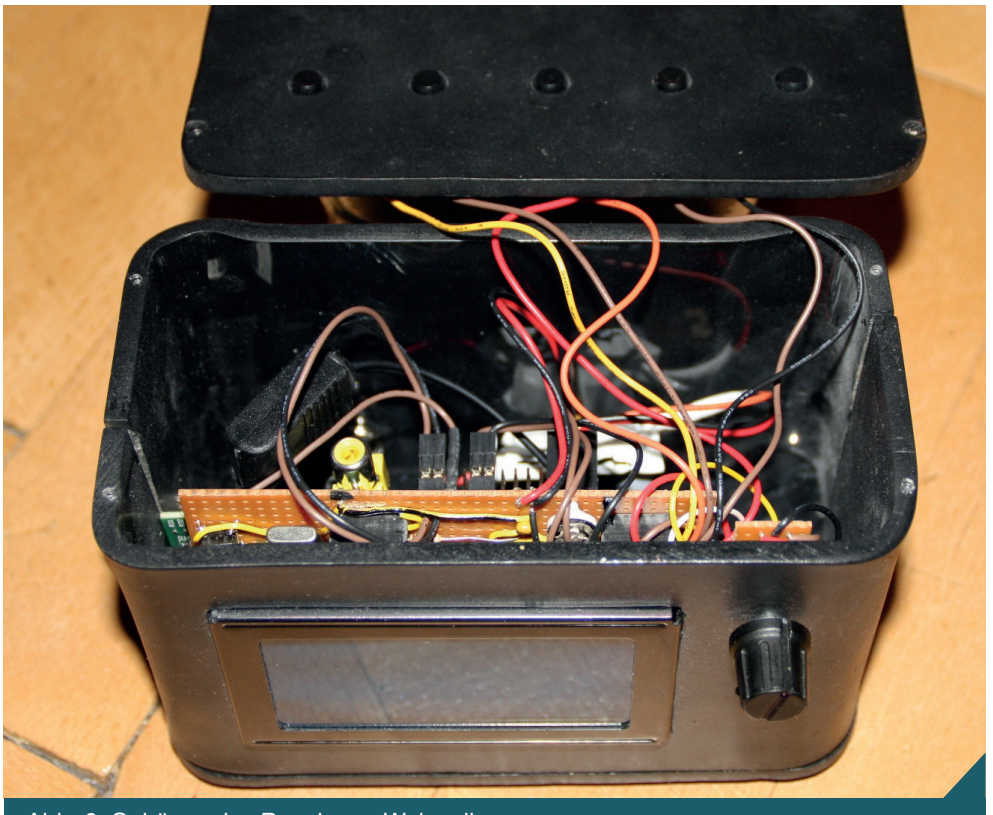


Abb. 6: Gehäuse des Raspberry-Webradio

Ausblick

Wie jeder Entwickler weiß, ist ein Produkt nie fertig. Das gilt auch bei diesem Projekt und so habe ich bereits einige Ideen für zukünftige Software Updates.

- Abspielen von Musik aus YouTube Videos (bereits in der Testphase)
- Eine Weckerfunktion
- Erweitern des Simulators um die Simulation der Hintergrundbeleuchtung und einer Debug Console
- Unterstützung weiterer Musik Formate (ogg und wma)
- Bessere USB Wiedergabe (Anzeige der Restzeit/Gesamtzeit, voriges Lied, zufälliges Lied, Wiederholen, ...)
- Aufnahme der Internet Streams als MP3
- Erweiterung der Android App um mehr Funktionen

Um immer auf dem neuesten Stand zu sein, empfiehlt es sich hin und wieder die Projektseite zu besuchen, bei größeren Updates werde ich natürlich auch diesen Artikel aktualisieren. Sollte das Projekt jemand nachbauen, wünsche ich viel Spaß dabei und ich würde mich über Fotos und/oder Videos freuen.

□

Downloads & Links

[1] Anleitung	http://goo.gl/vqunFn
[2] Sourcecode - Projektseite	http://goo.gl/3k08Da
[3] Image (Raspberry-Webradio Image)	http://goo.gl/aaQ2Be

High-Speed capture mit ATmega Timer

Michael Dreher

Einleitung

Die Timer der ATmega Mikrocontrollerreihe bieten Unterstützung für vielfältige Aufgaben wie PWM, zyklische IRQ Aufrufe oder als Zeitbasis für Messungen. In diesem Artikel geht es um die Input Capture Funktionalität, welche sehr präzise Zeitmessungen ermöglicht, da Programmlaufzeiten und Verzögerungen für die Genauigkeit keine Rolle spielen. Bei einem Flankenwechsel eines externen Signals an einem ICPn Pin speichert der Timer seinen aktuellen Wert im ICRn Register.

Das hier beschriebene Projekt zeichnet die High- und Low-Zeiten eines externen Binärsignals auf. Nach der Aufzeichnung wird es analysiert und ausgegeben. Der Focus des Artikels liegt auf der Lösung der bei der Umsetzung auftretenden Probleme.

Der Begriff „Highspeed Capture“ ist natürlich relativ zur Taktfrequenz des ATmega zu sehen. Das realisierte Programm speichert bis zu 3600 Flankenwechsel mit einer Auflösung von 62,5 ns. Die Zeitspanne zwischen zwei Flanken muss zwischen 3 µs und 65535 µs liegen.

Unterschied zwischen Messung und Protokollerkennung

Fernbedienungen modulieren ihr Ausgangssignal typischerweise mit Frequenzen zwischen 36 und 40 kHz. Es gibt Bauelemente (TSOP1736, 38, 40) um diese Modulation beim Empfang für eine bessere Signalempfindlichkeit auszunutzen (Fremdlichtunterdrückung) und das Signal zu demodulieren

Der Mikrocontroller bekommt vom TSOP17xx ein bereits aufbereitetes Signal, welches Pulsbreiten von mindestens 200 µs enthält (ein kompletter Burst aus z.B. 10 einzelnen Perioden), was gegenüber einer Einzel-Pulsbreite von 6 µs bei einem modulierten Signal sehr viel einfacher zu verarbeiten ist.

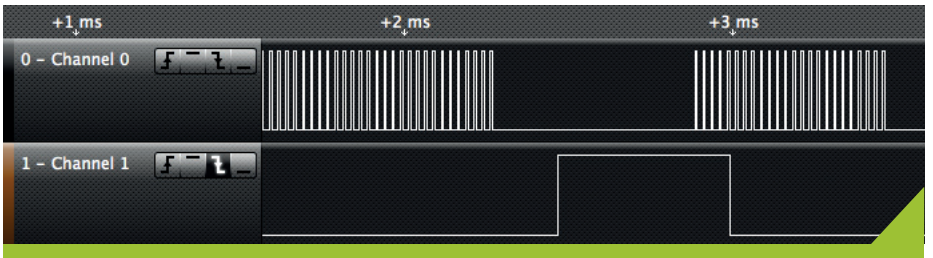


Abb. 1: Vergleich der Signale von SDP8600 mit TSOP1738

In Abbildung 1 ein Vergleich der Signale von SDP8600 (Channel 0) mit TSOP1738 (Channel 1, Ausgang negiert):

Bei der Analyse und Vermessung von Signalen wird eine sehr viel schnellere und exaktere Erfassung benötigt als bei der Verarbeitung von bekannten Signalen.

Als Untersuchungsobjekt wurde ein IR-Signal gewählt, in der Praxis kann aber jedes beliebige Signal vermessen werden, sofern der Abstand zwischen zwei Flanken größer als ca. 2,8 µs ist. Bei der IR-Messung werden die Modulationsfrequenz und das Puls/Pause Verhältniss bestimmt und zusätzlich die Zeitwerte ausgegeben. Es geht explizit nicht darum ein bestimmtes IR Protokoll (wie z.B. RC5) zu implementieren.

Der Aufbau kann auch dazu dienen eine bestehende Implementierung eines IR-Senders zu überprüfen, quasi als einfacher Ein-Kanal „Logik Analyzer“.

Hardwareaufbau

Der Hardwareaufbau des IR-Analysators ist denkbar einfach:

- Ein ATmega-Board. Getestete Arduino Boards siehe Tabelle unten.
- Ein Honeywell SDP8600 Infrarot Optoschmitt Detektor (wichtig: im Gegensatz zu TSOP17xx ohne eingebauten Demodulator)
- Ein Stützkondensator von 0,1 µF parallel zur Spannungsversorgung des SPD8600

Der SDP8600 wird an GND, +5V und Pin ICP1 des ATmega angeschlossen, da der 16-Bit Timer1 verwendet wird.

Beim Arduino Mega 2560 Board sind die ICP Pins ICP1 und ICP3 nicht nach außen geführt, daher muss auf ICP4 oder ICP5 ausgewichen werden.

Zum Debuggen habe ich einen Ausgangs-Pin des ATmega an einen Logic Analyzer angeschlossen. Er wird vom Programm gesetzt, wenn ein Flankenwechsel erkannt wird und zurückgesetzt, wenn das Abspeichern des Wertes beendet ist. Diesen Pin bezeichne ich im weiteren als ‚Dbg Pin‘.

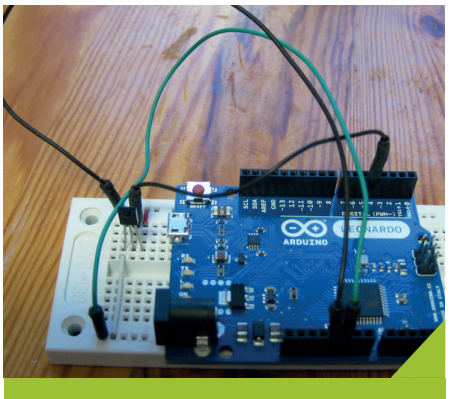


Abb. 2: Hardwareaufbau

Auflösung und Wertebereich der Messungen

Für die Zeitmessung wird der 16-Bit Timer1 verwendet. Dieser hat eine Auflösung von bis zu 1/F_CPU. Beim Arduino entspricht dies 1/16 MHz = 62,5 ns. Für den maximalen Timer-Wert 65535 ergibt dies eine Zeitspanne von 4,096 ms. Längere Zeiträume können bei dieser Auflösung nicht direkt mit dem Hardware-Timer gemessen werden. Für einige Anwendungsfälle ist dieser Zeitraum zu kurz.

Es gibt mehrere Möglichkeiten den Zeitraum zu erweitern:

- Timer Prescaler (Vorteiler) für das Taktsignal des Timers verwenden. Über die Prescaler Bits CS12 bis CS10 in Register TCCR1B kann man das System-Taksignal der CPU um folgende Faktoren runterteilen: 1, 8, 64, 256, 1024
- Software-Überlaufbehandlung des Timers

Der Nachteil beim Einsatz des Prescalers ist, dass sich die Auflösung dadurch verringert. Um Zeiträume von bis zu 65 ms erfassen zu können, müsste ein Prescaler-Wert von 64 verwendet werden. Dadurch würde die Auflösung von 62,5 ns auf 4 µs sinken. Damit könnte der genannte Zweck, Zeiten in der Größenordnung von 8 µs mit einer Genauigkeit von +-2% genau zu vermessen um die Modulationsfrequenz zu bestimmen, nicht erreicht werden. Aus diesem Grund verwendet dieses Projekt die Überlaufbehandlung.

Die Überlaufbehandlung inkrementiert einen Software-Zähler, wenn der Wertebereich des Timers überschritten wird. Damit können beide Ziele erreicht werden: eine hochauflösende Messung mit einem großen Messbereich.

Vergleich Logic Analyzer Messung mit ATmega Messung

Nachfolgende Grafik zeigt die Erfassung eines Pulses, mit einer ON-Zeit von 4,5 µs und einer OFF-Zeit von 21,6 µs. Die Messung wurde parallel mit dem ATmega und einem Logic-Analyzer durchgeführt und die Kurven übereinandergelegt. Die orangene Linie ‚LA' zeigt die Logic-Analyzer Messung, die schwarze Linie ‚AVR' die ATmega Messung. Die Zeitskala ist in Sekunden, die Grafik zeigt einen Zeitraum von 40 µs. Wie man sieht, weichen die beiden Kurven kaum voneinander ab:

Die blaue ‚Dbg' Kurve zeigt die Rückmeldung vom ATmega Programm an den Logic Analyzer. Hier kann man die Reaktionszeit (0,79 µs) und die Programmlaufzeit für die Bearbeitung der Flanken (2,38 µs) ablesen. Durch den aktivierten noise canceler (Bit ICNC1 in TCCR1B) ist diese Kurve gegenüber den anderen beiden um 4 CPU Zyklen (0,25 µs) nach rechts verschoben.

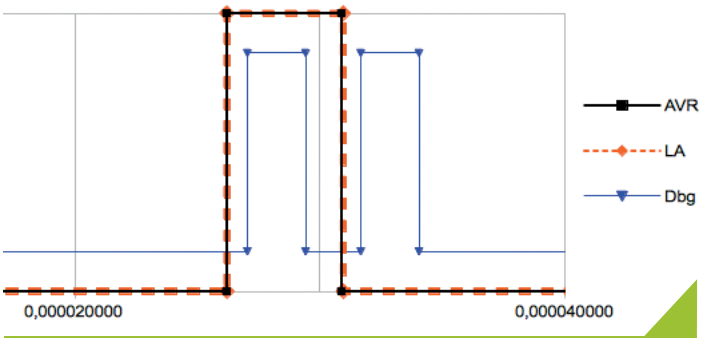


Abb. 3: Erfassung mit Logikanalysator und ATmega

Warten auf Flankenwechsel mit ISR

Als erster Ansatz wurde eine ISR (Interrupt Service Routine) verwendet, welche bei jedem Flankenwechsel getriggert wurde. Dies war zu langsam um mit dem Signal schritthalten zu können und wurde daher verworfen.

Der System Overhead beim Aufruf einer ISR in C ist recht hoch, da Register gesichert und wieder restauriert werden müssen (5 Register, insgesamt ca. 13 Zyklen, siehe diesen Artikel). Hinzu kommen noch 4 Zyklen, bis zur Ausführung der ersten Instruction der ISR, außerdem können andere IRQs (z.B. der Arduino Timer0 IRQ oder USB IRQs vom ATmega32U4) ins Gehege kommen und die Erfassung ausbremsen.

Für die Speicherung von Zuständen müssen globale Variablen verwendet werden, welche dann nicht in Registern gehalten werden können und jedes mal neu geladen werden. Man kann zwar auch globale Variablen in Register legen, diese stehen dann aber im restlichen Programm nicht mehr zur freien Verfügung.

Für die Variablen auf die von der ISR und vom normalen Programm zugegriffen wird, muss der type qualifier "volatile" verwendet werden, was dem Compiler einiger Optimierungsmöglichkeiten beraubt und den Code unter Umständen nochmal deutlich langsamer macht.

Warten auf Flankenwechsel mit Polling

Die schnellere Variante der Flankenabfrage ist polling des Timer Flag Registers. Der grobe Ablauf des Programms ist sehr einfach. Im Listing ist der Pseude-Code für die komplette Erfassungs-Schleife dargestellt.

Die Schleife für das Warten auf den Flankenwechsel besteht aus nur 3 Assembler Instructions, was deutlich kürzer und schneller ist als die ISR Variante.

```
IRQs deaktivieren
While(Ende nicht erreicht)
{
  Warten auf einen Flankenwechsel (Bit ICF1 in TIFR1)
  Auslesen des Input Capture Wertes (ICR1)
  Speichern der Zeitdifferenz zum letzten Ereignis in einem Array
  Trigger für Timer invertieren (TCCR1B ^= _BV(ICES1))
}
IRQs erlauben
```

Listing 1: Pseudocode für die Erfassungs-Schleife

Der C-Code:

```
uint8_t tifr;

while(! (tifr = (TIFR1 &
(_BV(ICF1) | _BV(OCF1A))))) {

}
```

Wird vom avr-gcc übersetzt zu

```
loop:
in      r18, TIFR1
andi    r18, _BV(ICF1) | _BV(OCF1A)
breq    loop
```

In dieser Schleife passiert folgendes:

Es wird abgefragt, ob ein Ereignis aufgetreten ist, für welches ein Timer-Wert durch den Input Capture gespeichert wurde (TIFR1 & (_BV(ICF1))

Es wird abgefragt, ob der Output Compare erkannt hat, dass ein Differenz-Überlauf passiert ist (TIFR1 & (_BV(OCF1A)), Details siehe Abschnitt Keine Angst vor Überläufen

Der Wert der UND Verknüpfung wird gespeichert um ihn weiter unten weiterzuverwenden und TIFR nicht erneut auslesen zu müssen.

