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

[EDITORAL]

ENGINEERING EVERYWHERE

Neue Projekte zum nachbauen oder weiterentwickeln

[PROJECTS]

- **FLASHCRAFT** - Funkboard RFM12
- **Embedded Linux** - Stoppuhr für Interrupts
- **USB IO Expander**
- **UART** ohne Kabel
- **MyCPU** - A Homebrew Computer
- **Grasshopper** - Standard C Programmierung
- **Disko - A GUI toolkit** for set-top boxes
- **NIOS II Embedded Evaluation Kit** - TMC-Staumeldungen

WELCOME

Ausgabe 3 von Embedded-Projects Journal

Aller guten Dinge sind drei?

Tja, wer hätte das gedacht - drei Ausgaben des freien Journals existieren bereits. Ein Open-Source Projekt Zeitschrift, das zusätzlich zur PDF-Datei auch als echte Papierversion verfügbar ist. Dass es klappen kann wurde bewiesen.

In dieser Einleitung möchte ich die Zeit nutzen einen Motivationsschub für alle Bastler, Projektbetreuer, Hochschulprofessoren, Ausbildungsleiter usw. zu starten. Schreibt Artikel für das Projekt Zeitschrift!

Tag für Tag entstehen viele Interessante Projekte, Versuche oder kleine Testschaltungen, die nach getaner Arbeit wieder in der Schublade verschwinden. Doch genau diese Informationen bedeuten für Entwickler, Baster & Co. einen wirklichen Mehrwert. Mein Wunsch: Nutzt das „embedded projects Journal“ als freie Plattform, solche Artikel und Berichte zu veröffentlichen. Es müssen nicht immer glänzende „Superschaltungen“ vorgestellt werden, es kann auch eine Informationsfundgrube vieler kleiner Dinge sein. Meist genügt es, sich nach einem Versuch noch ca. 1-2 Stunden hinzusetzen und alles kurz nieder zu schreiben.

Das Open-Source Heft „embedded projects Journal“ braucht noch mehr spannende Artikel und Berichte. Auch Sponsoren für Anzeigen sind immer willkommen! An dieser Stelle nochmals Dank an unseren Hauptsponsor IBV-Echtzeit- und Embedded aus Augsburg.

Bei Fragen wendet euch gerne wie immer an mich.

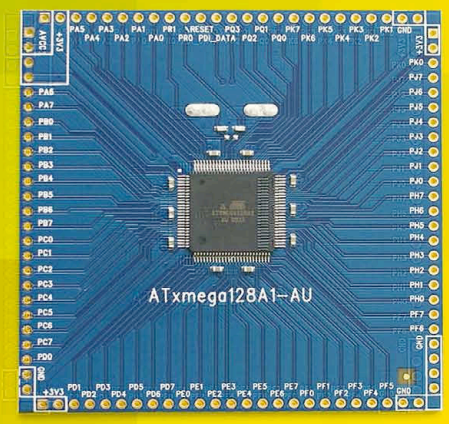
Benedikt Sauter
sauter@embedded-projects.net

shop.embedded-projects.net

THE ONLINESHOP - HARDWARE FOR YOUR PROJECTS

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

3,95 EUR Versand weltweit



XMEGA BOARD

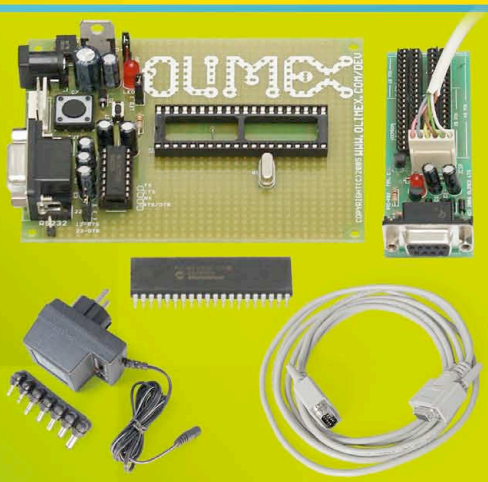
ATxmega128A1 TQFP-100 auf einer DIP-Adapterplatine

Der Prozessor ist auf eine TQFP-Adapterplatine gelötet, so dass er auf 2,54 mm Lochraster weiterverbaut werden kann.

- 128K Byte programmierbarer Flash Datenspeicher
- 8K Byte Bootloader-Speicher,
- 8K Byte SRAM und 2K Bytes EEPROM
- 4-Kanal DMA Kontroller
- 8-Kanal Event System und bis zu 32 MIPS Durchsatz bei 32 MHz
- 12-bit ADC, 12-bit DAC und Analog Comparator
- AES und DES Kryptoeinheit, 16-bit Timer Counters
- USART, SPI, und TWI.

NEU

ATxmega128A1 TQFP-100
16,95 €



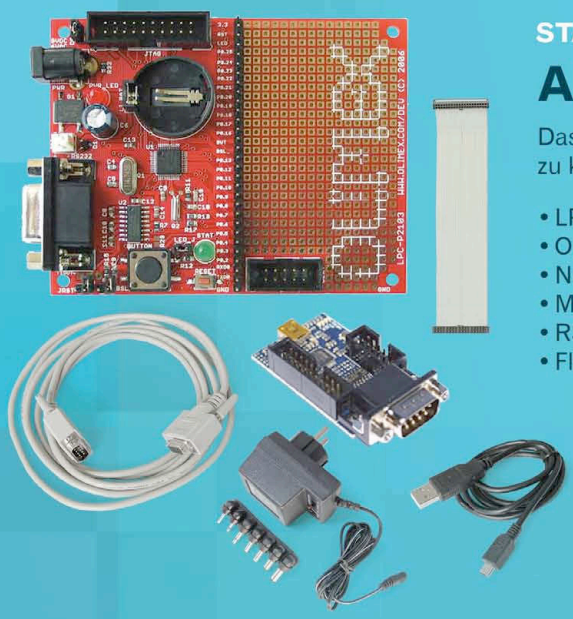
STARTERKIT

PIC Starterkit

Mit den beiliegenden Bestandteilen können Sie direkt mit der Entwicklung von Schaltungen für 40-polige Microchip Prozessoren anfangen. Der RS232 Anschluss auf dem Board ermöglicht eine einfache Kommunikation mit einem PC. Das Board hat Taster und LEDs. Das Board unterstützt 3V und 5V Controller.

- PIC Entwicklungsplatine für 40 polige PICs (PIC-P40-20MHz)
- PIC PG2 Porgrammer
- PIC 18F4550
- RS232-Kabel
- Steckernetzteil

PIC Starterkit
39,90 €



STARTERKIT

ARM7 Starterkit

Das ARM7 Starterkit beinhaltet alles, um direkt in die ARM-Welt einsteigen zu können. Was man braucht ist nur ein Computer mit USB-Schnittstelle.

- LPC2103 Board
- OpenOCD USB Adapter (Programmierkabel)
- Netzteil (300mA)
- Mini USB-Kabel
- RS232 Kabel
- Flachbandkabel (20 polig für JTAG-Verbindung)

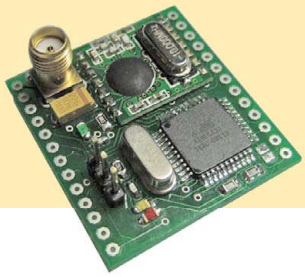
ARM7 Starterkit
89,90 €

http://shop.embedded-projects.net

FLASHCRAFT - Funkboard

Funken mit dem RFM12

Florian Scherb <florian.scherb@web.de> www.flashcraft.de



Einleitung

Fast jeder spielt einmal mit dem Gedanken Irgendetwas über Funk zu verbinden. Angefangen von kleinen Laboraufbauten, bis hin zur hauseigenen Alarmanlage oder der mit dem Wecker gekoppelten Kaffeemaschine. Doch stellt sich schnell die Frage, wie denn die Funkverbindung mit möglichst geringem Aufwand realisiert werden soll?
Dieser Artikel soll einen Anstoß und eine Lösung anhand eines kompletten Projekts für diese Frage liefern.

Projektvorstellung

Da das Übertragungsvolumen bei vielen Hobbyanwendungen relativ gering ist (nur Bruchteile bis wenige Kilobyte pro Sekunde) und sich die Reichweite in der Regel auf die eigenen Räumlichkeiten beschränkt, stellen die Funkmodule dieser Kategorie wohl den interessantesten Teil dar.
In diesem Artikel will ich ein Projekt vorstellen, das einem bereits große Teile an Arbeit für die Einrichtung einer Funkübertragung abnehmen kann.

Das Projekt läuft unter dem Namen **FLASHCRAFT Funkboard.**

Es basiert auf dem RFM12 Modul (siehe Erläuterung unten auf dieser Seite) und einem ATmega32. Das Projekt beinhaltet
· einen Schaltplan + Platinenlayout
· einen umfangreichen C-Code
· ein Terminalprogramm für den PC
· eine ausführliche Dokumentation

Umfang

Die Motivation war es, ein universell einsetzbares Modul zu entwickeln, das einfach in der Anwendung, leicht konfigurierbar und auf das individuelle Projekt anpassbar ist ohne dass die Zielschaltung, in die das Board eingebaut wird irgendwelche Informationen über das Funksystem kennen muss.
Das Projekt beinhaltet eine komplette Funklösung, die man bei Bedarf auch frei abändern kann. Die wichtigsten Leistungsmerkmale sind im nebenstehenden Kasten aufgelistet.

Leistungsübersicht

· 433MHz oder 868MHz ISM Band
· Keine zusätzliche Funklizenz nötig
· Bidirektional, halbduplex
· Reichweite zwischen 100-300m
· Funknetzwerk mit bis zu 125 Boards!
· Im Betrieb voll konfigurierbar, autom. Laden von Konfigurationen beim Startup
· Sicherheitfeatures wie CRCs vorhanden
· Ansteuerung über UART / RS232
· Maximale Paketlänge (momentan) 70 Nutzdatenbytes
· Interne Datenpufferung von ausgehenden und eingehenden Daten
· Stromverbrauch 44mA + Sleepmodes
· 5V oder 3,3V-Betrieb möglich
· Festspannungsregler ZLDO xxx, RS232-Treiber, u.A. optional direkt onBoard!
· Plus PC Software, Dokumentation und vielen weiteren Features!

Abbildung 1: RFM12 in der S-Version

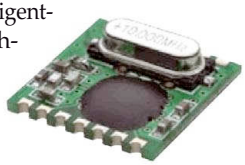
Wie funktioniert´s?

Die Schaltung mit AVR, RFM12 und der zusätzlichen Peripherie bilden eine in sich geschlossene „Blackbox“, die alles beinhaltet, was ein Funksystem haben muss.

„Und das heißt…?“

Oft hat man zwar passende Funkmodule gefunden, steht nun aber vor der Aufgabe diesen Modulen Leben einzuhauchen. Was am Anfang noch einfach klingen mag, kann schnell in einen wochenlangen „Programmierkrieg“ ausarten, wenn man vor hat das Funksystem etwas umfangreicher zu gestalten. Hier steht man an einem Punkt an dem man schnell mehr Zeit in die Umsetzung der Funkübertragung investiert als in das eigentliche Projekt selbst!

Das hier vorgestellte Projekt soll genau dies vermeiden und eine fertige Funklösung für durchaus anspruchsvollere Aufgaben bieten. Die externe Schaltung selbst (z.B. ein Datenlogger) muss keinerlei Dienste zum Funksystem beitragen und kann komplett ihrer eigentlichen Aufgabe nachgehen.



Das RFM12

Den Kern bildet das RFM12 Funkmodul der Fa. HOPE Microelectronics. Dieses Modul ist mit gerade einmal 2*2cm Grundfläche und rund 6-7 € Kosten klein und preiswert. Aber auch seine Leistungsdaten sind für dieses Platinchen besser als zunächst man es erwarten würde. Das Modul wird mit Spannungen zwischen 2,2 bis 5,4V betrieben und verbraucht dabei in etwa zwischen 23mA (Senden) und 14mA (Empfangen). Das Modul gibt es für die Frequenz-bereiche 433MHz und 868MHz. Die maximale Über- tragungsrate beträgt 115,2 kBaud. Weiter hat es eine Sendeleistung von 4dBm und eine Empfindlichkeit von -100dBm bei einer Bandbreite zwischen 67kHz und 400kHz. Die Ansteuerung erfolgt mittels SPI.

Die Schaltung

Die Schaltung für das Board beschränkt sich im Wesentlichen auf die Ansteuerung des RFM12 sowie einigen Peripherieelementen, die bei Bedarf auch entfallen können.

Interner Aufbau des Moduls

Das RFM12 wird seriell über SPI (Serial Peripheral Interface) angesteuert, die maximale Taktrate beträgt rund 2MHz. Das Modul verfügt über einige Zusatzanschlüsse, deren Erläuterungen den Rahmen jedoch sprengen würden. Einzig auf die (nicht-)Verwendung des Interrupt-Pin NIRQ soll hier explizit eingegangen werden:

Der NIRQ wird bei bestimmten internen Ereignissen getriggert und dient als zusätzliche Statusleitung neben dem SPI. Eigentlich eine feine Sache, so ließe sich dadurch das Pollen im Empfangsbetrieb vermeiden, wäre da nur nicht die etwas bescheidene Umsetzung.

Der NIRQ ist leider derart überladen, dass die softwareseitige Auswertung der Frage, welches Ereignis den NIRQ nun ausgelöst hat, seinen Vorteil wieder verspielt. Aus diesem Grund wirdder Pin hier nicht verwendet. Allerdings sollte man sich bewusst sein, dass ein Pollen dadurch unumgänglich wird.

Alle folgenden Komponenten sind optional:

Weiterer Teil der Schaltung ist ein ZLDO-Festspannungsregler. Der Regler hat einen sehr geringen Voltage Drop, eine Übertemperatursicherung und kann mittels einer Steuerleitung deaktiviert werden. Sein Ausgang kann hier auch zur Versorgung externer Schaltungen verwendet werden (bis ca. 260mA Last).
Der MAX3221 ist Teil der RS232-Schnittstelle. Dieser IC hat den Vorteil, dass er neben dem manuellen Deaktivieren über die Signale FORCEON und FORCEOFF auch einen AutoShutdown besitzt (nach ca. 30µs Inaktivität) und man den Status der Schnittstelle zusätzlich abfragen kann (Gegenstelle, z.B. ein Computer, aktiv?)