Der im Arduino Leonardo verbaute ATmega32U4 verwendet keinen externen USB-seriell Wandler sondern implementiert direkt das USB Protokoll. Reagiert ein USB Device einige Zeit nicht auf die Anfragen des USB Hosts, wird es abgekoppelt. Dies passiert z.B. wenn man die IRQ Bearbeitung für einige Zeit abschaltet wie es in diesem Projekt gemacht wird. Ein Workaround ist, die IRQs nur so kurz wie möglich zu sperren, d.h. erst nachdem die erste Flanke erkannt wurde. Eine stabile Lösung ist dies aber nicht, daher ist für diese Anwendung der ATmega32U4 nicht zu empfehlen.

Zeitmessung

Um die Zeitdifferenz zwischen zwei Ereignissen zu messen gibt es mehrere Möglichkeiten:

- Nach jedem Ereignis den Timer auf 0 zurücksetzen
- Den Timer weiterlaufen lassen und immer die Differenz zum letzten Timer-Wert zu bilden.

Die erste Variante hört sich erst einmal verlockend einfach an. Zwischen dem Erkennen einer Flanke und dem Zurücksetzen des Timers vergehen aber einige µs. Der nächste Timer Messwert wäre genau um diesen Versatz zu klein. Sofern dieser Versatz bekannt ist, kann man ihn herausrechnen, bei einem Test lag er bei mir bei 30 CPU Zyklen. Mit jeder Programm- oder Compileränderungen muss dieser Versatz aber neu bestimmt werden.

Die zweite Variante ist eleganter, da sie den Timer durchlaufen lässt und die Differenz zwischen zwei Timer-Werten bildet. Ein Versatz wird dadurch komplett vermieden.

Keine Angst vor Überläufen

Es gibt unterschiedliche Überläufe zu betrachten:

- Überlauf des Timer Wertes TCNT1. Dies ist kein Problem und wird durch die vorzeichenlose Modulo 2^16 Berechnung der Differenz kompensiert, Rechenbeispiel siehe unten
- Überlauf der Differenz zwischen zwei Zeitpunkten, d.h. wenn die Differenz zwischen zwei Flanken größer als 65535 ist (Zeitspanne größer als 4,096 ms). Diese Überlaufe müssen mitgezählt werden (in ovlCnt) und als obere Bits der Zeitdifferenz gespeichert werden.

Die Differenz zwischen T(7) und T(6) muss über die Differenz von Timer1 und ovlCnt bestimmt werden:

T(7) - T(6) = (9836 - 43600) MOD 2^16 + 2^16 * 2 (ovlCnt) = 31772 + 2^16 * 2 = 162844

Die Differenz-Berechnung erfolgt Modulo 2^16, da der Datentyp uint16_t verwendet wird. Diese Art der Berechnung ist etwas gewöhnungsbedürftig, wenn der Subtrahend größer ist als der Minuend und daher ein negatives Ergebnis erwartet wird. Daher hier eine genauere Ausführung in hexadezimaler Schreibweise (damit man die 16-Bit Wortgrenze erkennen kann):

T(7) - T(6) = 0x266C – 0xAA50 = 0xFFFF7C1C (als uint32_t)

T(7) - T(6) = 0xFFFF7C1C MOD 2^16 = 0x7C1C (als uint16_t) = 31772

Um Differenzwerte größer als 65535 zu verarbeiten, muss man die Überläufe der Timer-Differenz mitzählen, was in der Variable ovlcnt passiert. Zwischen den Zeitpunkten T(7) und T(6) hat Timer1 den Wert 43600 (Timer-Wert bei T(6)) noch zweimal erreicht, d.h. die Differenz ist 2 mal übergelaufen und der Wert um 2*2^16 größer:

T(7) - T(6) = (0x266C – 0xAA50) + 2*2^16 = 0x7C1C + 2 * 0x10000 = 027C1Cx = 162844

Zeitpunkt	Signal	CLKs	Timer1	T(n) - T(n-1)	ovlCnt
T(5)	0->1	10000	10000		
T(6)	1->0	43600	43600	33600	0
T(7)	0->1	206444	9836	162844	2

Tabelle 1: Beispiele für erfassste Timer-Werte

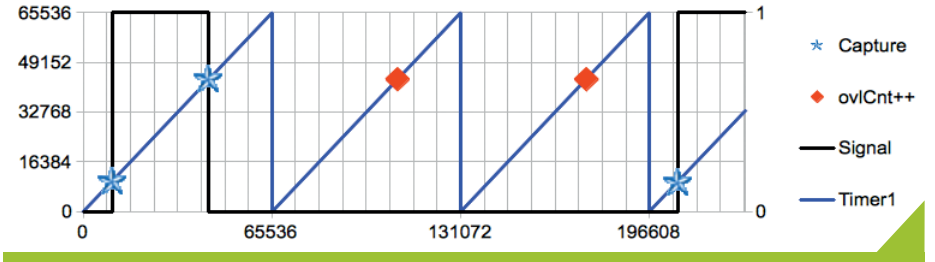


Abb. 4: Verlauf von Signal, Timer1 und die Erfassungs-Zeitpunkte (Capture)

In Abbildung 4 ist eine Grafik, welche den Verlauf von Timer1, die Erfassungs-Zeitpunkte (Capture) und das Hochzählen von ovlCnt darstellt

‚Capture' bezeichnet die Erfassungszeitpunkte T(5) bis T(7), bei welchen Timer1 aufgrund der Flanke von ‚Signal' den aktuellen Wert speichert.

Damit man nicht „von Hand" vergleichen muß, ob der Wert 43600 zwischen zwei Flanken nochmal vorkam (die Zeitpunkte mit ‚ovlCnt++' in der Grafik markiert sind), kann man den Output Compare Wert des Timers setzen und dies die

Hardware des ATmega machen lassen. Wenn der Timer diesen Wert erreicht, wird das Output Compare Flag OCF1A im Timer Flag Register TIFR1 gesetzt und daraufhin die Variable ovlCnt erhöht. Nach jeder Erfassung eines Timer-Wertes wird OCF1A auf den erfassten Timer-Wert gesetzt.

Speicherung der ermittelten Werte

Die Zeit-Differenzwerte werden in einem Array gespeichert. Das für die Wertspeicherung verfügbare RAM hängt davon ab, wie viel Speicher für andere Zwecke benötigt wird (Stack, globale Variablen). Bei Arduino 1.0.3 mit ATmega328P (2048 Byte RAM) bleibt Platz für ca. 870 16-Bit Werte, mit dem ATmega32U4 kommt man auf ca. 1050 und mit dem ATmega2560 kommt man auf 3600 Werte. Die Puffergröße im Programm ist relativ zur Arbeitsspeichergröße RAMEND festgelegt, d.h. bei größerem Speicher erweitert sich automatisch der Puffer. Der vom Programm belegte Speicher wird durch projectRamUsage festgelegt:

```
const uint16_t projectRamUsage = 380 + 64;

const uint16_t bufSize ((RAMEND - 0x100 - projectRamUsage) / sizeof(uint16_t));
```

Wenn das Programm erweitert wird, oder in einer anderen Umgebung eingesetzt wird, muss die Konstante projectRamUsage angepasst werden, da es sonst abstürzt.

Bei einer Auflösung von 62,5 ns können in einem uint16_t Wert maximal Zeiten von 4 ms gespeichert werden. Um den Messbereich zu erweitern greift man zu einem Trick. Bei kurzen Zeitspannen ist die absolute zeitliche Auflösung sehr wichtig, damit die prozentuale Ungenauigkeit nicht zu groß wird. Um bei langen Zeitspannen dieselbe prozentuale Genauigkeit zu erhalten, benötigt man eine geringere absolute zeitliche Auflösung. Dies ist vergleichbar mit der Messbereichsumschaltung bei einem Digitalmultimeter.

Dies könnte man dadurch erreichen, dass man Fließkommazahlen verwendet, welche durch die getrennte Verarbeitung des Exponenten eine automatische Messbereichsumschaltung eingebaut haben. Diese haben aber den Nachteil, dass die Verarbeitung ohne Hardwareunterstützung auf einem Mikrocontroller sehr langsam ist und sie viel Platz benötigten (4 Byte für einfache oder sogar 8 Byte für doppelte Genauigkeit pro Wert).

Eine Alternative ist eine spezielle Codierung, des 16-Bit Wertes. Das höchste Bit wird als Umschalter zwischen zwei Messbereichen verwendet:

- Messbereich 1: Wenn Bit 2^15=0 ist, dann enthält der Wert die Differenz-Zeit ohne weitere Skalierung. Damit sind alle Werte zwischen 0 und 32767 ohne Lücken direkt darstellbar (zeitliche Auflösung 62,5 ns, maximal 2,048 ms).
- Messbereich 2: Wenn Bit 2^15 den Wert 1 hat, enthält die Zahl die Differenz-Zeit mit einer zeitlichen Auflösung von 2 µs. Der Wertebereich und die zeitliche Auflösung sind also jeweils um den Faktor 32 verschoben. Dadurch sind Zeiten bis zu 65,535 ms darstellbar.

Der Verschiebungsfaktor zwischen den beiden Messbereichen kann über die Konstante RANGE_EXTENSION_BITS (Standardwert: 4 Bits) geändert werden. Es gilt:

```
Verschiebungsfaktor = 2^(RANGE_EXTENSION_BITS + 1)
```

Um die Zahlen schneller verarbeiten zu können und weniger Schiebebefehle bei der Ablage zu benötigen werden im Messbereich 2 die Bits 2^16 bis 2^19 (der Überlauf-Zähler) in den unteren 4 Bits abgelegt und erst beim Auslesen in die korrekte Position geschoben, siehe Funktion packTimeVal(). Beim Auslesen der Daten ist dann eine Dekodierung notwendig. Dies erfolgt außerhalb der zeitkritischen Erfassungs-Schleife in der Funktion unpackTimeVal().

Ende der Erfassung und Ausgabe der Daten

Für den Abbruch der Erfassung gibt es zwei Kriterien:

- Der Puffer für die Speicherung der Werte ist voll
- Die maximal darstellbare Zeit wurde überschritten (entspricht einem Timeout, d.h. kein Ereignis für eine bestimmte Zeit). Dies passiert, wenn 65 ms lang keine Flanke erkannt wird.

Am Anfang der Erfassung wird endlos auf eine low->high Flanke gewartet, da man sonst nach dem Start der Erfassung innerhalb von 65 ms das zu analysierende Signal anlegen müsste.

Die erfassten Werte werden in drei Formaten über die serielle Schnittstelle ausgegeben.

- In einem kompakten Format, bei welchem die einzelnen High- und Low-Zeiten durch + und - markiert sind, die Werte sind in ns
- Im CSV-Format welches direkt in OpenOffice oder Excel importiert werden kann um ein Diagramm zu erstellen. Für jede Flanke werden zwei Zeilen ausgegeben um eine senkrechte Linie im Diagramm darzustellen:
- Als Zusammenfassung werden die Periodendauer und die Modulationsfrequenz ausgegeben (hierbei werden nur Pulse/Perioden berücksichtigt, die maximal 10% über der kürzesten Periodendauer liegen.

□

Downloads & Links

Der Komplette Source Code für die AVR's ist unter folgendem Link zu finden:

[1] http://www.mikrocontroller.net/articles/High-Speed_capture_mit_ATmega_Timer

[2] http://www.mikrocontroller.net/articles/Datei:HighSpeedCaptureAtmegaTimer_Source.zip

Retrocomputing auf FPGA

FPGAküchle - mikrocontroller.net [1]

Der folgende Artikel führt an Hand eines Nachbaus eines 8-bit Computers in den Bereich des Retrocomputing mit FPGAs ein. Es wird gezeigt wie durch Nutzung von FPGA-Baugruppen, Internet-Ressourcen (Handbücher, ROM-Images, open-VHDL-Sources) und üblicher Peripherie (SVGA-Monitor, PS2-Tastatur) Jahrzehntealte Computertechnik nachgebaut werden kann. Da jeder Artikel über einen FPGA-Thema schwer zur Mutation in ein hundertseitiges Buch neigt, gibt dieser Artikel Praxis-Erfahrungen über ein DoItYourself Projekt an Geübte wieder und verzichtet auf die Erörterung von FPGA-Grundlagen. In diesem Projekt wird der Z1013, ein 8-bit Computer-Bausatz von 1985 aus Ostdeutschland auf einem Xilinx Spartan-3 Starterkit nachgebaut.

Computer Bausatz Z1013

Warum Z1013? Zum Kennenlernen der Architektur eines Computersystems ist der Z1013 wegen seiner detailierten Dokumentation ideal. Spezialchips zur Sound und Grafikausgabe gibt es nicht, man kann sich also auf den kleinsten gemeinsamen Nenner aller Computer/Controllertechnik beschränken. Das Handbuch beschreibt die Interna wie Speicheraufteilung, Interruptsystem, IO-Portadressen im verständlichen Deutsch, Schaltpläne sind Bestandteil des Anhangs. Durch den geringen Speicherbedarf kann das System ohne externen Speicher auf einem LowCost-FPGA realisiert werden. Für Systeme mit einer Z80 CPU ist mit CP/M ein kommerzielles Betriebssystem und damit eine umfangreiche Nutzsoftware verfügbar. Dazu muss das Z1013 System um Massenspeicher (Emulation Diskettenlaufwerke mit 800k Speicherplatz) und Arbeitsspeicher (mindestens 64k) erweitert werden, was mit einem geeigneten FPGA-Evalboard leicht möglich ist. Homecomputer mit einer anderen CPU (6502 wie C64, Atari 800, Apple II) bieten mangels eines Betriebssystems diese Breite an Software nicht oder sind restriktiver hinsichtlich Firmware-Nutzung in Nachbauten.

Eine Z1013-FPGA System kann auch als Grundlage für den Nachbau einer ganzen Reihe von Z-80 basierenden Ost-Computer dienen. Beispielsweise des PC1715 (Quasi-Industriestandard Ostdeutschland und Russland) oder der KC-Reihe (KC-85-2/3/4) die an Hochschulen und Universitäten weit verbreitet war.

Und es gibt einen freien PC-Emulator (J1013) der bei der Entwicklung hilfreich sein kann.

Kein Nachbau nach Schaltplan. Ein Nachbau strichgenau nach Schaltplan ist nur auf dem ersten Blick leichter umsetzbar als ein Neu-Entwurf nach den ursprünglichen Vorgaben. Das Original muss mit alter, wenn nicht gar nahezu ausgestorbener Peripherie (Kassettenrekorder als Datenspeichers, Antenneneingang an einem TV-Gerät) funktionieren. Zum Entwicklungszeitpunkt waren statische RAMs größer 2k-Byte im Entwicklungsland (ehemalige DDR) nicht verfügbar und so wurden - aus strengem Preisdiktat - asynchrone dynamischen Speicher mit einem einzelnen bidirektionalen Datenport und RAS/CAS Addressmultiplex verbaut. Aber auch beim Entwurf des Apple II entschied sich

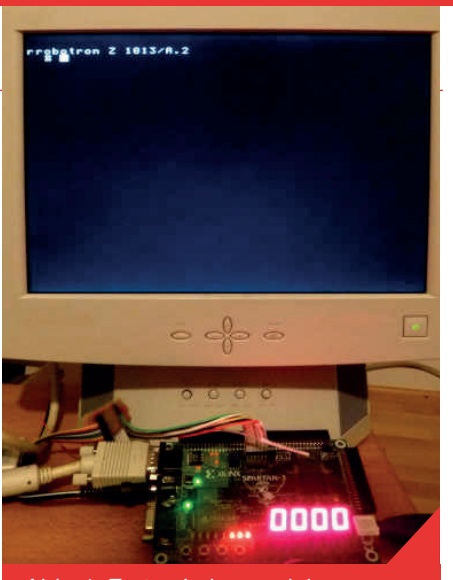


Abb. 1: Erstes Lebenszeichen

Woz für DRAM. Eine DRAM-Zelle benötigt einen Transistor, SRAM dagegen um die 6. DRAM bietet so etwa 4 mal mehr Speicher für den gleichen Preis und ihm genügt ein kleineres Netzteil.