Zusätzlich gibt es noch 3 LEDs zur Anzeige „Power-Good“, „Empfange Daten“ und „Sende Daten“.

Tab. 1: Stückliste

Bezeichnung	Wert	Bezeichnung	Wert
IC1	RFM12	R2,R3,R4	470Ω
IC2	ATmega32	R5	10kΩ
IC3*	MAX3221	R6,R7	100kΩ
IC4*	ZLDO500/ZLDO330	C1,C2	32pF
Q1	8MHz	C9*	10pF
LED1/LED2/LED3	Chip-LEDs R/G/B	C6,C7,C12	100nF
JP1	3*2,54mm Leiste	C3,C4,C5*	470nF
CON1,CON2	13*,254mm Leiste	C11*	100nF
SMA	SMA Printbuchse	C8*	47µF
R1	47kΩ	C10*	22µF

* nicht Teil der Minimalbeschaltung

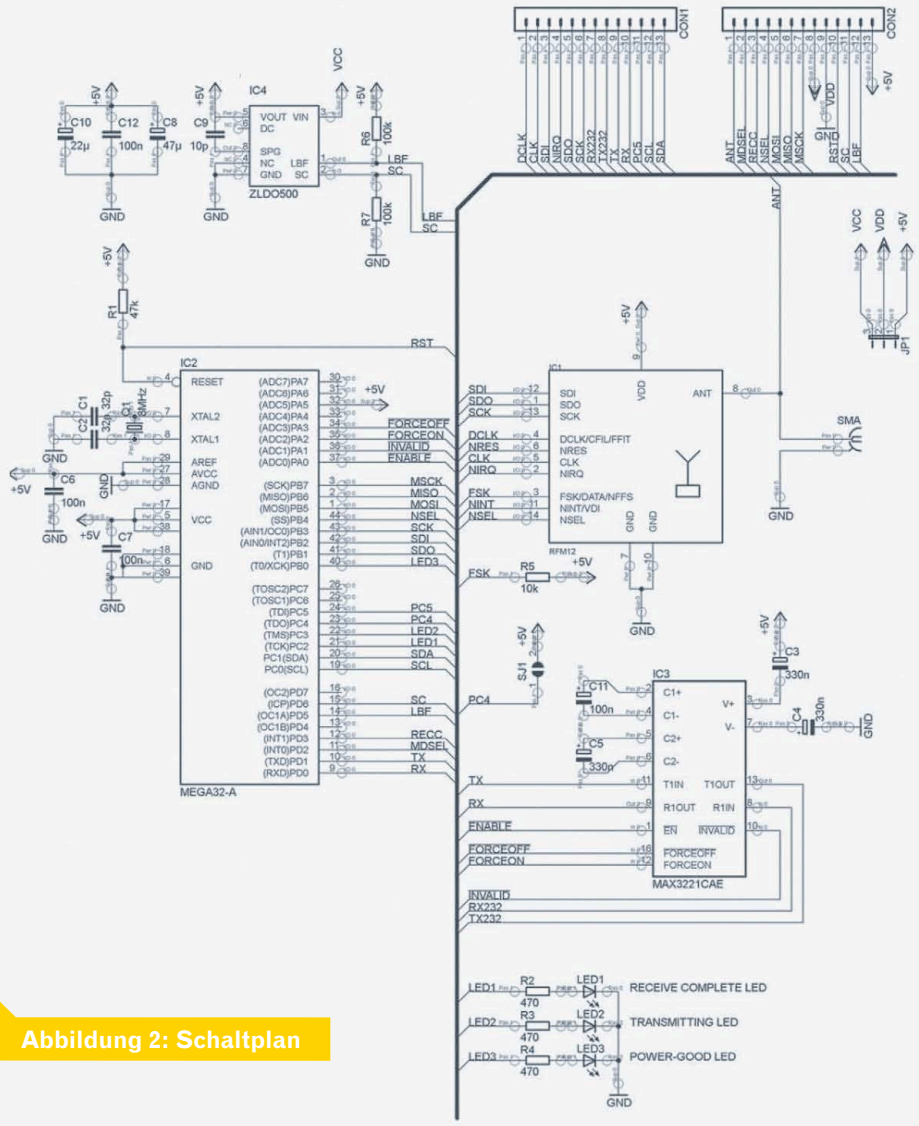


Abbildung 2: Schaltplan

Schnittstelle zur Außenwelt

Das Funkboard wird mit 2 Stiftleisten einfach in die Zielschaltung gesteckt. Über diese Stiftleisten findet auch die Kommunikation mit dem Board statt.

Die Anschlüsse können grob in 3 Gruppen eingeteilt werden.
1. Interfaces: UART, RS232 (zur Ansteuerung), I2C, SPI (für spätere Erweiterungen) sowie weiterer Steuerleitungen.
2. Debugging: Anschlüsse zum RFM12 zum Debuggen des Moduls.
3. Sonstiges: Spannungsüberwachung, Reset, Antenne, Statussignal, GPIO etc.

Für den Anschluss einer Antenne ist ein SMA-Connector für Printmontage vorgesehen. Der Antennenanschluss kann aber auch über einen Pin der Stiftleiste herausgeführt werden.

Die Software

Die Software für den AVR wurde in C unter der Entwicklungsumgebung „AVRStudio“ geschrieben. Der Code ist zu groß für einen ATmega8, passt jedoch in einen ATmega16 oder ATmega32.

Das Manko vieler OpenSource-Lösungen ist, ohne diese kritisieren zu wollen, dass der Code oftmals sehr minimalistisch gehalten oder nicht ausreichend kommentiert und strukturiert ist. Will man diese Codes verwenden, hat man zwar einen guten Startpunkt, die eigentliche (Durst-)Strecke muss man jedoch selbst bewältigen. Gerade im Hinblick darauf, dass das Funksystem oft selbst nur „Mittel zum Zweck“ sein soll, ist dies dann ärgerlich.
Der hier vorgestellte Sourcecode soll es etwas besser machen: Es wurde großen Wert darauf gelegt diesen übersichtlich, strukturiert und möglichst umfangreich zu gestalten. Der Code bietet bereits viele Möglichkeiten und kann ohne großen Aufwand von jedem selbst ergänzt werden!

Funktionelle Struktur

Die Strukturierung des Codes gliedert sich in 2 Teile, einen für das Empfangen und Senden (sog. Transceivemodus) und einem für das

Konfigurieren der Funkverbindung (sog. Konfigurationsmodus). Die Ansteuerung beider Modis erfolgt über die UART und eine Steuerleitung, deren Pegel den ausgewählten Modus bestimmt.

Sende- und Empfangsbetrieb

Überlässt man das Board „sich selbst" springt es automatisch in den Empfangsbetrieb und wartet bis das RFM12 Daten empfängt (Polling). Beim Senden werden die Daten paketweise übertragen. Danach wird wieder in den Empfangsbetrieb gesprungen. Jedes Board hat eine änderbare Identifikationsnummer (ID). Beim Senden von Daten kann dadurch der Empfänger ausgewählt werden. Außerdem ist es möglich ein Handshaking des Empfängers anzufordern. Alles Weitere übernimmt das Board, ab hier muss man sich um nichts mehr kümmern.

Konfigurationsbetrieb

Eine Funkübertragung ist nur so gut wie sein am schlechtesten eingestellter Parameter.