Das erfordert allerdings zusätzliche Digitallogik für das Umschalten der Busse (Refreshzyklen, Bildspeicher etc.). Bei Verwendung aktueller Peripherie (SVGA-Monitor) und Bauteilen (Embedded Dual-port SRAM im FPGA) entfallen Teile der Schaltung komplett (HF-Modu-

lator). Dual-Port RAMS benötigen keine Umschaltung zwischen CPU und Bildgenerator und können mit weniger Logikgattern aufgebaut werden.

Firmware: Die Grundsoftware für den Z1013, in PCs BIOS oder Firmware genannt, heißt hier Monitorprogramm und passt auf 2 KByte. Andere nachzuladende Software wie BASIC-Interpreter setzen auf diese Firmware auf, indem sie auf Routinen bspw. zum Auslesen der Tastatur aufsetzen, resp. diese auf vereinbarten Adressen (Sprungverteiler) aufrufen.

Peripherie: Der Hersteller der Z80 CPU entwickelte für den Anschluss weiterer Geräte wie Tastaturen oder Tapes einen Spezialschaltkreis für parallelen In- und Output (PIO). Dieser wird wie die von Mikrocontrollern bekannten GPIOs eingesetzt und generiert und bedient Interruptanforderungen an die CPU. Die PIO stellt zwei 8bit breite Ports (PORT A und PORT B) zur Verfügung, nimmt IRQ-Anforderungen andere Peripherie-IC's in der Daisy Chain entgegen und leitet sie an die CPU weiter. Die Ports müssen zu Beginn auf eine Betriebsart (Byteweise IN, Byteweise OUT, Einzelbitbetrieb,...) konfiguriert werden. Im Z1013 wird die Matrixtastatur und das Diskettenlaufwerk über die PIO gesteuert. Daher wird die Betriebsart Einzelbit benutzt und nicht wie erwartet der Intterupt-betrieb. Dadurch genügt eine teilweiser Nachbau dieses IC.

Für den Z1013 ist eine Matrix-Tastatur vorgesehen, die von der Firmware aktiv abgefragt wird. Die Firmware wählt immer eine Spalte der Tastatur an und liest eine n bit-Zahl wobei jedes bit für eine andere Taste der Tastatur steht. Einige Modelle des Bausatzes unterstützen zwei verschiedene Matrix-Tastaturen, zusätzlich zu der Standard 8x4 Tastatur eine „Komforttastatur" mit einer 8x8 Matrix. So wurden die Tasten nicht übermäßig mehrfach belegt.

Massenspeicher: Der Z1013 lädt und speichert Anwenderprogramme/Daten vom Arbeitsspeicher zu einem Magnetband. Dazu werden zwei Pins der PIO benutzt. In einem FPGA-Nachbau kann man stattdessen von einem externen Speicher (Flash) direkt den RAM schreiben/speichern. Die Ansteuerung kann weiterhin über das Monitorprogramm geschehen, nur werden dabei nicht die Load/Save Routinen benutzt. Der Flashcontroller kann als IO-Gerät an den Adressen oberhalb 0x11 eingebunden werden, die Steuerkommandos (Start/Zieladressen/Kommandos) werden mit dem Out Befehl abgesetzt. Für den Datentransfer zum Arbeitsspeicher gibt es unter anderem folgende Datenpfade:

- Dualport
- Memorycontroller
- Busarbitrierung (DMA) des Z80

Die dritte Variante eröffnet die meisten Optionen, das sie unabhängig von der Speicherarchitektur ist.

- Z80 - DMA

Hochfahren und Programm Start. Programme werden nach dem Starten des Monitorprogramms mit dem L(oad) Kommando von Band geladen und mit J(ump) gestartet. Davor initialisiert sich die Hardware:

- Der Datenbus ist auf „00" gezogen, die CPU zählt den Adressbus hoch
- bei Erreichen des (P)ROMS Adressraums wird der Datenbus freigegeben und das Monitorprogramm startet
- das Monitorprogramm initialisiert die PIO zur Benutzung der Tastatur und Bandgerät
- der Anwender lädt vom Band das Programm (hier TinyBASIC) mit dem Kommando L-Kommando (L startadresse Endadresse) ,hier: „L 100 B0FF"
- das Programm wird mit einem Sprung zur Start Adresse gestartet, hier „J 100"

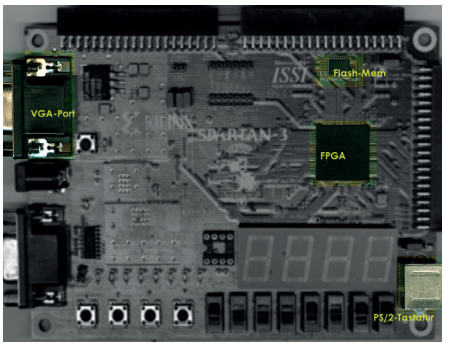


Abb. 2: Xilinx Spartan-3 Starterkit

Computer Bausatz Z1013

FPGA-Boards gibt es zwischen rund 100 € und mehreren Tausend Euro. Fast jedes eignet sich zum Nachbau eines 8-bit Homecomputers. Die folgende Beschreibung der Hauptkomponenten soll helfen ein Board hinsichtlich seiner Eignung für Retrocomputing einzuschätzen. Das Board sollte alle nötige Hardware mitbringen.

VGA-Port: An diesem wird der Monitor angeschlossen, es gibt nur eine Bauform. Das Nachrüsten eines VGA Ports an Erweiterungsport ist möglich. Mit einem FPGA-Pin pro Grundfarbsignal wie im Spartan3-SK sind 8 Farben möglich Weiss auf Schwarz, Grün auf Schwarz, und Gelb auf Blau sind gut lesbare Kombination für Vorder- und Hintergrundfarbe. Bei eingen Evalboards sind 8 pins über Spannungsteiler mit den RGB-Leitungen, das gestattet 256 Farben. Für den Z1013 mit seiner Zeichensatzgrafik ist der VGA-Anschluss mit 3 bit Farbtiefe völlig ausreichend.

Tastaturanschluß: Ein reinrassiger USB-Anschluß ist komplexer als ein 8 Bit Computer selbst, daher sollte eine PS/2 Tastatur benutzt werden. Einige Tastaturen mit USB-Anschluß arbeiten mit einem PS/2 Adapter im sogenannten COMBO. Beim Xilinx Spartan3AN Starterkit liegen die Anschluße allerdings so beengt, das die benachtbarten RS232 Anschlüsse seitlich weggedrückt werden. Das Evalboard sollte also mit einem PS2-Anschluß ausgerüstet sein. Natürlich lässt sich eine PS2 Tastatur auch an zwei Pins und 3.3V fädeln (obwohl PS2 für 5V spezifiziert ist).

Speicher: Im Original ist der Arbeitsspeicher aus dynamischen asynchronen single Port RAM aufgebaut, der Bildspeicher aus asynchronen statischen RAM mit bidirektionalem Datenport. Der Adressbus des Bildspeichers wird über einen Multiplexer zwischen CPU und Adresszähler geteilt.

Im FPGA dagegen stehen uns mit dem BRAM synchroner statischer dual port Ram zur Verfügung, der bereits nach PowerUp initialisiert ist. Dies vereinfacht insbesonders die Beschaltung des Bildspeichers. Im Original wird der einzige Port zwischen CPU und Displaylogik mittels der Bustreiber im Adress- und Datenbus umgeschaltet. Im FPGA dagegen wird die CPU an einem Port, der nur zum Beschreiben genutzt wird angeschlossen, die Videologik an den zweiten Read-Only Port.

Zuweilen genügen die Speicherblöcke im FPGA, die 12 BRAMS Spartan3-200 sind genug für 16kB Arbeitsspeicher, 4kB ROM mit Urlader, 2kByte Zeichensatz und 1kByte Bildspeicher. Dem Betriebssystem CP/M mit typischerweise 80x24 Zeichen genügt das völlig. Dagegen benötigt Pixelgrafik schnell ein Mehrfaches, schon bei heute als grobpixelig geltenden 256 Farben.

Während Arbeitsspeicher einfach mit FPGA-externen Standard-RAM ersetzbar ist, vereinfacht interner Bildspeicher

als Dual-port RAM das Systemdesign deutlich. Allerdings reicht interner RAM insbesonders der kleineren XILINX-FPGAs kaum für mehrfarbige Pixelgrafiken. Altera - FPGAs gelten hinsichtlich RAM als besser ausgestattet. Alternativ zum Embedded RAM kann ein zweiter externer SRAM eingesetzt werden. Bilddaten könnten zeilenweise zwischengespeichert und während der Blankphasen (VBlank, HBlank, Schwarzränder geschrieben werden.

Dem Digilent BASYS2 Board fehlt leider jeder Speicherchip und der on-board Spartan-3 - 100 mit 2 BRAM's (4 kByte) ist zu klein um diesen Mangel auszugleichen.

Massenspeicher: Programme für 8bit Computer sind meist kleiner als der zur Verfügung stehende RAM, also 64 kB. Ein Eval-Board für Retrocomputing sollte also mindestens 64 kByte an nichtflüchtigen Speicher mitbringen, für Diskettenorientierte Systeme (C64 ca. 180 kByte; CP/M mit 400 - 800 kByte pro Diskette) deutlich mehr. Der ROM für das FPGA-Bitfile kann bis zu 128 kByte an Nutzerdaten aufnehmen, so das Platform-Flash XCF02S auf dem Spartan3 Starterkit und kann vom PC aus mit dem FPGA Programmiertool impact) beschrieben werden. Anderer Boards haben Flash Chips on Board (bspw S-3A SK mit ???). Speicher mit seriellen Interface

Computer	Mode	Bildspeicher	FPGA	Embedded RAM
Z1013	32x32 Zeichen	2+1 kByte	Sparten3A-200	36 kByte
C64	320x200 16 Farben	2x16 kByte	16	64 kByte
VGA	320x200 256 Farben	64 kByte	EP3C40	126 kByte
SVGA	800x600 HiColor	960 kByte	Actel-Igloo-600	~25 kByte

Tabelle 1: Übersicht über Grafik und Bildspeicher

genügt den geringen Geschwindigkeitsanforderungen.

Der Platform-ROM hat nur eine begrenzte Anzahl von Schreibzyklen, die für den xcf02s auf dem Startekit von Xilinx mit mindestens zwanzig Tausend angegeben wird. Das ist für die gelegentliche Nutzung ewig, bei intensiver Nutzung durch eine Person 10 Schreiben pro Arbeitstag noch normal endlich (8 Jahre), aber bei Nutzung in der Schulung oder als CP/M - Arbeitsrechner mit häufigen Zwischenspeichern Zum Beispiel von Basicprogrammen/Spielständen/Texten bedenklich. Zumal ein Austausch eine BGA-Reworkstation erfordert und nur begrenzt oft möglich ist. Eine SD-Karten Lösung wie in der ct beschrieben oder in den ZTEX Modulen realisiert ist der bessere Disk-Ersatz.

Zusammenfassung Boardauswahl: Für den Nachbau eines 8 Systems genügen fast alle FPGA-Boards, mit Einschränkungen auch die vor 10 Jahren gefertigten. Nachrüstungen sind über die Expansionports meist einfach möglich, aber oft nicht nötig.

Für den lötfreien Einstieg:

- VGA-Port: hat mindestens 3 bit Farbcodierung (Rot, Grün, Blau) für 8 Farben, besser sind 8 bit
- PS2 Schnittstelle (3V3 sollten OK sein, obwohl PS/2 für 5V spezifiziert)
- 512kbit Flash/PROM
- RAM: 24 kB intern für die allerkleinsten 8-Biter, besser 64-128kB SRAM
- kleiner LowCost-FPGA (mindestens 200k Spartan-3)
- RS232 für eine Terminalverbindung zum PC
- 4 LED's, 3 Taster, 4x7 Segment zum schöner debuggen

Implementierung

Toolchain/Filestruktur: Begonnen wurde das Projekt mit einer einzigen Zielhardware - dem Xilinx Spartan3 starterkit - im Kopf. Die Dateien und der Flow sind noch für dieses Target ausgelegt, wünschenswert ist eine Trennung zwischen den Files für die unterschiedlichen Targets, wie Spartan 3-AN Evalboards, ZTEX-Spartan-6 boards, Altera DE2, Actel-FLASH FPGA's.

Die Sourcefiles wurden bereits aufgeteilt in

- Board-spezifische Dateien (Pin-definitions, wrapper, testbench) unter /Starterkit
- vom Autor für den Nachbau erstellte files /src/vhdl
- kleine Hilfsmodule (LED-blinker, 7-Seg Muxer) unter /kleinkram
- 3rd party sources aus anderen Open-Source o.ä. Projekten haben ebenfalls eigene Verzeichnisse

Dateien die sich von board zu board unterschieden wie Makefile, Pinassignments (.ucf für Xilinx) und die top-entity finden sich im Verzeichnis /board_specific. Die Dateien für das Sparta3-Starterkit liegen bspw. unter board_specific\Xilinx\Spartan-3_StarterKit. Architekturspezifische Files wird es für das Taktsystem und den Zugriff auf den Konfig-ROM (User-Data im Platform-Flash) geben. Diese Files werden dann im Flow durch die für Altera etc ersetzt. Kleinere Unterschiede können über das generic SYSTEM ausgewählt werden und im HDL-Code ohne Dateiaustausch umgesetzt werden.

Die Steuerung über ein Makefile trägt ebenfalls zur Portierbarkeit bei. Für die Umsetzung auf Windows-pc ist das make-tool aus den GNU-tools for windows zu installieren. Mit make werden alle tools aufgerufen um aus den sourcen ein binäry für die direkte Programmierung des FPGA's zu erzeugen und ein PROM-Image für das Starterkit zu erzeugen.

Synchrone BRAM ersetzen asynchrone SRAM/ROM/DRAM: Im Original sind CPU und die PIO (Parallel In/Out) mit 1 - 4 MHz getaktet, die anderen Bausteine (RAM, ROM's) arbeiten asynchron. Dagegen sind die Speicherblöcke auf dem FPGA (BRAM) synchron, müssen also mit der CPU an einem Taktnetz liegen. Um Latenzen gegenüber den originalen Taktschema zu vermeiden wird die CPU mit der entgegengesetzten (Steigend - rising) Flanke bezüglich der Speicherbausteine/PIO (Fallend - falling) betrieben.

Takt: (Source: dcm_sys.vhd) Für die Erzeugung des SVGA-Timing ist ein Takt von 40 MHz nötig. Auf dem Evalboard befindet sich ein Taktgenerator mit 50 MHz. Das Originalsystem wurde je nach Qualität der verbauten Schaltkreise (sogenannte Bastlertypen) mit 1, 2 oder 4 MHz betrieben. Jedes Board konnte (auf eigenes Risiko) von 1 auf 2 MHz umgelötet werden. Auch wenn das FPGA-Design deutlich schneller getaktet werden könnte, wird ein Takt nahe am Original erzeugt. Mit den vier Digitalen Clock Manager (DCM) im Spartan 3 kann der Eingangstakt geradzahlig bis 16 geteilt werden und/oder durch Digitale Frequenzsynthese (DFS) auf eine Frequenz mit einem nicht-ganz-

zahlige Faktor umgewandelt werden.

Für eine Umsetzung auf andere Boards muss das Taktsystem möglichst einfach sein. Deshalb wird nur eine DCM verwendet, die DFS erzeugt aus den 50 MHz den 40 MHz für das SVGA-Module und mit dem maximal möglichen Teiler von 16 wird ein Takt von 3.125 MHz für den Computer erzeugt . Der VHDL-Code (dcm_sys.vhd) wurde mit dem „architecture wizard“ (Xilinx\14.3\ISE_DS\ISE\bin\nt64\arwz.exe) erstellt.