So oder so ähnlich könnte man die Aufgabe des Konfigurationsbetriebs charakterisieren. Das RFM12 stellt schon von Haus aus viele Parameter zum (Fein-)Tuning bereit, zusammen mit dem AVR werden es nochmals einige mehr. Alle diese Parameter können im Konfigurationsmodus manuell eingestellt werden. Von RSSI und LNA Gain, über die Identifikations- und Kanalnummer bis hin zu Softwareresets und Sleepmodes.
Damit man nicht bei jedem Neustart das Spiel von vorne beginnen muss, speichert die Software auf Wunsch die Parameter ab und lädt sie automatisch.beim Startup.
Zu guter Letzt ist es auch möglich alle Parameter auszulesen. Besonders im Hinblick auf das später beschriebene PC Terminalprogramm bietet dies interessante Möglichkeiten.

Der saure Apfel

Irgendwo gibt es immer einen Haken. Hier sind es die Hardwareressourcen des.AVRs. Möchte man dem AVR auch andere Aufgaben zuteilen und ihn nicht nur der Funkerei überlassen, muss man Einschränkungen hinnehmen.
Bei der Programmierung wurde buchstäblich aus dem Vollen geschöpft und reger Gebrauch von Timern und Interrupts gemacht. Zwar lassen sich kleine Subroutines für andere Aufgaben recht einfach hinzufügen, allerdings muss beim Programmieren auf diese Problemstellung Rücksicht genommen werden.

Und sonst...?

Ein Schaltplan und der dazugehörige Sourcecode sind zwar schön, zum Abrunden des Projekts fehlen aber noch einige Dinge…

PC Terminalprogramm

Teil des Projekts ist ein in C# geschriebenes PC-Programm, das über die serielle Schnittstelle auf das Funkboard zugreifen kann. Hierfür ist lediglich ein kleiner Adapter mit Pegelwandler-IC nötig. Die Software ist dann in der Lage die gesetzten Konfigurationen auszulesen und umzuändern. Außerdem kann auch der Funkbetrieb getestet werden. Die Software kann auch als Basis für andere Projekte dienen und bietet damit eine Starthilfe.

Dokumentation und Skripte

Abgerundet wird das Projekt durch eine 66 seitige Dokumentation. Hinzu kommen Skripte zum schnelleren Einstieg sowie eine Anleitung zum Antennenbau.

Kosten

Die Kosten des Funkboards können nicht pauschal angegeben werden, da das Board modular aufgebaut und (noch) nicht käuflich zu erwerben ist.
Die Platinen dürften mit rund 3 bis 6 € zu Buche schlagen. RFM12, ATmega32 sowie Quarz und einiger R's und C's als Minimalbeschaltung können zusammen mit grob 7-10 € angesetzt werden. Nach oben hin liegt es an jedem selbst.
Bei kompletter Bestückung können nochmals 5-7 € hinzuaddiert werden. Macht summa summarum im Mittel etwa 15-20 € für eine ausreichend bestückte Platine.

Fazit

Das hier vorgestellte Projekt ist für jeden interessant, der plant ein kleines Funksystem einzusetzten, jedoch nicht das Rad neu erfinden oder sich hohen Kosten aussetzen möchte.
So kann man ein Ultra-Low-Cost Funkmodul mit einer umfangreichen Hard- und Software-Umgebung kombinieren und hat mit geringem Kosten- und Zeitaufwand eine flexible und leistungsstarke Funklösung. Es soll aber auch als Inspiration für eigene Projekte dienen.

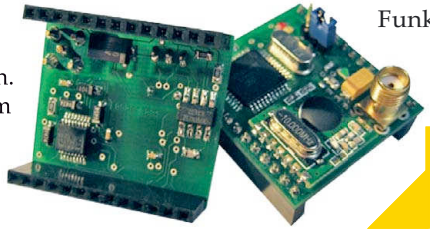


Abb. 3: Funkboard Vorder- und Rückseite

Embedded Linux - Stoppuhr für Interrupts

an der Hochschule Augsburg

Hubert Högl <Hubert.Hoegl@hs-augsburg.de >

Im vergangenen Oktober hat zum ersten Mal mein neuer Kurs **Embedded Linux [2]** an der Hochschule Augsburg stattgefunden. Etwa 20 Studentinnen und Studenten haben sich dafür interessiert. Mir war es sehr wichtig, dass jeder Teilnehmer von Anfang an einen passenden preiswerten Rechner zum Üben hatte, im Idealfall sogar als sein „Eigentum". Nach etlichem hin und her fiel die Wahl auf das **Network Gateway** von Atmel (NGW100), weil es im Hochschulprogramm des Herstellers nur knapp 50 Euro kostet, es zum Basteln mit einfachen Mitteln gut geeignet ist, und vor allem weil seine „Linux Heimat" `www.avr32linux.org` aktuell und vorbildlich aufgeräumt ist, so dass man auch als (relativer) Anfänger sofort loslegen kann. Man findet dort auch ein Wiki mit manchen sehr gelungenen Anleitungen, das uns gerade bei den hardwarenahen Versuchen weitergeholfen hat.

Als Alternative war auch noch der Grasshopper von Benedikt Sauter eine Überlegung wert, allerdings ist das Board teurer und es unterscheidet sich bei der Hardware an manchen Stellen ein wenig vom NGW100. Ausserdem ist die Entwicklungsumgebung für den Grasshopper leider bisher noch abgekoppelt von der Hauptentwicklung auf avr32linux. Für einen Anfänger ist der Zugang zu diesem Board also deutlich schwerer als beim NGW100.

Die Unterstützung der Kursteilnehmer durch Skripte, Anleitungen und sonstiges Material war von meiner Seite noch etwas „experimentell", wie man bei wackeliger Software sagen würde. Das Skript enthält noch viele Lücken, insgesamt gibt es noch viel an Schreib- und Organisationsarbeit zu leisten, allerdings gibt es einige sehr gute Bücher, in denen die gesamte Thematik erklärt wird. Ausserdem sammle ich seit Jahren freie Texte zu diesem Thema auf einer CDROM, die ich den Studenten zur Verfügung stelle.

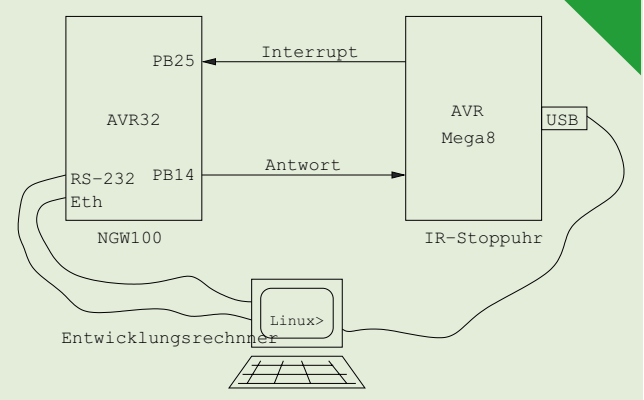
Viel Zeit hat am Anfang die Besprechung der Hardwareeigenschaften des Boards, des Bootvorganges (was passiert Schritt für Schritt beim Booten?) und der Cross-Entwicklungsumgebung (buildroot) benötigt. Das Ziel war, das NGW an einem Entwicklungsrechner, meist das Notebook des Studenten, so zu betreiben, dass sowohl der Kernel als auch das Root-Filesystem über das Netzwerk geholt bzw. gemountet werden. Mit diesem Aufbau wurden dann die ersten selbstgeschriebenen Programme getestet, z.B. „Hello World" über die serielle Schnittstelle. Je nach Vorwissen haben sich manche Studenten dann auch bereits an die manchmal gar nicht so einfache Kompilierung von grösseren Programmen gemacht, die es bisher noch nicht für AVR32 gab. Unter anderem wurde ein Jabber-Client und Software aus dem Audio-Bereich für den AVR32 kompiliert **(siehe Studienarbeiten unter [2])**. Den Python 2.6.1 Interpreter für AVR32 zu kompilieren hatte ich mir vorgenommen, da ich mit Python auf Embedded Linux ein paar Experimente machen möchte.