Alternativ hätte man statt mehreren Takten auch einen Takt und ClockEnable verwenden können. Dann hätte man aber auch den CPU-Core umschreiben und neu testen müssen. Da FPGA's und der Designflow mit mehreren Takten umgehen können, wäre dies kein wirklicher Gewinn an Kompatibilität oder Einfachheit. Auch werden die Taktübergänge schon in der System-Architektur abgefangen, Dual-Port Video-RAM und Handshake Signale zur Peripherie haben sich seit Anbeginn der Computertechnik bewährt.

Adressdekoder: Im Adressdecoder werden die select-Leitungen für die Speicherbaugruppen (ROM mit Monitorprogramm, Bildwiederholspeicher, Arbeispeicher) und IO-Geräte (Parallel-IO Port PIO, Tastatur-Zeilen-Ansteuerung) erzeugt. Im Original ist das mit Binär zu Dezimal-Dekodern, AND-Gattern und Diodenlogik realisiert. Im Nachbau genügen VHDL-select Anweisungen. Die Unterscheidung zwischen IO-Ports und Speicher-bereichen geschieht anhand der CPU-Signale IREQ und MREQ. Die Speicheraufteilung und IO-Adressen sind im Anhang des Z1013 Handbuchs beschrieben. Ein REFRESH ist bei SRAM-Blöcken nicht nötig, daher werden die Refresh-Zyklen anhand des Refresh-Signals ausgeblendet.

Arbeitsspeicher: (Source File RAM.vhd). Der Arbeitsspeicher wird als 16kx8bit array beschrieben und ist mit 8 BRAM's des FPGA realisiert. Die unscheinbare Definition von 3 Bytes an den Adressen x0000 - x0002 ersetzt die PullDowns R44-R51, das FF aus A25 und A26. Im Original wird nach dem Reset der Datenbus auf x00 gehalten, so dass die CPU NOP (No-OPeration)-Instruktionen „ausführt“, also nur die Adresse auf dem Addressbus hochzählt. Erst beim Erreichen der PROM Startadresse wird der Datenbus freigegeben und das Betriebssystem wird gestartet. Im Nachbau ist an der ersten Adresse die die CPU einliest ein Sprungbefehl „gebrannt“, - der Nachbau beginnt daher sofort mit der Firmware. Bei Verwendung von externen RAM ist dies nicht möglich und der datenbus muß nach dem Reset vom Speicher getrennt werden. Im Original trennt der Bustreiber A1 die CPU in zwei weiteren Fällen vom Arbeitsspeicher:

- während eine REFRESH-Zyklus und
- einer Busfreigabe für einen DMA (Direct Memory Access) über die CPU-Signale BUSRQ und BUSAK

Letzterer Fall wird benötigt wenn der Speicher von einer anderen Quelle als der CPU genutzt wird. Beispielsweise beim Lesen schreiben durch einen HOST-PC über eine RS232 Schnittstelle o.ä..

Textausgabe: Der Bildspeicher (video_ram.vhd) ist als Dual-Port-Speicher realisiert. Die CPU beschreibt und liest den 1 kB großen Speicher über den einen Port, der SVGA-Controller liest ihn über den zweiten Port aus. Zu Testzwecken kann die CPU vom Bildspeicher getrennt werden (Generic: G_READONLY). Der VGA-Controller stellt dann den Bildspeicher so dar wie er beim Laden das FPGA's initialisiert wurde. In dem Package video_ram_pkg.vhd sind zwei Konstanten für die Initialisierung definiert. Mit C_VRAM_ARRAY_INIT stellt die wichtigsten Zeichen auf den oberen Zeilen dar:

Somit kann die Monitorausgabe überprüft werden ohne das ein Programm von der CPU abgearbeitet wird. Dagegen füllt die andere Konstante C_VRAM_ARRAY_SPACES_INIT den Schirm mit Leerzeichen. Der Datenausgang des Bildspeichers ist mit den oberen 8 Adressbits des Zeichengenerators (Eingang: addr_char_i) verbunden und wählt so das zu darstellende Zeichen aus. Die unteren 3 Addressbits (addr_line_i) bestimmen die aktuell auszugebende Zeile des Zeichens. Der Zeichengenerator ist als 2KByte großer ROM realisiert (Datei: charrom.vhd). Ist der PROM nicht selektiert wird eine Zeile eines Leerzeichens (alle bits 0) ausgegeben.

Der Zeichensatz selbst ist im Package bm100_pkg.vhd [11] definiert. BM100 steht für „Bitmuster 100“ dem Standard-Zeichensatz des Z1013. Die Originale finden sich auf [12] als Bitmuster-Dateien (siehe Erläuterungen oben). Alternativ können Zeichensätze des C64 Homecomputers benutzt werden, bspw.: [13]. Beide Computer verwenden Zeichen aus 8 mal 8 Pixeln. Auf dem SVN Server wurde ein Package mit leeren Zeichen abgelegt. Wie es mit mittels der Binärdatei modifiziert werden kann, wurde oben beschrieben.

Für die Ausgabe über die VGA-Schnittstelle nach VESA-Standard wurde der Beispielcontroller von Digilent ohne Änderungen übernommen. Im Modul

Abb. 3: Testbild (video_ram_pkg.vhd)

video.vhd wird der Bildspeicher ausgelesen und als RGB Signal ausgegeben. Mit 6 Schaltern auf dem Starterkit kann Vordergrund- und Hintergrundfarbe an persönliche Sehgewohnheiten angepasst werden.

Tastatur: Für den Anschluss der PC-Tastatur an die PIO sind zwei Module zuständig. Das erste liest über das serielle 2-Draht Interface den Code der Taste, das zweite emuliert die originale 8x4 Matrix und wie diese den Tastendruck signalisieren würde. Für das PS/2 Interface wurde zuerst ein Beispiel für das S3-Starterkit verwendet, eine nicht so gute Wahl. Das Demo setzt den Tastendruck direkt auf die 7-Segment Anzeige um und „unterschlägt“ dabei die Steuercodes die anzeigen, ob die Taste gedrückt oder losgelassen wurde. Das Modul wurde daher komplett umgeschrieben, so dass der Tastencode bereits beim Drücken der Taste mit 8 bit signalisiert wird und nicht erst bei deren loslassen. Dies ist für die Erkennung einer Eingabe mit gleichzeitig gedrückter Shift-Taste unabdingbar. Tückisch ist eine Komfortfunktion der MF2-Tastatur - die automatische Tastenwiederholung (typematic). Wird eine Taste längere Zeit (typematic delay) gedrückt gehalten, wird deren scancode automatisch wiederholt (typematic rate (per second)). Daher wird die Tastatur in ein Schieberegister eingeschoben bis der Code xF0 gelesen wird. Dieser geht dem scancode voran sobald die Taste losgelassen wird. Das Zeichen wird also erst nach Loslassen der Taste angezeigt. Das neue Interface dagegen liest 8 bits nach dem Startbit und analysiert dann ob es um ein Steuerzeichen handelt (weiter Auswertung: xF0= -> Taste losgelassen, xE0 extended scancode) oder nicht (Taste gedrückt).

Der zweite Controller (Keyboardemulator) ist an den Spaltentreiber des Z1013 und an 4 auf Eingang konfigurierten Leitungen der PIO angeschlossen. Diese 4 Leitungen sind an den einen Datenausgang eines kleinen 8x4bit Speicherblocks angeschlossen. Dieser Block emuliert das originale 8x4 Speicherfeld. Eine ‚0‘ zeigt an das die entsprechende Taste gedrückt

ist. Der Tastaturemulator überprüft ob ein neuer Scancode anliegt. Liegt der Scancode der (PS/2)-Shift-Taste an wird der Status entsprechend geändert und bei der Auswertung des Scancodes herangezogen. Dazu wird intern den 7 bit unteren Bits des Scancodes eine ‚1' vorangestellt. Aus dem Scancode wird ermittelt:

- ob eine der vier Z1013 Shifttasten zu emulieren ist
- die Spalte (1 von 8, Nummer 8 ist außen rechts) der gedruckten Taste in der originalen 4x8 Tastatur
- die One Hot Codierung der Taste auf einer der vier Zeilen („0001" für die unterste (Shifttasten))

Die Zuordnung von Scancode zu Shift und Tastenmatrix ist in einem Record hinterlegt.

Die Spalten (col) werden in dem Record von 1 bis 8 nummeriert, die Zeilennummer reichen von 1 bis 4. Die Shifttasten sind mit NON,S1,S2 und S3 codiert. Im FPGA wird diese codierung in einen ROM mit 7bit breiten Datenworten umgesetzt. Von deisen 7 bit stehen 2 für den SHIFTcode, 3 für die Spaltennummer und 2 für die Zeile. Die Konvertierung vom record

```
CONSTANT C_ALL_KEYS : T_ALL_KEYS := (
--Index =>  Scancode    , Shift, Col,  Row
  K_0     => (16#45#,     S1,     1,    2),     --'0'
  K_0_S   => (16#C5#,     S1,     6,    3),     --'='
  K_1     => (16#16#,     S1,     2,    2),     --'1'
  K_A     => (16#1C#,     NONE,   2,    1),     --'A'
  K_A_S   => (16#9C#,     S3,     2,    1),     --'a'
```

Listing 1: Zuordnung von Scancodes zu Shift und Tastenmatrix

in die 7bit Zahl realisiert die Funktion c8x4slv. Die binär codierte Zeilennummer wird in eine 1 aus 4 Codierung umgesetzt, wobei ‚0' für eine gedrückte Taste der 8x4 Matrix steht. Die Spaltennummer wird als Adresse, das Bitmuster der Spalte(n) als Datum für den Dualport 8x4 bit Speicher genutzt. Dieses kleine Speicherfeld „emuliert" das Tastenfeld.

Eine Stamachine setzt das Drücken bzw. Loslassen einer Taste in zwei Phasen um. Zuerst wird in der Tastermatrix die S1|S2 oder S3 Taste nachgestellt, dann die eigentliche Taste. Für sie S1-S3 Tasten wird die Spaltenadresse ausdekodiert, die Zeilennummer ist immer „1110" (unterste Zeile). Da in den vier Spalten links zwei Tasten gedrückt sein können (bspw S1 für x) wird in der zweiten Phase die zeilennummer über den zweiten Port zurückgelesen und die Zeilennummer der eigentlichen Taste hinzugefügt.

Beim Loslassen einer Taste wird für die S1|S2|S3 und für die eigentliche Taste immer „1111" geschrieben also die gesamte Spalte als „nicht gedrückt" signalisiert.

Zwischen den vier Phasen (Shift/Taste „loslassen", eigentliche Shift/Taste „drucken) die für eine Taste der PS/2 abgearbeitet werden Pausen von zirka 10 Milisekunden eingelegt, damit die Firmware auch das „Loslassen" 4x8 Tasten erkennt. Ohne diese Pausen werden von der originalen Firmware nur ein Drittel der Tasten und diese auch noch unzuverlässig erkannt.

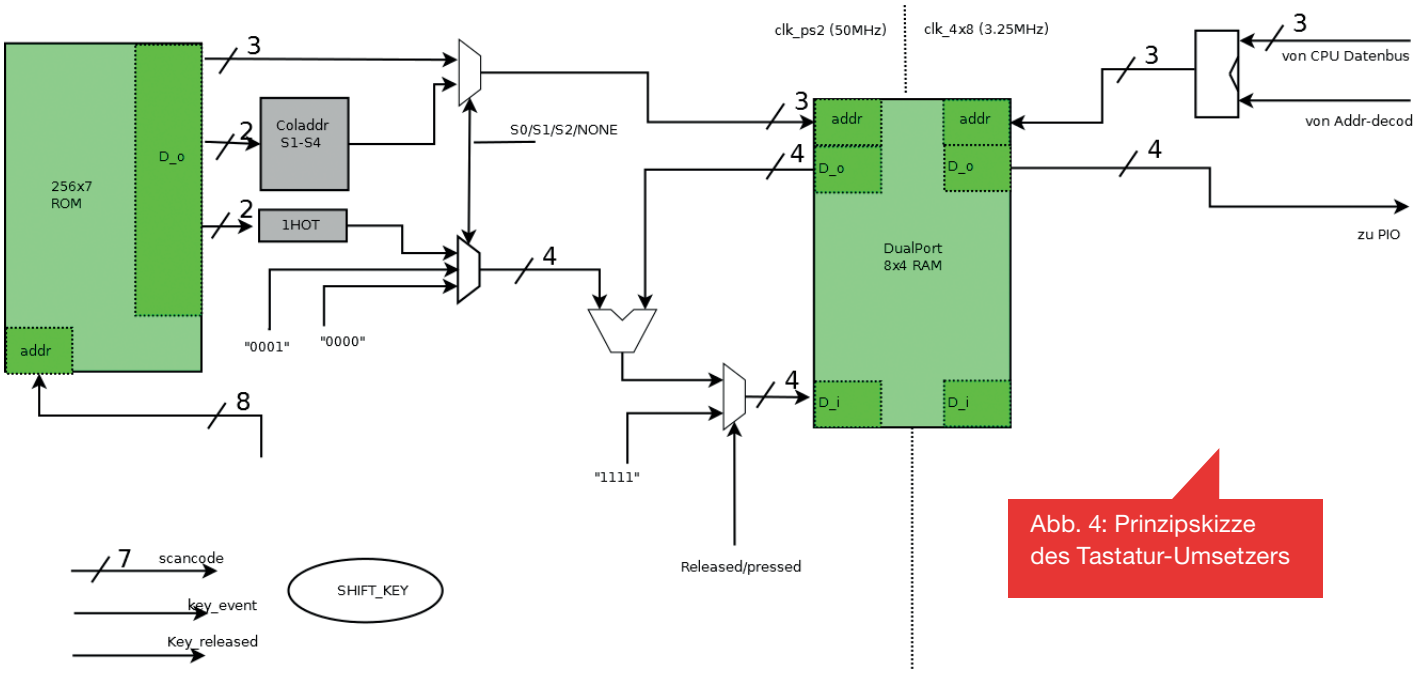


Abb. 4: Prinzipskizze des Tastatur-Umsetzers

„Vor"-laden und Starten eines Programmes

Der Inhalt des Arbeitsspeicher kann bereits im VHDL-Code initialisert werden und so ein programm ohne den Umweg Ladebefehl des Monitorprogrammes geladen werden. Programme für den Z1013 liegen meist als *.z80 Dateien vor. Die ersten 32 byte enthalten als Header die Start-und die End adresse des Programmes, die folgenden bytessind ab der Startadresse in den RAM zu speichern. Der (entfernte) Header für tinyBASIC ist in Listing 2 zu sehen. Die ersten zwei Bytes nennen die Staradresse (0100 in getauschten bytes) bytes drei und vier nennen die endadresse (ebenfalls gedreht 0bFF).

```
--0000000  00  01  ff  0b  00  01
00  00  00  00  00  00  43  d3  d3  d3

--0000020 5a 31 30 31 33 20 54 69
6e 79 2d 42 61 73 69 63
```

Listing 2: 32 byte header

- FPGA kompilieren und laden
- Befehl „J 100“ eintippen -> das Programm starten

Das Monitorprogramm muß vor dem Anwenderprogramm gestartet werden, da es die PIO konfiguriert. Ein Sprungbefehl auf die Adresse 0100 stat auf F000 in den ersten 3 Bytes des RAM's startet daher nicht korrekt.

Quelltexte und Compilation

Vom Autor erstellte Quellen liegen auf dem hiesigen SVN-Server (siehe Downloads [2]). Fürs Gesamtprojekt brauchts auch Quelltexte die auf anderen Open-Source Depots lagern:

Bitte die Quellen von den angegebenen Servern downladen und in die zugehörigen Verzeichnisse kopieren. Die Tastaturansteuerung wurde im Laufe des Projektes komplett neu geschrieben und liegt daher auf dem hiesigen Server.

Neben diesen VHDL-Quelltexten werden auch das ROM-Image für das Monitorprogramm benötigt. Die Originale und Hobbyisten-Varianten finden sich auf [21] als Bitmuster-Dateien (siehe Erläuterungen oben). Alternative Firmwaren gibt es dort: [22] Wie man Binärdateien nach VHDL-konvertiert wurde im Abschnitt Retrocomputing_auf_FPGA#(P)ROM-Images gezeigt.

Das Kompilieren erfolgt per make. Aufgerufen wird das make aus den GNU-Utilities for Windows [40]. Andere maketools wie nmake von microsoft sind frustrierend inkompatibel zu dem gnu-make, daher wird nur dieses unterstützt.