Das grosse Ziel bei Linux auf Mikrocontroller-Boards ist aber natürlich, den Rechner um externe Hardware zu erweitern bzw. das „Interfacen" an externe Elektronik. So waren die sehr wichtigen schmalen Schnittstellen wie SPI und I2C bei den Übungen Pflicht. Fast immer werden in der Praxis darüber viele Peripheriebaugruppen angesteuert wie Displays, Tastaturen, LEDs, AD- und DA-Wandler und vieles weitere.

Eine Stoppuhr für Interrupts

Bei einem anderen ganz einfachen aber trotzdem lehrreichen Experiment in unserem Kurs wurde ein Schalter und eine Leuchtdiode (LED) an I/O Pins des AVR32 angeschlossen. Die Absicht war, durch Drücken des Tasters einen Interrupt beim AVR32 zu erzeugen, auf diesen Interrupt in geordneter Weise in einem „Handler" im Gerätetreiber unter Linux zu reagieren und als Folge davon die LED ein- und auszuschalten. Aus diesem Experiment entwickelte sich schnell die Frage, wie lange es in etwa dauert, bis ein Interrupt vom riesigen Linux-Kern beantwortet wird. Diese Zeitspanne wird auch Latenzzeit genannt. Bei kleineren 8-Bit Mikrocontrollern ohne Betriebssystem wussten wir, dass dieser Vorgang ungefähr ein paar Mikrosekunden dauern würden.

Abb.1: Anwendung der Interrupt-Stoppuhr



Um die Frage zu beantworten bauten wir eine „Interrupt-Stoppuhr" die wie **Abbildung 1** angeschlossen wird.

Die Stoppuhr entspricht genau dem Aufbau mit Taster und LED, nur dass jetzt ein kleiner 8-Bit AVR Mikrocontroller die Signale erzeugt bzw. entgegennimmt. Die Zeit zwischen dem Interrupt und der Antwort wird einfach in einem schnell laufenden Timer gestoppt. Die Koordination der Messung soll dabei der PC übernehmen, am besten über den USB Bus. Bei der Auswahl eines geeigneten kleinen AVR Moduls, das an USB angeschlossen werden kann, dachten wir zunächst an **USBprog [5]**. Es geht aber letztendlich noch einfacher und preiswerter, wenn man mit USB ***low-speed*** (maximal 1.5 MBit/sec) zufrieden ist. Dann kann man die zwei USB Datenleitungen direkt an den AVR schalten und spart sich die USB Bridge USBN9604, die beim USBprog für full-speed (12 MBit/sec) nötig ist. Eine hervorragende low-speed USB Firmware für AVR entstand beim **AVRUSB** Projekt der Firma **Objective Development [6]**. Diese war die Grundlage für unsere Stoppuhr.

In der nächsten **Abbildung 2** sieht man die konkrete Beschaltung. UART, SPI und I2C braucht man zunächst nicht.

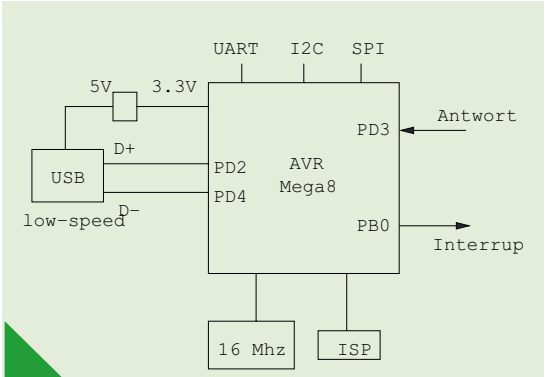


Abb.2: Beschaltung des AVR Mega8

Die Firmware haben wir im wesentlichen nur um einen Timer erweitern müssen, der in 500 nsec Schritten zählt, bis die Antwort auf den Interrupt beim AVR ankommt. Da der Zähler 16-Bit breit ist, kann man damit maximal etwa 32 msec messen, was mehr als ausreichend sein dürfte. Falls man längere Zeiten messen möchte, könnte man die Zeitschritte auf 4 oder 16 usec verlängern.

Wenn man den AVR über USB an den PC steckt, dann wird er über die Firmware im AVR als „Human Interface Device" (HID) erkannt. Treiber für diese USB Geräteklasse sind in jedem Betriebssystem enthalten. Unter Linux (auch Windows) gibt es die `libhid`, mit der man auf HID-Geräte zugreifen kann. Diese baut wiederum auf der `libusb` auf. Beim AVRUSB Projekt ist ein in C geschriebenes Kommandozeilenwerkzeug dabei, das die Kommunikation über HID mit diesen Bibliotheken demonstriert (`hidtool`). Da es für beide Bibliotheken „Wrapper" für Python gibt, war es leicht, das hidtool in C durch eine viel kürzere Variante in der Sprache Python zu ersetzen. Zur Kommunikation über USB verwendet man die Funktion `hid_set_feature_report()` der libhid, bei der einfach ein Array aus 128 Bytes zwischen PC und AVR ausgetauscht werden. Die Interpretation von ein paar Bytes am Anfang des Arrays bestimmt in unserem Fall die Funktion des AVR Controllers. Bei der Stoppuhr wird z.B. über einen Feature Report die Messung gestartet und über einen anderen Report das Messergebnis geholt. Meinen Probeaufbau sieht manin **Abbildung 3**.

Die ersten Messungen ergaben übrigens, dass die Interrupt-Antwortzeit bei etwa 40 bis 50 usec liegt. Dabei muss man jedoch beachten, dass die Zeit abhängt von der Rechenlast, die der AVR32 gerade zu bewältigen hat. Bei unserer Messung war das Linux nach dem Booten im Leerlauf.

Die Details der Stoppuhr findet man unter http://www.hs-augsburg.de/~hhoegl/elinux/misc/ir-stoppuhr/

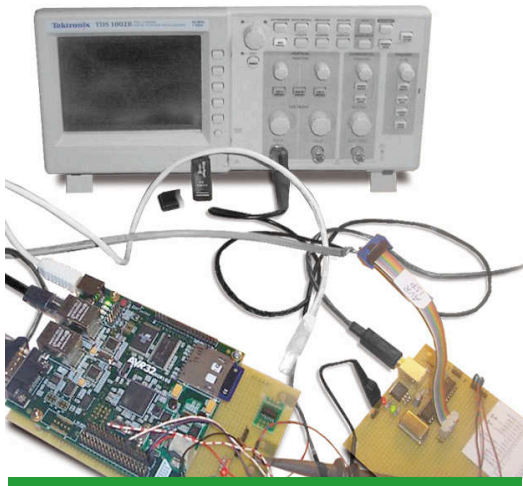


Abb.3 Versuchsaufbau der Interrupt Stoppuhr

Verbesserungsvorschläge

Wenn man sich eine professionelle Interrupt-Stoppuhr ansieht wie zum Beispiel die **Latency-Box** des **OSADL [3]**, dann sieht man schnell, was man besser machen könnte:

Man sollte möglichst viele Messungen machen und eine Häufigkeitsverteilung erstellen. Bis zu einer Milliarde Messungen werden vorgeschlagen. Das ist natürlich auch ein Zeitproblem, bei uns kommt noch dazu der Overhead durch die USB Kommunikation für jede Messung, was derart viele Messungen unmöglich macht. Zumindest könnte der AVR von sich aus mehrere Messungen erledigen, die im sehr kleinen RAM Speicher des AVR zwischengespeichert werden und dann im Stück auf den PC übertragen werden.

Wenn man den „1 Euro" Aufwand unserer Stoppuhr mit der Latency-Box (PowerPC mit 600 MHz und 64MB Speicher) vergleicht, dann schneidet erstere aber gar nicht so schlecht ab.

Eine andere Verbesserungsmöglichkeit kommt von den bisher ungenutzten Schnittstellen UART, SPI und I2C. Da wir auf dem NGW die entsprechenden Schnittstellen durch geeignete Treiber in Betrieb nehmen, ist es sehr praktisch, wenn man zum Test auch das jeweilige Gegenstück der Schnittstelle hat. Die Stoppuhr kann also sehr einfach um einen einfachen ***Schnittstellentester*** erweitert werden. In einer ganz einfache Variante, die wir vor kurzem realisiert haben, wird einfach jeder Schnittstelle eine Zählervariable zugeordnet. Pro korrekt empfangenem Zeichen wird die zugehörige Variable um eins erhöht. Diese Zählerwerte kann man über die USB Schnittstelle abfragen.

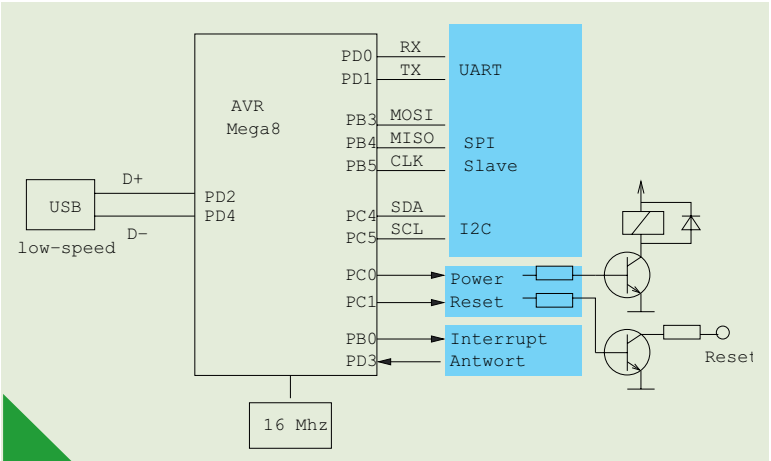


Abb.4: Der Mega8 AVR als Schnittstellen-Tester, Power/Reset-Schalter und Interrupt-Stoppuhr.

Es gibt noch eine weitere zusätzliche Verwendung für diese kleine Schaltung. Manchmal ist es praktisch, das NGW an einem Server-Rechner zu betreiben und entfernt mit dem Board über eine ssh-Verbindung zu arbeiten. Allerdings kann sich das Board so „aufhängen", dass nur noch das Drücken auf den Reset-Taster hilft. Mit einem zusätzlichen Pin am Mega8 AVR kann man einen Transistor ansteuern, der den Reset-Knopf auf dem NGW fernsteuert. Auch die Stromversorgung des NGW lässt sich mit einem Relais ein- und ausschalten, das über einen Transistor an einem weiteren Pin des Mega8 angeschlossen ist. Die PC Software zur Kommunikation mit dem AVR muss lediglich um ein paar Kommandozeilenoptionen erweitert werden.

Wie geht es weiter?

Schön wäre es, wenn wir in unserem Kurs auch mit Echtzeit-Erweiterungen für Linux experimentieren könnten. Es gibt zum Beispiel den **Realtime Preemption Patch** oder die Erweiterungen **RTAI** bzw. **Xenomai** für ganz strenge Anforderungen bei der Echtzeit. Die Entwicklung konzentriert sich dabei aber noch auf die seit langem etablierten Prozessoren wie x86, PowerPC und ARM. Vom **OSADL [3]** sagte mir aber jemand vor kurzem, dass man dort den RT-Preemption Patch für den AVR32 bald portieren möchte.

Der gute Stand bei den Echtzeit-Möglichkeiten und bei den Peripheriemodulen lässt einen natürlich auch immer ein wenig auf Platinen mit ARM Controller schielen. Zur Zeit arbeite ich an einer Schaltung mit einem AT91SAM9260 im QFP208 Gehäuse von Atmel, die man zur Not auch selber bestücken könnte. Es wird auch noch der USB Controller FT2232 drauf sein, mit dem man direkt eine serielle Schnittstelle und den JTAG Port des ARM ansteuern kann, so dass man sofort nach dem Anstecken an USB mit der Arbeit beginnen kann. Als Debugger kann man den freien **OpenOCD [4]** verwenden, der mittlerweile bei jedem Linux als Standardpaket dabei ist. Leider gibt es für den AVR32 keinen freien Debugger. Man ist hier an den **JTAGICE mkII** gebunden, der immerhin gut 200 Euro kostet. Das ist die zweite bittere Pille (neben der fehlenden Echtzeit-Unterstützung), die man beim AVR32 schlucken muss. Ein dritter, oft beklagter Mangel ist die fehlende USB Host Schnittstelle beim AVR32AP7000. Der neue AP7200 hätte zwar eine gehabt, allerdings wurde die Markteinführung dieses Bausteines vor kurzem auf unbestimmte Zeit verzögert.

Den OpenOCD hat übrigens Dominic Rath in einer Diplomarbeit an der Hochschule Augsburg vor ein paar Jahren in die Welt gesetzt. Die Diplomarbeit gibt es auf meiner Homepage zum freien Download. Mittlerweile wird das Programm von einer Entwicklergemeinde weitergeschrieben.

Der nächste Embedded Linux Kurs beginnt im Sommersemester 2009 (Mitte März). Ich freue mich jederzeit über Anregungen per E-mail.