Verzeichnis	Zweck	Woher
t80	CPU (Z80)	http://opencores.org/project,t80
VGARefComp	SVGA Controller 800x600 @ 60Hz	http://goo.gl/GI6Wa6

Tabelle 2: 3rd party sources

Debug-Module FPGA-Systeme

Viele FPGA-Systeme nutzen Softcores (NIOS, picoblaze) zur Ansteuerung und kommunizieren mit einem Steuerrechner über RS232 (Terminalverbindung) oder Ethernet. Der Softcore parst die eingehenden Kommandos (meist in der Form RD|WR Addr data) und beschreibt/liest dann die FPGA-Register. Wie gezeigt kann auch ein komplettes (Home)-Computer system in einem FPGA generiert werden. Im Unterschied zu selbstgestrickter Softcore-Software steht eine Menge von erprobter und schlanker Software zur Verfügung z.B. BASIC. In BASIC wird mit den IN/OUT Befehlen auf Register an der Peripherie zugegriffen, mit PEEK/POKE auf Speicherbereiche. Auch kommt ein solches System ohne einen Steuerrechner aus, solange eine Tastatur und ein VGA-Display oder gar Handy-Beamer mit VGA-Eingang angeschlossen werden kann. Natürlich ist die Rechenleistung eines historischen Computersystem fern von dem eines auf FPGA's optimierten 32bit Softcores. Fürs Parametrisieren von Registern, Auslesen von Speicherbereichen und aufbereitete Ausgabe mit kurzem BASIC-Programm reicht es allemal. Und kommt ohne Megabyte an C-Compiler Umgebung und „Micro“-Linux-Cores aus.

Altrechner transportabel: Für die meisten frühen Computersysteme wurde nie eine Mobile Version entwickelt. Eine FPGA Version ist nicht nur deutlich kleiner und kann mit einer Batterie betrieben werden, da VGA-Monitore oft verfügbar sind kann der überschwere Original Monitor (meist Röhren-TV) daheim bleiben. Eine Version mit TFT Display statt Monitor wird ebenfalls möglich und bleibt wegen der Textausgabe auch bei kleinen Anzeigen lesbar.

□

Downloads & Links

Weitere Links zu nützlichen und ergänzenden Themen und Beiträgen finden Sie auf bei dem Artikel auf mikrocontroller.net [1].

[1] http://www.mikrocontroller.net/articles/Retrocomputing_auf_FPGA

[2] http://www.mikrocontroller.net/svnbrowser/redz0mb1e/

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

elektor GOLD membership

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

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

UMWELTSCHONEND – GÜNSTIG – GREEN

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

elektor GREEN membership

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

Batteriemonitor mit Dual Slope Wandler

H. Stockhoff

Der Dual Slope Wandler wird häufig in Vielfachmessgeräten eingesetzt, da dieser kostengünstig in der Fertigung ist, und eine relativ hohe Auflösung bietet. Ein Beispiel für ein IC nach dem Dual Slope Wandler Prinzip ist der ICL7107 der Firma Intersil. Das Verfahren basiert auf dem Vergleich zwischen einer integrierten Mess- und Referenzspannung. Es arbeitet also mit zwei Rampen. Daher wird dieses Verfahren auch Zweirampenverfahren oder Dual Slope genannt. Ich möchte in diesem Artikel ein Batteriemonitor zur Spannungs- und Strommessung vorstellen. Das Messgerät kann später an einem 12V Akku betrieben werden und zeigt den aktuellen Stromfluss, die Spannung und die entnommene Ladung sowie die Laufzeit an. Mit entsprechender Bauteildimensionierung (Glättungskondensator und Spannungsteiler für die Betriebsspannungsmessung sowie Spannungsregler) kann das Gerät auch an Spannungen bis zu 28V betrieben werden.

Features

- Messung der entnommenen Ladung Q
- Auflösung: 0,001A
- max. I: 6A
- UBetrieb: 8-14V/DC
- Automatischer Nullabgleich
- einfache Bedienung
- Stromaufnahme bei eingeschalteter Beleuchtung: 39mA
- Stromaufnahme bei ausgeschalteter Beleuchtung: 15mA

Allgemeines zum Dual Slope Wandler Prinzip

Der Dual Slope Wandler setzt sich grundlegend aus einem Integrator und einem nachfolgenden Komparator zusammen. Eine Steuerlogik legt die entsprechenden Signale an den Eingang des Integrators. Ein Zähler, der mithilfe des Komparators gestartet und gestoppt wird, misst die am Integrator angelegten Messsignale aus und stellt dann einen Wert in Form eines Zählerstandes zur Verfügung. Der Wert des Timers ist der Mittelwert der angelegten Messspannung über die Messzeit. Daher werden Störungen kompensiert und es können mit diesem Wandler Genauigkeiten von bis zu 0,01% erreicht werden.

Anhand der nebenstehenden Schaltung kann die Funktion des Wandlers noch einmal verdeutlicht werden (Abb. 3).

Abb. 1: Batteriemonitor mit Dual Slope Wandler in Gehäuse

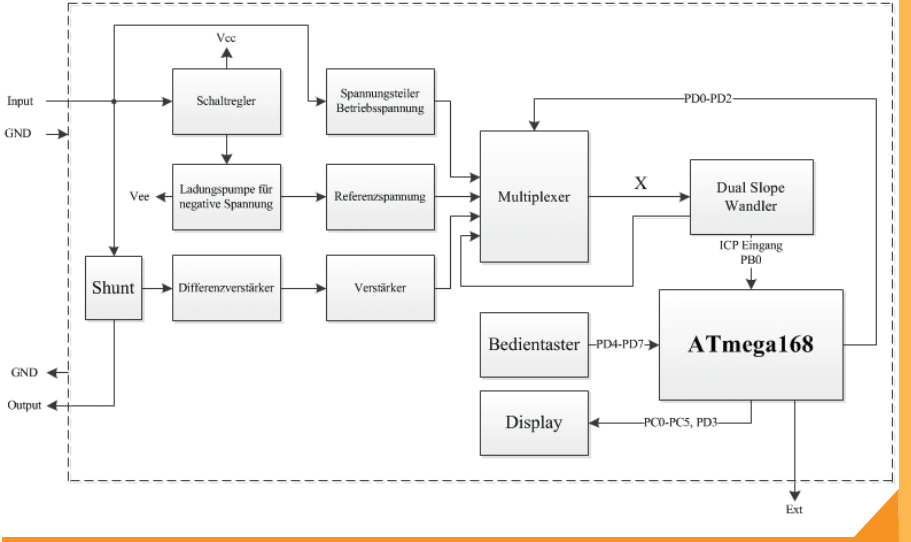


Abb. 2: Blockschaltbild des Batteriemonitors

Funktionsprinzip

Zu Beginn sind die Schalter S1 und S2 geöffnet. Schalter S3 ist geschlossen(siehe Abbildung 3). So wird der Integrationskondensator kurzgeschlossen und entladen. Im nächsten Schritt wird der Schalter S3 geöffnet und der Schalter S1 geschlossen. Es wird nun die zu messende Spannung am Integrator angelegt und aufsummiert. Der Vorteil des Integrators ist, dass der Integrationskondensator mit einem konstanten Strom geladen wird. Dadurch ergibt sich eine lineare Aufladung des Kondensators C. Die Spannung am Ausgang des Integrators besitzt eine umgekehrte Polarität, da die Eingangsspannung an den invertierenden Eingang des Operationsverstärkers gelegt wird. Da die Spannung am invertierenden Eingang des Komparators nun kleiner als 0V ist, gibt er das Tor frei(UND-Gatter). Das Signal wird nun so lange integriert, bis der Zähler einmal übergelaufen ist. Anschließend wird die Referenzspannung (umgekehrtes Vorzeichen zur Messspannung) an den invertierenden Eingang des Integrators angelegt. Dazu wird der Schalter S1 geöffnet und der Schalter S2 geschlossen. Die Spannung am Kondensator C des Integrators wird abintegriert. Sie sinkt linear, da auch hier der Kondensator C mit einem konstanten Strom geladen bzw. in diesem Fall entladen wird. Zu diesem Zeitpunkt gibt der Komparator das Tor noch frei, da die Spannung am invertierenden Eingang über ein Potential kleiner als 0V verfügt. Wenn der Kondensator nun so weit entladen bzw. aufgeladen ist und er das 0V Potential überschreitet, sperrt der Komparator das Tor. Er gibt dann ein negatives Signal am Ausgang aus. Somit ist die Messung beendet. Anschließend wird durch die Steuerschaltung der Schalter S2 wieder geöffnet. Um eine gleiche Anfangsbedienung vor jeder Messung zu bekommen, wird der Kondensator C wieder entladen. Dazu wird der Schalter S3 geschlossen. Der Kondensator C ist kurzgeschlossen.

Im nächsten Schritt kann mithilfe des Wertes des Zählers die gemessene Spannung Ue ermittelt werden. Dazu bediene ich mich der Formel für die Aufladung des Integrators über die Zeit t1:

$$U_I(t_1) = -\frac{1}{\tau}\int_0^{t_1} U_e dt = -\frac{1}{RC}\int_0^{t_1} U_e dt$$

mit τ = R·C

UI : Spannung am Integrator

t1 : Zeit der Integration der Eingangsspannung

Ue : Eingangsspannung des Integrators

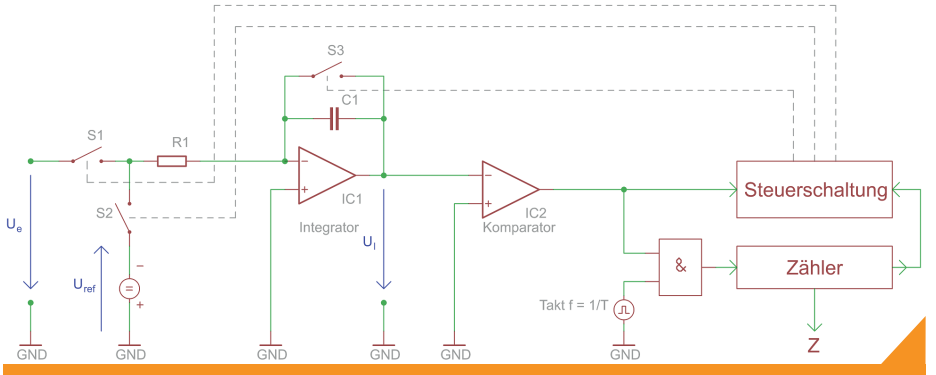


Abb. 3: Dual Slope Wandler

Das Prinzip des Dual Slope Wandlers besteht darin, die Mess- und Referenzspannung (Ue, Uref) zu integrieren. Die Referenzspannung wird mit der Anfangsbedingung UI0 integriert, da sie nach dem Anlegen der Messspannung aufsummiert wird. Da ich eine zeitlich konstante Spannung integriere, gilt der Sonderfall für die Berechnung der Spannung UI des Integrators:

$$U_I = -\frac{U_e}{\tau}\cdot t + U_{I0}$$

mit τ = R·C

Ue : Eingangsspannung des Integrators

UI : Spannung am Integrator

UI0 : Anfangsbedingung des Integrators (Anfangsspannung an UI)

t : Zeit der Integration

Die Zeit t setzt sich zusammen aus dem maximalen Zählerstand Nmax und der Periode T.

$$t = (N_{max}+1)\cdot T$$

Somit ist

$$U_I = -\frac{U_e}{\tau}\cdot(N_{max}+1)\cdot T + U_{I0}$$

mit τ = R·C

Da beide zu integrierenden Spannungen entgegengesetzte Vorzeichen besitzen, ist die Summe beider Spannungen null. Es gilt:

$$0 = -\frac{U_e}{\tau}\cdot(N_{max}+1)\cdot T + \frac{U_{ref}}{\tau}\cdot N\cdot T$$

Erster Summand: Integration der Messspannung

Zweiter Summand: Integration der Referenzspannung

Durch diese Gleichung kann mithilfe des aus dem Zähler erhaltenen Wertes die Eingangsspannung des Integrators berechnet werden. Dazu ist die Gleichung nach Ue aufzulösen:

$$U_e = \frac{N\cdot U_{ref}}{N_{max}+1}$$

Gut zu sehen ist, dass weder die Periode T bzw. die Taktfrequenz f = 1/T , noch die Zeitkonstante τ des RC-Gliedes mit in die Rechnung einfließen. Dies ist ein spezielles Merkmal des Dual Slope Wandlers. Wird ein Dual Slope Wandler aufgebaut, ist auf den Typ des Operationsverstärkers zu achten. OPs, aufgebaut aus bipolaren Transistoren, besitzen meist hohe Bias Ströme. Diese führen dazu, dass die Aufladung des Kondensators beeinflusst wird. So werden sich die Bias Ströme auf den Kondensator durch die lange Aufladung aufsummieren und das Messergebnis verfälschen. Es sollte daher ein Operationsverstärker mit geringen Bias Strömen gewählt werden, damit die Aufladung des Kondensators nicht beeinflusst oder gar verfälscht wird. Dazu eignen sich meistens OPs mit einem internen FET-Aufbau. Sie besitzen oft kleine Bias Ströme von wenigen Pikoampere (Diese können dann vernachlässigt werden). Somit müssen bei FET-Operationsverstärkern lediglich die Offsetspannungen kompensiert werden, um ein korrektes Ergebnis zu erhalten.

Hardware

Nun zur Hardware. Das Messgerät besteht aus einem Dual Slope Wandler, an den über ein Multiplexer (4051) die verschiedenen Messsignale angelegt werden. Gesteuert wird der Multiplexer mit einem ATmega168 der Firma Atmel. Auf dem Display können die aktuelle Spannung und der Strom eines Verbrauchers abgelesen werden.

Die Schaltung des Messgeräts besteht aus drei Teilen: der Spannungsversorgung, dem Analogteil (AD-Wandler) und dem Digitalteil (Mikrocontroller und Display).

Die Spannungsversorgung besteht aus einem Festspannungsregler und einer Ladungspumpe, sowie einer Diode zur Erzeugung der Referenzspannung von -2,5V (Abbildung 4). Als Festspannungsregler ist ein 7805 vorgesehen. Er stellt die Versorgungsspannung für den Mikrocontroller, das Display und den Analogteil zur Verfügung. Um einen höheren Wirkungsgrad zu erzielen, kann er aber auch gegen einen Schaltregler(TSR 1-2450) ersetzt werden, damit die Schaltung auch mit Spannungen größer als 14V versorgt werden kann. Der 7805 würde zwar die Eingangsspannung von 14V auf 5V herunter regeln; dies würde aber zu einer erhöhten Verlustleistung führen, was zur Folge hat, das sich der Festspannungsregler erwärmt und mit einem Kühlkörper ausgestattet werden müsste. Bei 14V Eingangsspannung und einer Stromaufnahme von 0,2A würde die Verlustleistung 1,8W betragen. Diese muss dann über die Kühlfläche in Abwärme umgesetzt werden. Die Ladungspumpe wurde mit einem ICL7660 realisiert. Sie stellt die negative Versorgungsspannung für die Operationsverstärker, die Referenzspannung und den Multiplexer her. Das einzige Problem bei dieser gewählten Ladungspumpe ist seine Belastbarkeit. Die Ausgangsspannung des ICL7660 kann mit nur wenigen Milliampere belastet werden, bevor sie einbricht. Daher wurden die Ladungskondensatoren etwas größer gewählt und die Ein- und Ausgangsspannung gepuffert.

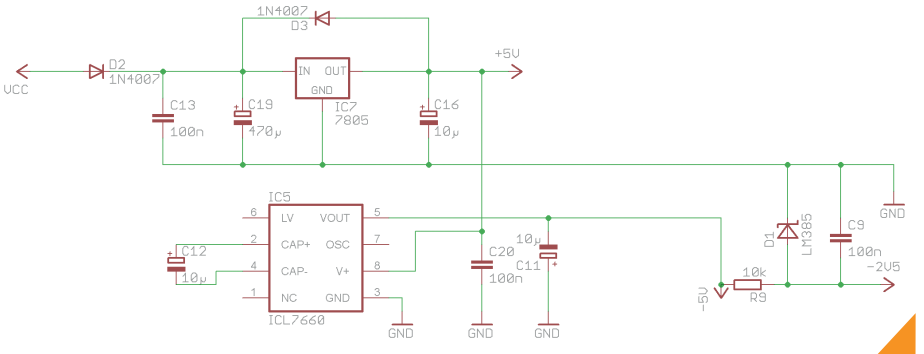


Abb. 4: Spannungsversorgung

Ein zweites wichtiges Kriterium ist eine stabile Eingangsspannung. Diese ist, da die Ladungspumpe am Ausgang des Festspannungsreglers 7805 angeschlossen ist, gewährleistet.

Mithilfe kleiner 100nF Kondensatoren in den Eingangsleitungen beider Regler werden Spannungstransienten (Spannungsspitzen) sicher abgeleitet und so die nachfolgenden Bauteile geschützt.

Die Diode LM385 ist so konzipiert, dass sie eine Spannung von -2,5V erzeugt. Gespeist wird sie durch den ICL7660. Der Widerstand R9 begrenzt den Strom auf 0,5mA. Ein nachfolgender Kondensator fängt noch einmal Spannungsspitzen ab.

Analogteil

Der Analogteil der Schaltung besteht aus einem Differenz- (INA117) und Vierfachoperationsverstärker (TLC274) sowie einem Multiplexer (CD4051). Grundlegend kann der Analogteil auch noch einmal in den Wandlerteil und in den Strommessteil gespalten werden. Beginnen möchte ich mit dem Strommessteil. Dieser besteht aus einem INA117, einem Tiefpassfilter und einem Verstärker. Die Messung des Stroms wird mithilfe eines Shunts realisiert. Genannt wird diese Methode auch indirekte Strommessung, da lediglich der Spannungsfall am Shunt betrachtet wird. Daher wird die Spannung nach dem Shunt von der Spannung vor dem Shunt subtrahiert. Über das ohmsche Gesetz kann anschließend der Strom resultierend aus dem Spannungsfall errechnet werden. In meinem Fall sind der Widerstand (R2 = 0,01Ω) und der gemessene Spannungsfall über dem Widerstand R2 bekannt. Um das ganze jetzt auch messen und digitalisieren zu können, wurde ein Differenzverstärker vom Typ INA117 ausgewählt. Der Differenzverstärker(INA117) besitzt die Eigenschaft, dass er zwei anliegende Signale voneinander subtrahieren kann. Es wird dann der Spannungsfall über dem Shunt(R2) gemessen (Abbildung 5). Die Verstärkung des Ausgangssignals ist 1:1. Ausgewählt wurde der INA117, da er im HIGH-Side Anschluss der Schaltung messen kann. Er ist durch den internen Spannungsteiler im Eingang bis zu Spannungen von 500V abgesichert. Für den Zweck in dieser Anwendung ist es ausreichend, da maximal Spannungen von 14V herrschen. Die Teilungen der Eingangsspannungen belaufen sich auf jeweils 1:20.

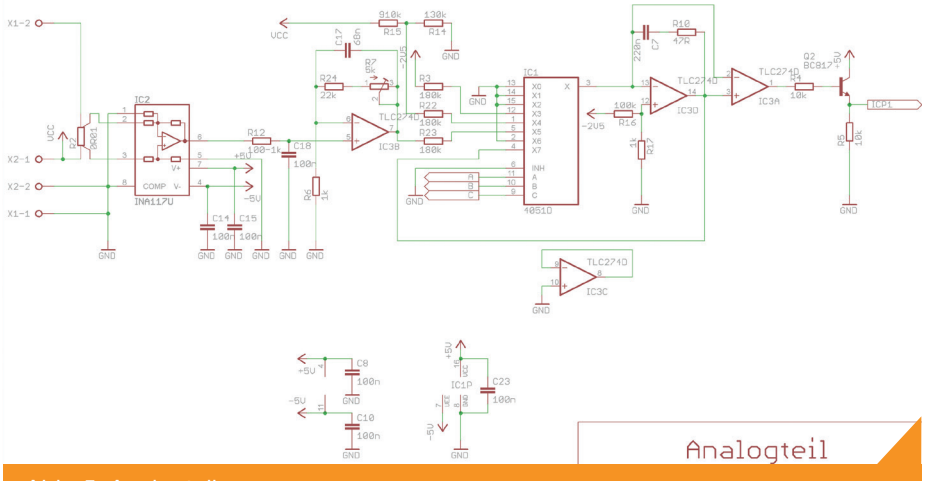


Abb. 5: Analogteil

Da das Signal am Ausgang des INA117 einen hohen Rauschpegel besitzt, wird es im nächsten Schritt durch ein RC-Glied gefiltert und anschließend mithilfe des 4-fach Operationsverstärkers auf das gewünschte Maximum, der Referenzspannung, verstärkt. In der Verstärkerstufe wird das Signal ebenfalls noch einmal

gefiltert, um eine bessere Stabilität zu erlangen. Dies geschieht einerseits durch den Verstärker selber, da er ein geringeres Rauschen als der INA117 am Ausgang ausgibt und andererseits durch das Filter C in der Rückkopplung parallel zum Rückkoppelwiderstand. Hier wurde der Wert 68nF gewählt(siehe dazu auch noch einmal die nachfolgende Rechnung). Dies ergibt dann bei einer Frequenz von 100Hz einen Blindwiderstand XC von 24kΩ.

f = 100HZ und X_C = 24000Ohm

$$X_C = \frac{1}{2 \cdot \pi \cdot f \cdot C}$$

$$\Leftrightarrow 66nF = C$$

Es wird der nächst größere Kondensator gewählt. Somit 68nF.

Der Verstärker dient im genaueren hinsehen dazu, dass Messsignal auf das Maximum der Referenzspannung zu normieren. Die Referenzspannung beträgt, wie oben schon erwähnt, -2,5V. Das bedeutet, dass das Messsignal den Betrag dieser Spannung nicht überschreiten darf.

Um nun den kompletten Messbereich von |-2,5V| auszunutzen, wird das Signal des INA117 mit dem Faktor 25 verstärkt. Anschließend wird es über einen Eingangswiderstand auf den Multiplexer gelegt.

Weitere Eingangssignale des Multiplexers sind neben der zum Strom äquivalenten Spannung auch die Referenzspannung, die aktuelle Spannung vor dem Shunt und die Spannung des Integrationskondensators (Kurzschluss des Kondensators). Die aktuelle Spannung vor dem Shunt dient später dazu, wie weit der Akku bereits entladen ist. Bei einer gewissen Grenze soll der Controller ein Alarmsignal ausgeben, damit der Akku nicht tiefentladen wird oder die angeschlossenen Geräte sich dann plötzlich wegen zu geringer Spannung abschalten. Dies könnte dann über den externen Anschluss (PB1) erfolgen. Derweil ist dies im Programm aber noch nicht vorgesehen.

Digitalteil

Als Mikrocontroller wird ein ATmega168 eingesetzt. Er bietet einen Timer, bei dem der Zählerstand über einen Hardwareeingang gesichert und im nächsten Schritt softwareseitig verarbeitet werden kann. Dies ist besonders bei dieser Anwendung von Vorteil, da genaue Zeitmessungen mit schneller Umschaltung und Auswertung nötig sind, um exakt gewandelte Werte zu erhalten. Da auch die Zählerfrequenz auf den kleinsten Strom abgeglichen werden muss, wurde ein Quarz mit einer Frequenz von f = 16MHz gewählt. Hierdurch werden ausreichend große Zählerstände und Auflösungen erreicht. Des weiteren ist an den Mikrocontroller das Display zur Anzeige der ausgerechneten Werte an PORTC angeschlossen. Die Beleuchtung des Displays kann wahlweise softwaremäßig über den OC2-Ausgang des Mikrocontrollers gedimmt oder auf konstante Helligkeit eingestellt werden. Der Kontrast kann Hardwareseitig mit einem Potentiometer eingestellt werden. Anderweitig verfügt der Mikrocontroller über einen zweiten externen PIN, an den über eine Stiftleiste ein Piezosummer angesteuert werden kann. Er dient zur akustischen Warnmeldung bei Unterschreitung der Akkuspannung, um ein zu tiefes Entladen zu verhindern.

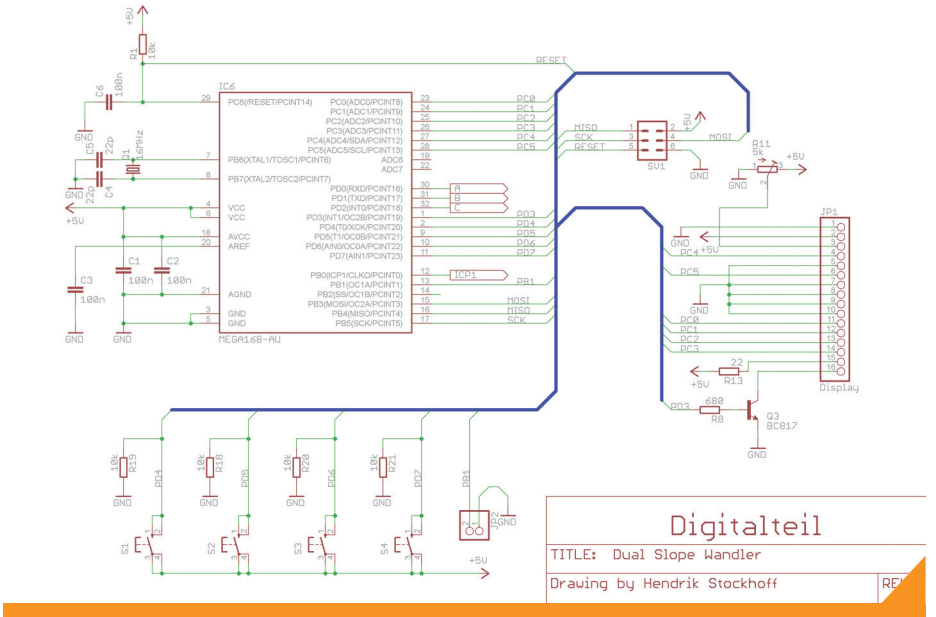


Abb. 6: Digitalteil

Funktion der Schaltung

Der eigentliche Dual Slope Wandler ist im Gegensatz zur obigen Erläuterung etwas anders aufgebaut (siehe Funktionsprinzip und Abbildung 3). Das Integrationsglied ist in dieser Anwendung schon ein wenig negativ Vorgespannt, um einen definierten Umschaltpunkt zu erhalten. Dadurch ist gesichert, dass der Komparator nicht bei kleineren Masseschleifen umschaltet. Die Vorspannung beläuft sich auf -0.025V. Dadurch, dass der Operationsverstärker immer versucht, zwischen seinen beiden Eingängen (PIN 12 u. 13, IC3D) einen virtuellen Nullpunkt zu erzeugen, ist die Vorspannung auch an PIN 13 anzutreffen. Sie gibt nun, wie oben schon gesagt, die Schaltschwelle des Komparators an PIN 2 vor. Sobald nun die Messspannung an den Eingang des Integrators angelegt wird, lädt sich der Kondensator negativ auf. Dies geschieht über einen fest vom Mikrocontroller vorgegebenen Zeitraum. Ist die fest vorgegebene Zeit verstrichen, wird die Referenzspannung an den Integrator angelegt. Der Integrationskondensator wird nun mit positiver Spannug aufgeladen. Wird jetzt der definierte Schaltpunkt überschritten, also ist die Spannung des Integrationskondensators größer als -0.025V, kippt der Komparator um. Komparator mit logischem Ausgang:

y = 1 für U1 > U2

U1 : Eingangsspannung nichtinvertierender Eingang

U2 : Eingangsspannung invertierender Eingang

y : logisches Ausgangssignal

Der Komparator gibt den Transistor am Ausgang frei. Der Transistor legt ein High-Signal an den ICP-Pin des Mikrocontrollers. Nachdem nun die Referenzspannung erfolgreich Integriert wurde, der Komparator den Transistor freigegeben hat, wird der Integrationskondensator durch den Multiplexer kurzgeschlossen. Dadurch wird der Kondensator auf das Potential der Vorspannung aufgeladen (UI0 = -0,025V). Zur Erläuterung sei hier nochmal ein Oszillogramm dargestellt, dass die Spannungen an den Messpunkten MP1, MP2, MP3 und MP4 zeigt (Abbildung 7 und 8).

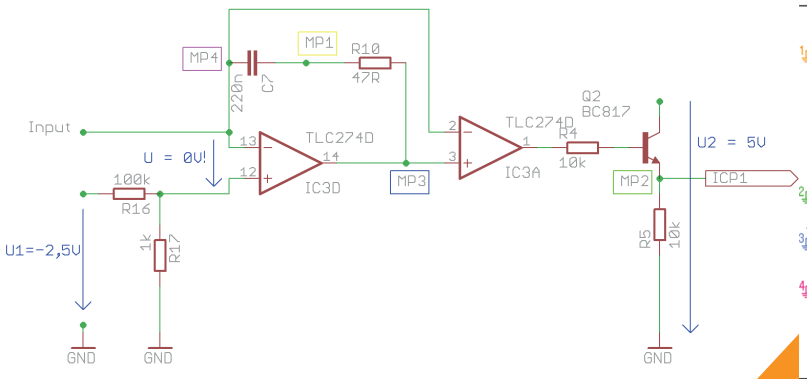


Abb. 7: Ausschnitt Analogteil mit MP1, MP2, MP3 und MP4

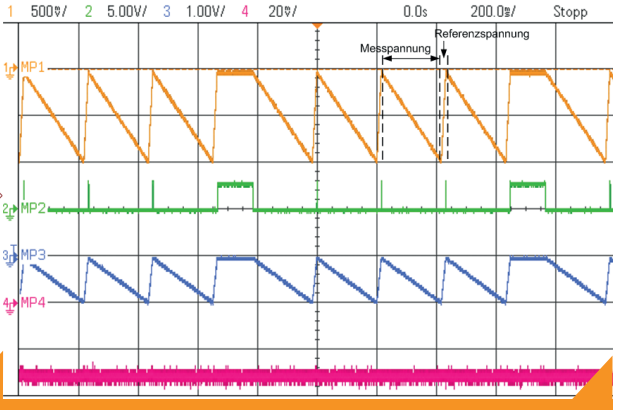


Abb. 8: Oszillogramm zur Erläuterung der Spannungen

Das Platinenlayout

Die Platine des Messgerätes ist zweiseitig aufgebaut (es ist aber auch möglich, diese einseitig zu fertigen, da alle Drahtbrücken dementsprechend ausgelegt sind). Sie beläuft sich auf eine Größe von 71,11mm x 107,3mm.

Auf der Lötseite, bzw. auf der Unterseite, wurden die SMD-Bauteile, wie der Mikrocontroller, Operationsverstärker und Multiplexer untergebracht. Auf der Oberseite befinden sich neben den konventionellen Bauteilen (Spannungsversorgung und Display) auch noch Drahtbrücken, die eine sichere Verbindung zwischen der Unterseite und der Oberseite herstellen. Zusätzlich wurde auf der Oberseite noch eine Beschriftung zur Anschlussbelegung des Messgerätes eingefügt. Das Layout des Messgerätes wurde speziell darauf ausgelegt, möglichst keine Masseschleifen zu erzeugen. Daher besitzt jeder Schaltungsteil (Spannungs-, Analog- und Digitalteil) seine eigene Masseleitung. Es wurde kein Polygon als Massefläche gewählt, da dort nicht explizit der Stromfluss ermittelt werden kann. Zusätzlich wurden die Leiterbahnlängen im Analogteil möglichst kurz gehalten, um einem eventuellen Spannungsfall vorzubeugen.

Im rechten Bereich wurde der ISP-Stecker zur Programmierung des Mikrocontrollers ATmega168 herausgeführt. Das Display wird später mit ausreichend langen Abstandsbolzen über die Elektronik gesetzt. Dabei muss die Länge der Abstandsbolzen entsprechend der Bauhöhe der Kondensatoren und des Shunts gewählt werden. Für die Befestigung der Abstandsbolzen sind bereits vier Bohrungen hervorgesehen. Die 16-polige Stiftleiste verbindet die Platine mit dem Display. Außerdem kann später die Beleuchtung mithilfe des Mikrocontrollers gedimmt bzw. ein- und ausgeschaltet werden. Ein Trimmer für die Kontrastspannung ist auch vorgesehen. Da das Display auf die Top-Seite der Platine befestigt wird, musste die Anschlussklemme für die Spannungsversorgung (X1, X2) etwas nach links versetzt werden, um auch noch ein nachträgliches anbringen der Versorgungskabel zu ermöglichen. So muss bei einem Anschlusswechsel nicht jedes Mal das Display demontiert werden, um an die Schrauben der Anschlussklemme zu gelangen. Des weiteren wurden auch die vier Kurzhubtaster etwas unterhalb des Displays positioniert.