Links

[1] Mein Startseite an der Hochschule http://www.hs-augsburg.de/~hhoegl
[2] Der Embedded Linux Kurs. Hier findet man auch die ersten entstandenen Ausarbeitungen.
http://www.hs-augsburg.de/~hhoegl/elinux/elinux.html
[3] Open Source Automation Development Lab (OSADL) http://www.osadl.org
[4] OpenOCD Debugger http://openocd.berlios.de
[5] USBprog Adapter http://www.embedded-projects.net
[6] AVRUSB Projekt der Objective Development Software GmbH http://www.obdev.at/products/avrusb/index.html

USB IO Expander

Michael Wittmann <www.mikrocontroller.net>

Einleitung

Oft wäre es praktisch, einige digitale bzw analoge Signale möglichst schnell und einfach grafisch am PC darzustellen oder per digitalem Ausgang verschiedene externe Baugruppen anzusteuern. Da es relativ umständlich ist, dafür jedes mal einen Mikrocontroller zu programmieren, eine Datenübertragung mit dem Rechner herzustellen und das ganze dann auch noch zu visualisieren, ist dieses Projekt entstanden.
Der USB IO Expander besitzt jeweils 16 digitale Ein- und Ausgänge, sowie 4 analoge Eingänge. Zusätzlich sind 4 Relais integriert, sowie SPI und IIC Schnittstellen. Das ganze lässt sich über eine einfach zu bedienende Software von jedem PC aus steuern. Die Anbindung an den Rechner erfolgt über eine galvanisch getrennte USB Schnittstelle, über die das Gerät auch mit Strom versorgt wird.

Anforderungen

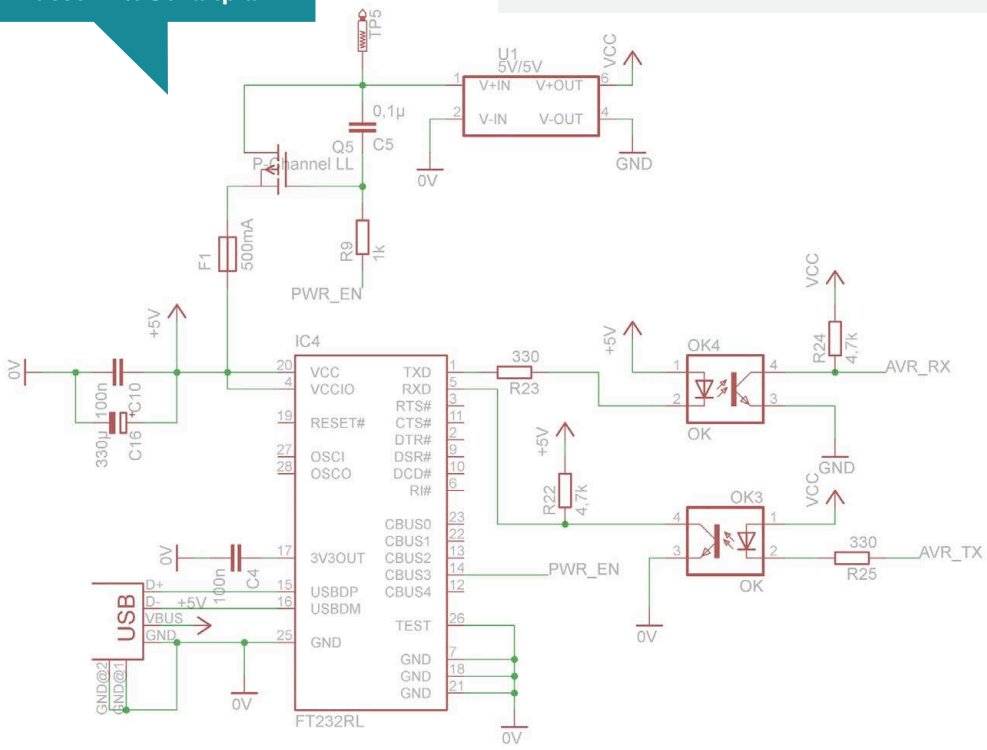
Mit dem USB IO Expander soll kein Ersatz für ein vollwertiges Oszilloskop oder einen Logikanalyser geschaffen werden. Dies wäre auf Grund der begrenzten Rechenleistung des verwendeten AVR sowie aus Kostengründen auch gar nicht zufriedenstellend möglich. Aus diesem Grund habe ich mich bewusst auf den niederfrequenten Bereich (einzelne Funktionen bis 50kHz) beschränkt, da alles andere nur Kompromisse gewesen wären. Ein wesentlicher Punkt war jedoch ein vollständige galvanische Trennung vom PC. Dies vereinfacht die Arbeit mit dem Gerät doch sehr, da man nicht aufpassen muss, über andere Geräte Kurzschlüsse über den Schutzleiter herzustellen (Die Masse eines PCs liegt auf Erdpotential, so dass zum Beispiel der gleichzeitige Einsatz eines Programmieradapters kritisch wäre).

Features

- USB Schnittstelle
- 16 digitale Eingänge
- 16 digitale Ausgänge
- 4 unabhängige Analoge Eingänge
- 4 Relais
- SPI Schnittstelle
- IIC Schnittstelle (für zukünftige Erweiterungen)
- Stromversorgung über USB
- galvanische Trennung zum Rechner
- einfach zu bedienende Steuersoftware am PC
- Ausgabe von Mustern an den digitalen Ausgängen möglich (Patterngenerator, bis maximal 50kHz)
- eingeschränkt als Logic Analyser verwendbar (auch bis 50 kHz, gleichzeitigmit dem Pattern-Generator verwendbar)

Hardware

Abb.1: Ausschnitt Schaltplan



Schnittstellen - USB

Die USB Schnittstelle hat 2 Funktionen. Einmal geschieht die Stromversorgung über diese, zum anderen wird sie auch für die Datenübertragung zum PC genutzt, da die meisten neuen Computer keine seriellen Schnittstellen mehr haben. Das Gerät meldet sich als USB-seriell Konverter am PC an und fordert den maximal möglichen Strom von 500mA an. Deshalb kann man den USB IO Expander nur an aktiven USB Hubs bzw direkt am Rechner verwenden. Passive Hubs werden nicht unterstützt. Die 500mA werden jedoch höchstens benötigt, falls alle 4 Relais eingeschaltet sind und alle 16 digitalen Ausgänge mit den maximal erlaubten 20mA belastet werden. Um eine komplette galvanische Trennung vom PC zu erreichen, wurden die RX/TX Leitungen der UART mittels Optokopplern isoliert und die 5V USB Spannung mit einem galvanisch isolierten DC/DC Wandler entkoppelt. Zusätzlich ist ein FET vorhanden, der – wie in der USB Spezifikation gefordert – die Versorgungsspannung des Gerätes abschaltet, falls der USB Controller im PC das verlangt. Es wird ein FT232R USB-seriell Wandlerchip verwendet. Es wäre zwar auch möglich, eine USB Schnittstelle in Software im AVR zu integrieren, hier sprechen jedoch mehrere Dinge dagegen: Zum einen ist es so nur sehr schwer möglich die serielle Schnittstelle galvanisch zu trennen, zum anderen ist der Controller damit ziemlich ausgelastet, so dass weniger Rechenleistung bzw Speicher für seine eigentlichen Aufgaben zu Verfügung steht. Da man den FT232 für unter 4 Euro bekommt, war mir das den zusätzlichen Aufwand nicht Wert. Zudem sollte der FT232 Treiber auch

unter Linux funktionieren (jedoch noch nicht getestet).

Das Bauteil ganz oben in **Abbildung 1** ist der DC/DC Wandler, der die Versorgungsspannungen galvansich trennt. F1 ist eine Polyfuse, die im Falle eines Kurzschlusses den USB Port trennt. Auf Grund des begrenzten Kurzschlussstroms des DC/DC Wandler kann man sich die Sicherung evtl. sparen, was jedoch nicht unbedingt zu empfehlen ist. Die beiden Optokoppler auf der rechten Seite sorgen schließlich noch für die galvanische Trennung der RX/TX Leitungen, so dass der USB Port komplett vom restlichen Gerät entkoppelt ist.