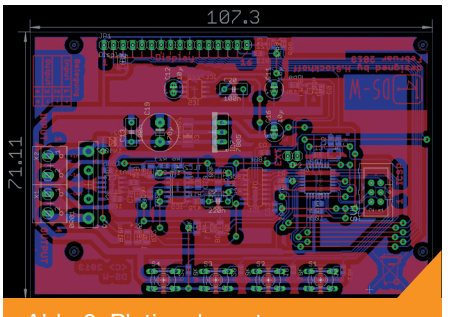


Abb. 9: Platinenlayout

Software

Die Software für das Messgerät ist in der Programmiersprache C geschrieben. Der Mikrocontroller übernimmt das Zeitmanagement für die Umschaltung der zu integrierenden Spannungen sowie die darauf folgende Auswertung und Umrechnung der Zählerstände. Schließlich wird der ermittelte umgerechnete Wert auf dem LCD-Display angezeigt.

Das Programm des Batteriemonitors ist so aufgebaut, dass zuerst eine Spannungsmessung durchgeführt wird. Anschließend wird die Strommessung bei nicht angeschlossener Last initialisiert. Dieser Stromwert dient dann als relativer Nullpunkt für die späteren Messungen. Im nächsten Schritt wird der Timer2 initialisiert. Er dient zur Zeitmessung. Dabei wurde der Top-Wert so gewählt, dass er alle 500µs ein Überlauf auftritt. Dadurch kann leichter eine volle Sekunde errechnet werden. Nachdem nun die Zeitmessung gestartet wurde, betritt das Programm eine Endlossschleife, in der eine kontinuierliche Spannungs- und Strommessung durchgeführt werden. Die gewandelten Werte werden dann umgerechnet auf dem Display angezeigt. Innerhalb der kontinuierlichen Spannungs- und Strommessung kann über die Bedientaster die Errechnung der Ladung Q gestartet, gestoppt und zurückgesetzt werden. Eine nachträgliches abgleichen des relativen Nullpunktes für die Strommessung ist auch möglich. Dieser Vorgang soll als nächstes noch einmal etwas detaillierter erläutert werden.

AD-Wandlung

Die AD-Wandler-Routine habe ich in der Software so gestaltet, dass sie schnell auf einen anderen Mikrocontroller übertragen werden kann. Es müssen lediglich die entsprechenden Umschaltung zur Anlegung der Spannungen am Multiplexer verändert werden. In der Headerdatei wird zunächst festgelegt, an welchem Port der Multiplexer angeschlossen ist und an welchem Port sich Hardwaremäßig der ICP-Pin des Mikrocontrollers befindet. Dazu einmal ein kleiner Ausschnitt aus dem Programm:

```
// Steuerbits des Multiplexers
#define A PD0
```

```
#define B PD1
#define C PD2

// PORT des Multiplexers
#define Multi _ PORTx PORTD

// DDR des Multiplexers
#define Multi _ DDRx DDRD
#define ICP _ PINx PINB
#define ICP _ Px     PB0
```

Hier sind auf einem Blick die wichtigen Anschlüsse des Multiplexers einsehbar und können schnell an den jeweiligen Mikrocontroller angepasst werden.

Damit das Signal auch gemessen werden kann, muss ein Timer eingesetzt werden, der die Aufladungszeiten in Form von Zählerständen ermitteln kann. Dazu eignet sich am Besten der 16-Bit Timer1 des ATmega168. Er verfügt zudem über einen Capture-Betrieb, bei dem der Zählerstand hardwaremäßig über einen Flankenwechsel am ICP-Pin im ICR1-Register gespeichert werden kann. Der Timer1 wird folgendermaßen initialisiert:

```
TCCR1B |= (1<<WGM12);
//Trigger auf steigende Flanke
//Noise Canceler eingeschaltet
TCCR1B |= (1<<ICES1) | (1<<ICNC1);
TCCR1B |= (1<<CS10) |
(1<<CS11); // Vorteiler 64
OCR1A = 0xFFFF;
// Input Capture Interrupt
TIMSK1 |= (1<<ICIE1);
sei();
```

Im nächste Teil kommt nun die eigentliche Wandler-Routine. Dort wird über eine switch-case Anweisung differenziert, ob der Strom oder die Spannung gemessen werden soll. Der spätere Zählerstand wird in der Variablen ad_wert abgespeichert. Welcher Steuereingang des Multiplexers zum Anlegen der korrekten Spannung eingeschaltet werden muss, wurde anhand einer Wahrheitstabelle aus dessen Datenblatt ermittelt. Es wird nun zuerst die zu messende Spannung an den Integrator angelegt. Im nächsten Schritt wird solange gewartet, bis der Komparator ein Low-Signal ausgibt. Erst danach wird der Zählerstand des Timer1 zurückgesetzt. Es wird nun das Messsignal über einen festen Zeitraum integriert. Festgelegt ist die Zeit durch die Variable timer_max. Nachdem das Register TCNT1 größer ist als die Variable timer_max, wird die Referenzspannung an den Integrator angelegt. Das Timer/Counter Daten-Register(TCNT1) wird zurückgesetzt. Der nun folgende Zählerstand ist der Entscheidende. Es wird so lange die Referenzspannung an den Integrator angelegt bzw. gewartet (while), bis die Schaltschwelle von -0,025V durchlaufen ist. Ist dies geschehen, erzeugt der Komparator eine steigende Flanke und der Zählerstand des Timer1 wird in dem ICR1-Register gespeichert. In der Interrupt-Routine für den Capture-Betrieb wird anschließend der Wert des ICR1-Registers in die Variable ad_wert kopiert. Nachdem die Interrupt-Routine beendet ist, wird der Integrator kurzgeschlossen und wieder entladen, um eine gleiche Anfangsbedingung für die nächste Wandlung zu erhalten. Nachfolgend ist dieser Vorgang nochmal in Form des Sourcecodes dargestellt (siehe auch Abbildung 10):

```
<avr/io.h>
<avr/interrupt.h>
unsigned int timer _ max = 50000; // entspricht 0,2s
// gewandelter Analogwert in Form eines Zählerstandes
unsigned int ad _ wert = 0;
ISR (TIMER1 _ CAPT _ vect)
{
   ad _ wert = 0;
   ad _ wert = ICR1;
}
// Analog-Digital-Wandlungen
void ad _ wandlung(unsigned char wert) {
   switch (wert) {
     case ‚i': Multi _ PORTx &=~ (1<<B); break;
     case ‚u': Multi _ PORTx &=~ ((1<<B) | (1<<A)); break;
     }
            // Warte, solange der ICP-Pin noch High ist
   while (ICP _ PINx & (1<<ICP _ Px));
   TCNT1 = 0; // Timer zurücksetzen
   while (TCNT1 < timer _ max);
   // Anlegen der Referenzspannung
   switch (wert) {
     case ‚i': {
            Multi _ PORTx &=~ (1<<C);
            Multi _ PORTx |= (1<<B);
            }
            break;
     case ‚u': {
            Multi _ PORTx &=~ (1<<C);
            Multi _ PORTx |= (1<<B) | (1<<A);
            }
            break;
     }
   TCNT1 = 0;
   while(!(ICP _ PINx & (1<<ICP _ Px)));
   // Multiplexer kurzschließen
   Multi _ PORTx |= (1<<C);
}
```

Listing 1: AD-Wandlung

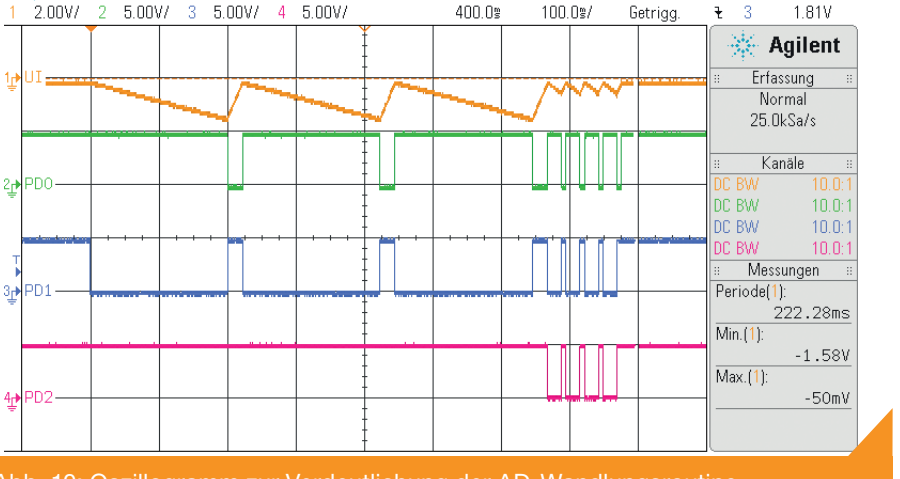


Abb. 10: Oszillogramm zur Verdeutlichung der AD-Wandlungsroutine

Korrektur Gleichtaktunterdrückung

Mit obiger Software ist es jetzt möglich, den Strom bei konstanter Spannung zu messen. Da das Messgerät jedoch durch einen Akku betrieben wird, sinkt die Spannung nach einer gewissen Zeit zwangsläufig. Es muss jetzt festgestellt werden, ob der zuvor initialisierte relative Nullpunkt auch bei sinkender Spannung gleich bleibt bzw. der gemessene Wert größer ist als der relative Nullpunkt. Um dies zu überprüfen, wird das Messgerät so programmiert und eingeschaltet, dass der Zählerstand und der umgerechnete Zählerstand gleichzeitig auf dem Display angezeigt werden.

Es wurde nun folgende Messreihe für den maximalen Aufladungsbereich von timer_max = 50000 aufgenommen:

Betriebs-spannung U in V	Betriebs-spannung U in Zähler	Strom I in A	Strom I in Zähler
14	2251	0	543
13	2088	-0,005	517
12	1931	-0,009	497
11	1773	-0,013	473
10	1622	-0,018	450
9	1468	-0,023	427

Tabelle 1: Messreihe bei timer_max = 50000

Messreihe bei timer_max = 30000:

Betriebs-spannung U in V	Betriebs-spannung U in Zähler	Strom I in A	Strom I in Zähler
14	2251	0	319
13	2088	-0,005	308
12	1931	-0,009	295
11	1773	-0,013	282
10	1622	-0,017	268
9	1468	-0,022	256

Tabelle 2: Messreihe bei timer_max = 30000

Anhand der Messwerte kann man gut erkennen, dass die Abweichungen bei beiden Messreihen annähernd linear sind. Somit können daraus folgende lineare Gleichungen erstellt werden:

Gleichung 1 für timer_max = 50000:

$$y = 0,1471 \cdot x + 211,5$$

y : Strom I in Zähler

x : Betriebsspannung U in Zähler

Gleichung 2 für timer_max = 30000:

$$y = 0,0856 \cdot x + 129,86$$

y : Strom I in Zähler

x : Betriebsspannung U in Zähler

Die linearen Gleichungen korrigieren die relativen Nullpunkt in Abhängigkeit von der Eingangsspannung. Die y-Achsenabschnitte n = 211,5 und n = 129,86 werden später noch durch eventuelle weitere Abweichungen im Programm automatisch abgeglichen. i = (((((((float)mittel_i-null_wert_r+nach_null) * ref) / timer_max))/25)*100)*korrekt;

Bei den Wandlungen werden die Zählerstände in einem Array gespeichert. Der erste gewandelte Wert wird jeweils verworfen, da dort noch Differenzen durch eventuelle Vorspannungen auf dem Integrationskondensator vorhanden sein können. Anschließend wird der Mittelwert aus den restlichen Messwerten gebildet und in der zugehörigen Variable (mittel_i, mittel_u) gespeichert.

Zeitmessung

Die Zeitmessung wurde, wie oben schon kurz erwähnt, mit dem Timer2 des ATmega168 realisiert. Initialisiert wird der Timer2 im CTC-Mode. Hier kann die Obergrenze durch das Register OCR2A festgelegt und so die Periode T der Überläufe variabel gestaltet werden. Es wurde hier der Wert 250 bei einem Timer Vorteiler von 32 gewählt. Begründung:

Bei f = 16000000 Hz ist $f_{Vorteiler}$ 500000 Hz.

$$T = \frac{1}{f_{vorteiler}}$$

$$\Leftrightarrow T = 0,000002s$$

Überläufe bei einem Top-Wert von 250:

$$T = 0,000002s \cdot 250$$

$$\Leftrightarrow T = 500\mu s$$

Um jetzt auf eine volle Sekunde zu gelangen, müssen demnach 2000 Überläufe erfolgt sein.

Automatische Messbereichsumschaltung

Da der Integrationskondensator nur eine gewisse Menge an Ladung aufnehmen kann, darf das Eingangssignal ein gewisses Maximum bei gleich bleibender Aufladungszeit nicht überschreiten. Dies ist jedoch nicht sehr vorteilhaft, da so nicht der vollständige Zählerbereich der maximalen Aufladung ausgenutzt wird. Außerdem wären bei kleiner Aufladungszeit die Zählerstände zu gering, was eine kleinere Auflösung zur Folge hat. Daher wird bei dem Batteriemonitor eine automatische Messbereichsumschaltung durchgeführt. Es wird bei großen Strömen, die große Spannungsfälle am Shunt bewirken und verstärkt werden, die maximale Aufladungszeit herunter gesetzt. Hierfür dient der zweite Aufladungswert timer_max = 30000. Die Aufladung ist folgendermaßen gestaffelt:

- Messbereich 1: timer_max1 = 50000. Die Aufladung beträgt umgerechnet 0,2s. Der Bereich gilt von 0 < I < 3,5A.
- Messbereich 2: timer_max2 = 30000. Die Aufladung beträgt umgerechnet 0,12s. Der Bereich gilt von I ≥ 3,5A.

Die Messbereichsumschaltung erfolgt nach der Umrechnung des Zählerstandes in einen Stromwert. Zusätzlich werden in der Messbereichsumschaltungsroutine noch die entsprechenden relativen Nullpunkte durch die lineare Gleichung der Gleichtaktunterdrückung berechnet. In der Variablen nach_null stehen Abweichungen von dem Nullpunkt null_wert_r.

Nullabgleich

Der erste Nullabgleich des Displays ist komplett automatisiert. Es muss lediglich über die Taster (Taster 4) der Fortschritt bestätigt werden. Das heißt, es muss bestätigt werden, das keine Last am Batteriemonitor angeschlossen ist. Andernfalls kann der Nullpunkt nicht korrekt initialisiert werden. Um den Nullpunkt null_wert_r mit Abweichung in nach_null zu setzen, wird mit der Ermittlung der Betriebsspannung begonnen. Hierfür werden 4 gewandelte Spannungswerte in Form von Zählerständen in ein Array (speicher_u[4]) geladen. Im nächsten Schritt wird der Mittelwert aus den letzten drei Werten gebildet. Nun kann mit der Spannung der relative Nullpunkt null_wert_r berechnet werden. Die Aufladungszeit timer_max_u beträgt timer_max_u = 5000. Also 0,2s. Es wird also die Differenz des durch die lineare Gleichung ermittelten relativen Nullpunktes und dem ersten ohne angeschlossene Last gemessenen Zählerstandes gebildet. Dieser Wert nennt sich hier nach_null. Auch er muss für beide Messbereiche (timer_max1 und timer_max2) separat bestimmt werden.

Bedienung

Zu beachten ist, dass der Batteriemonitor nur mit Spannungen von 8V/DC bis 14V/DC betrieben werden darf!

Die maximale Belastung beträgt 6A.

Der Batteriemonitor wird über die 4 Taster bedient. Taster 1 dient zum Starten und Stoppen der Ladungsermittlung. Taster 2 setzt die Zeit und die Ladungsermittlung im „Stop"-Zustand zurück. Mit Taster 3 kann bei gestoppter Ladungsermittlung ein erneuter Stromabgleich vorgenommen werden. Es wird dann der neue Nullpunkt gesetzt. Taster 4 wird nur während der Initialisierung benötigt. Mit ihm werden Bestätigungen ausgeführt.

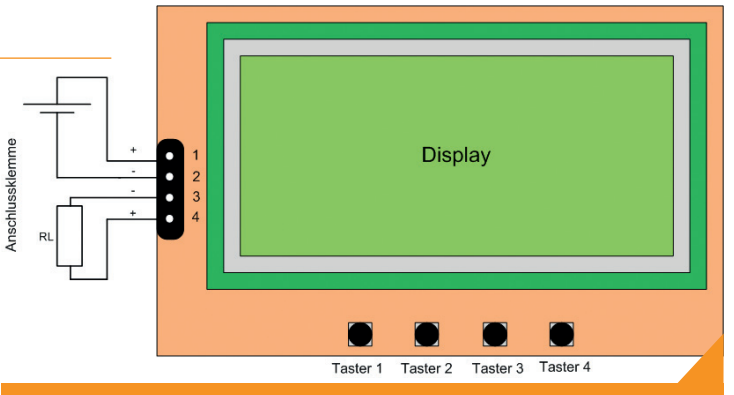
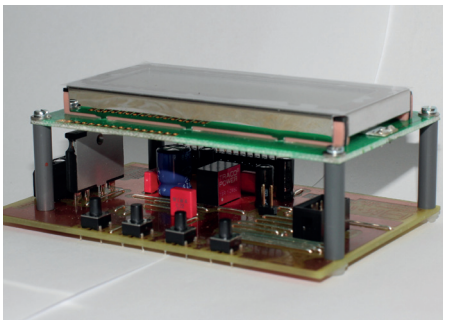


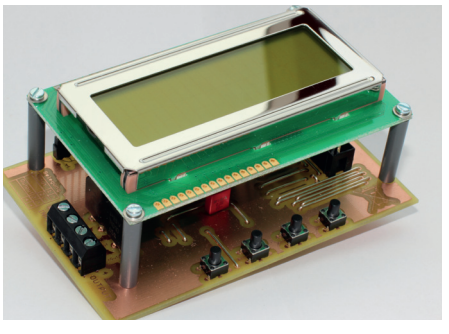
Abb. 11: Technologieschema

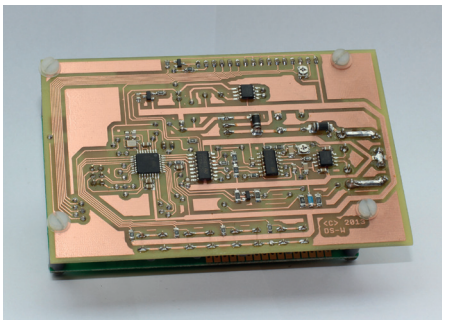
Probleme bei der Entwicklung

Die Entwicklung des Gerätes hat doch sehr viel Zeit in Anspruch genommen. So wurde zuerst der Wandler auf einem Breadboard getestet, um das Verhalten zu untersuchen. Als Operationsverstärker haben dort OP07 gedient, die keinen internen CMOS-Aufbau haben. Dadurch wurde das Ergebnis aus der Wandlung immer wieder verfälscht. Hinzu kamen noch Masseschleifen durch die langen Steckleitungen, die das Ergebnis des Differenzverstärkers immer wieder zunichte machten. Außerdem war zu diesem Zeitpunkt auch nicht klar, dass der Kondensator nicht bis zur Sättigung aufgeladen werden darf(wird in der Fachliteratur meist nicht so deutlich). Erst nach einigen Versuchen, konnte festgestellt werden, dass nur ein kleiner aber dennoch ausreichender Bereich der linearen Aufladungskurve des Integrationskondensators nötig ist, um ein gutes Ergebnis zu erzielen. Kurz darauf wurde dann die erste Testplatine entworfen und gefertigt. Dort konnten, durch gezieltes Routen der Leiterbahnen, Masseschleifen vermieden werden. Das Projekt konnte nun gezielt verbessert und Störungen beseitigt werden. Eine zweite Testplatine mit konventionellen Bauteilen und einem CMOS-OP wurde auch noch hergestellt, um dessen Verhalten zu untersuchen. Wie oben in dem Funktionsprinzip schon dargestellt wurde, bestätigte sich die These, das CMOS-OPs für diesen Typ von Integratorschaltung am Besten geeignet sind. Zum Schluss wurde dann das Endprodukt (Abbildung 1) hergestellt. An ihm wurden die Aufladungszeiten angepasst und die Bedienung durch Taster noch hinzugefügt.

Bilder







Downloads

[1] Sourcecode www.mikrocontroller.net/wikifiles/8/80/Sourcecode.zip

[2] Schaltplan im Eagle Format www.mikrocontroller.net/wikifiles/1/13/Schaltplan_im_Eagle-Format.zip □

WEdirekt.

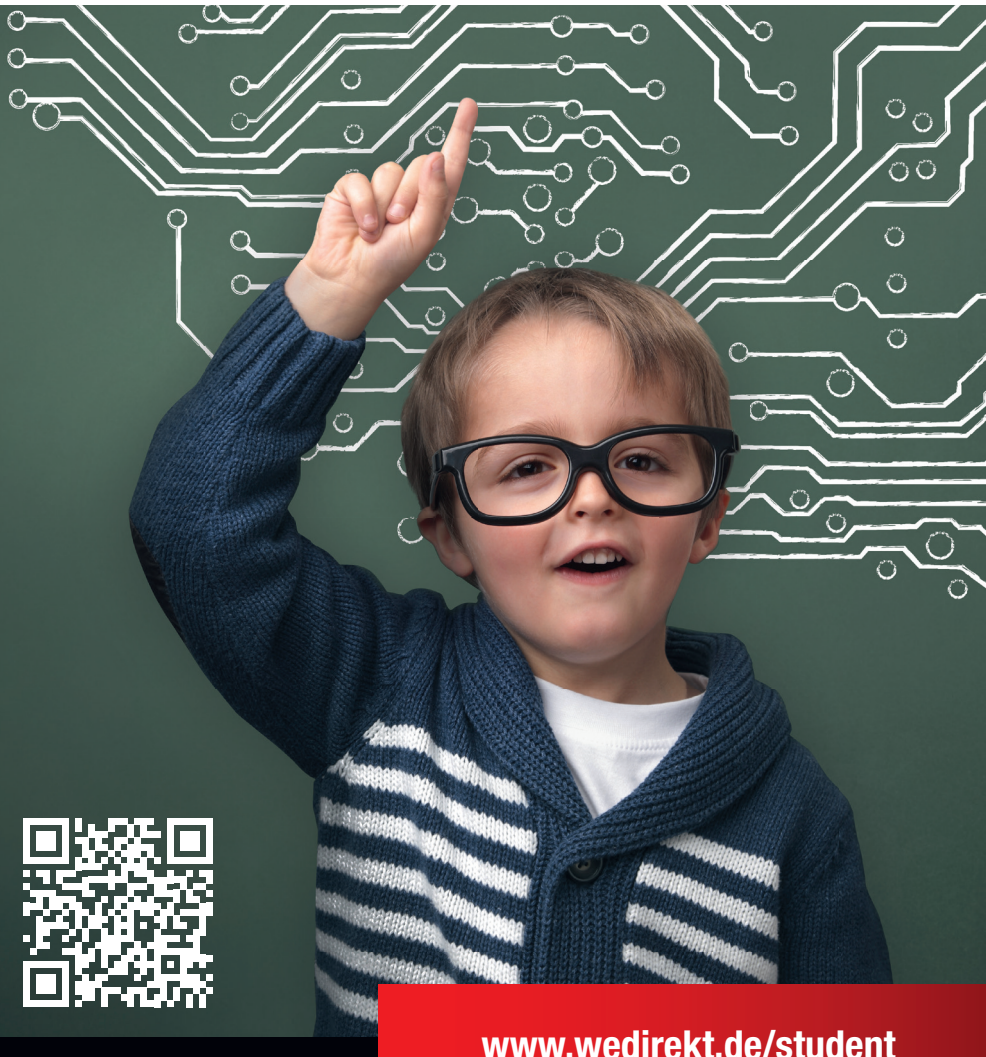
- Leiterplatten ab 1 Stück
- Industriequalität zu attraktiven Preisen
- Expresslieferung ab 2 AT in chem. Ni/Au
- Unkomplizierte Konfiguration und Bestellung

ONLINE PCBs

powered by Würth Elektronik

www.wedirekt.de

WEdirekt.



PCB Online Shop für Schüler und Studenten – jetzt Vorteile sichern.

Wir sind Dein Partner!

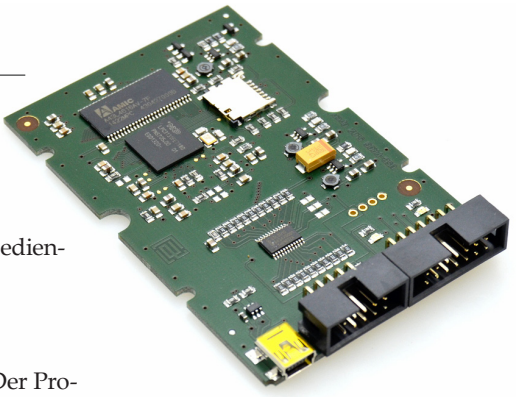
powered by Würth Elektronik

www.wedirekt.de/student

Marktplatz / Neuigkeiten

Die Ecke für Neuigkeiten und verschiedene Produkte

Und so sieht er aus - der neue USBprog 5

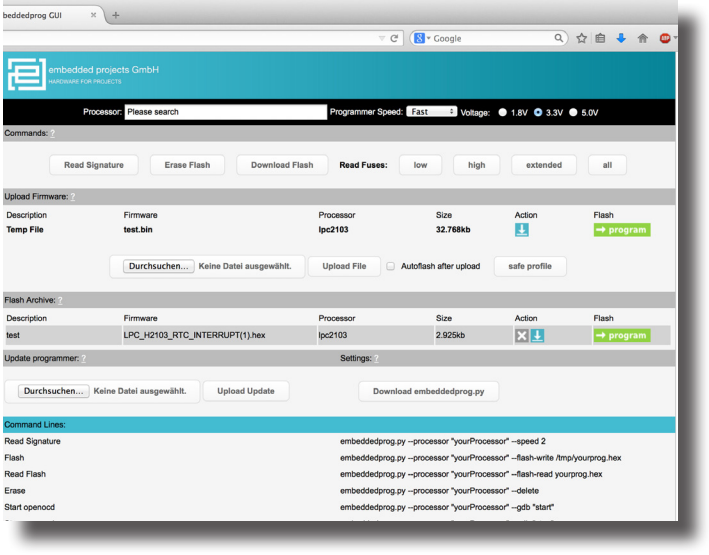


Das kann er alles:

- Flashen von Firmware in ARM und AVR Prozessoren
- Debuggen von ARM Prozessoren
- Per Browser oder Kommandozeilentool bedienbar bzw. über API steuerbar

Das ist sein Innenleben:

Auf dem USBprog 5 arbeitet ein Linux. Der Programmer ist mit freien Tools wie avrdude und OpenOCD umgesetzt. Um die notwendige Performance zu erreichen sind die JTAG Signale für das Programmieren über die SPI Leitungen geführt. Als zentrale Software gibt es einen in Python geschrieben Server, der die Anwendungen ansteuert und die Kommunikation über das Netzwerk ermöglicht.



Als Highlight bietet der neue USBprog 5.0 eine Webanwendung direkt auf dem integrierten Webserver an. So kann man schnell ohne kompliziertes Konfigurieren einen Prozessor programmieren oder in den Debugmodus bringen. Mittels GDB kann man sich per Netzwerk auf die Zielschaltung verbinden.

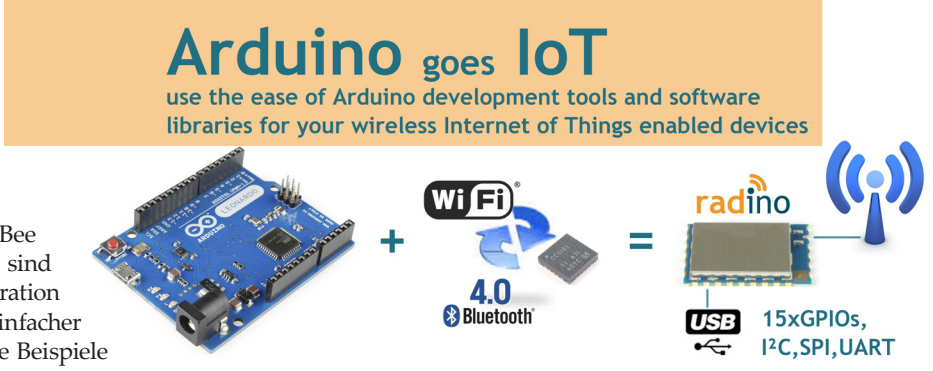
Da die USB Verbindung mit allen gängigen Betriebssystemen funktioniert, ist niemand ausgeschlossen :)

Geplant ist noch weitere Firmware bzw. weitere Prozessoren. Funktionen können später per Softwareupdate nachgerüstet werden.

Hersteller: embedded projects GmbH, Augsburg www.usbprog.org

Modul mit ESP8266

Die radino Funkmodule vereinen Arduino Micro mit verschiedenen Funk-Schnittstellen, hochintegriert und untereinander 100% Pin-zuPin kompatibel.



Das radino WiFi ist mit dem ESP8266 SoC von Espressif Systems ausgestattet und bietet dadurch je nach geladener Firmware Kompatibilität zu Funkstandards wie WLAN, ZigBee oder 6LowPan, auch Per-to-Peer Verbindungen sind damit leicht möglich. Durch die einfache Integration der radino-Module in die Arduino-IDE ist ein einfacher Upload neuer Firmware möglich und zahlreiche Beispiele wie z.Bsp ein Webserver sind verfügbar.

Hersteller: In-Circuit GmbH, Dresden www.in-circuit.de, www.radino.cc

www.Facebook/radino.cc

Interesse an einer Anzeige?

info@embedded-projects.net

embedded-projects
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Werdet aktiv!

Das Motto: Von der Community für die Community !

Das Magazin ist ein Open Source Projekt.

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

journal@embedded-projects.net

Regelmäßig

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

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

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

Sponsoren gesucht!

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

Bitte frei machen

embedded projects GmbH
Holzbachstraße 4
D - 86152 Augsburg

Name / Firma

Straße / Hausnummer

PLZ / Ort

Email / Telefon / Fax

◯ Ich möchte jede zukünftige Ausgabe erhalten

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

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

Impressum

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

Verteilt durch:

embedded projects GmbH
HARDWARE FOR PROJECTS

Veröffentlichung: 4x / Jahr
Ausgabenformat: PDF / Print

Auflagen Print: 2500 Stk.
Einzelverkaufspreis: 1 €

Layout / Satz: EP
Druck: flyeralarm GmbH
Titelbild: Claudia Sauter

MIT FLEXIBILITÄT MEHR BEWEGEN.

FLEXIBLE LEITERPLATTEN ONLINE BESTELLEN.

STARR FLEX 4 – Bis zu 4 Lagen auf Anfrage möglich!

LEITON
RECHNEN SIE MIT BESTEM SERVICE

Erfolgreich ist, wer flexibel auf neue Marktanforderungen reagiert. Gefragt sind heute kompakte, komplexe sowie sehr leichte Aufbauten, welche dynamische Biegebelastbarkeit aufweisen und dabei höchste Zuverlässigkeit der elektrischen Verbindungen bieten. Die Lösung lautet **flexible Leiterplatten von LeitOn.** Damit sparen Sie gleich dreimal: **Platzersparnis** durch optimales Anpassen der Baugruppen an die Gehäuse, **Gewichtsersparnis** aufgrund sehr dünner Folien sowie **Kostenersparnis** wegen der Reduktion von Steckverbindungen. Und Sie gewinnen **mehr Flexibilität** dank persönlicher Beratung am Telefon, einem kompetenten Außendienst und Angeboten auch per E-Mail in Windeseile. Sie können bei LeitOn immer mit bestem Service rechnen.

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