Digitale Ein-/Ausgänge

Es sind jeweils 16 Ein- und Ausgänge vorhanden, die für 5V TTL Pegel ausgelegt sind. Es wurde großer Wert darauf gelegt, das Gerät möglichst robust zu gestalten. Deshalb vertragen die Eingänge Spannung bis zu 12V. Die Ausgänge sind kurzschlussfest. Sie sind darauf ausgelegt, pro Ausgang bis zu 20mA zu liefern. So kann man LEDs ohne zusätzliche Transistoren anschließen. Die digitalen Ein-/Ausgänge sind zusätzlich durch integrierte gesockelte Treiber geschützt, so dass man im Falle eines Defekts nur diesen austauschen muss. Die Ansteuerung der insgesamt 32 Leitungen ist über 4 Schieberegister vom Typ 4021 realisiert. Als Treiberbausteine kommen 4 ICs vom Typ 74AC245 zum Einsatz (der AC Typ wurde wegen der höheren Strombelastbarkeit gewählt, prinzipiell kann aber jeder kompatible Chip verwendet werden – dann aber evtl. mit Einschränkungen beim maximal entnehmbaren Strom).

Analoge Eingänge

Zusätzlich sind 4 analoge Eingänge verfügbar, die Spannungen im Bereich von 0-5V mit einer Auflösung von 8Bit messen können. Auch hier sind Spannungen bis 12V ohne Beschädigung der Hardware möglich, auch wenn sich der messbare Bereich auf 5V beschränkt. Eventuell muss man hier einen externen Spannungsteiler zuschalten. Zur A2D Wandlung wird der integriert Wandler des AVR verwendet.

Relais

Die 4 eingebauten Halbleiterrelais können Ströme bis zu einem Ampere schalten. Sie sind durch optionale interne 5x20mm Sicherungen abgesichert, so dass auch hier eine Beschädigung der Hardware fast ausgeschlossen ist.

SPI

Weiterhin befindet sich eine SPI Schnittstelle im Gerät. Diese kann frei konfiguriert werden und sollte so für jede Anwendung verwendbar sein, die 8Bit Daten erwartet. Auch die SPI Schnittstelle ist tolerant gegenüber 12V Spannungen, arbeitet selber jedoch auch mit 5V TTL Pegeln. Es wird die im AVR integrierte SPI Schnittstelle verwendet.

IIC

Für das IIC Interface wird die im AVR integrierte Schnittstelle verwendet. Außerdem ist diese ebenfalls mit Schutzdioden ausgestattet, und überlebt so Spannungen bis 12V. Im momentanen Entwicklungsstand ist die IIC Schnittstelle jedoch noch nicht in die Software integriert, da sie nur für eventuelle Erweiterungen dient (z.B. Hinzufügen eines Temperatursensors).

Controller

Als Controller ist ein Mikrocontroller vom Typ AtMega8 verbaut. Dieser ist mit 14,746 Mhz getaktet, um eine Problemlose UART Übertragung zu gewährleisten. Außerdem bietet er genug Speicherreserven um evtl noch weitere Softwarekomponenten hinzuzufügen.

Platine

Die Platine besteht eigentlich aus zwei Teilen, und zwar der eigentlichen Controller-Platine und einer weiteren optionalen, die nur die Sicherungen für die Relais aufnimmt und zur Befestigung einer Schraubklemme am Gehäuse dient. Die doppelseitige Controller-Platine hat eine Größe von ca 7x10 cm und passt genau in ein „TEKO WALL 3" Gehäuse von Reichelt. Sie ist auf Abbildung 2 zu sehen.

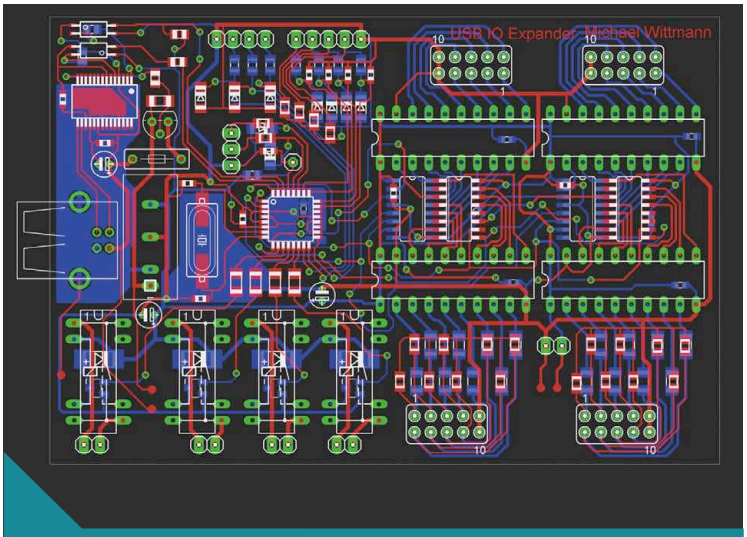


Abb.2: doppelseitige Controller-Platine

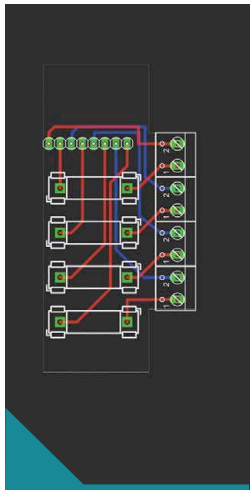


Abb.3: Relais-Out-Platine

Die Relais-Out Platine ist ebenfalls doppelseitig und genau so groß, dass darauf vier 5x20mm Sicherungen Platz haben. Sie hat auf einer Seite eine Ausbuchtung. Diese sollte durch einen Schlitz im Gehäuse gesteckt werden, um dort eine Schraubklemme aufzulöten. Diese ist dann von außen erreichbar. Die Relais-Out Platine ist optional und kann weggelassen werden, falls man die Sicherungen für die Relais nicht benötigt bzw die Anschlüsse der Relais anderweitig aus dem Gehäuse führen möchte. Die Relais-Out-Platine ist in Abbildung 3 zu sehen.

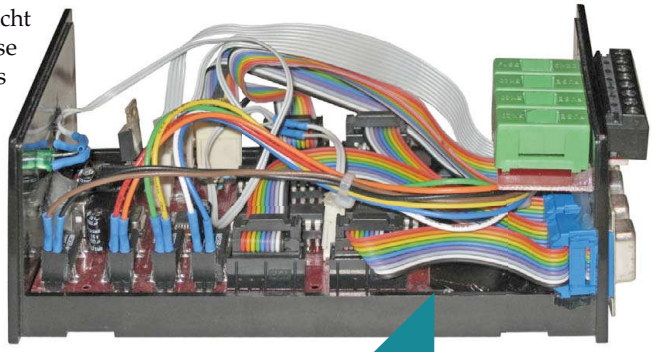


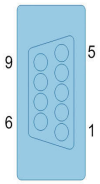
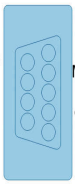
Abb.4: Gehäuse offen, Innenverkabelung

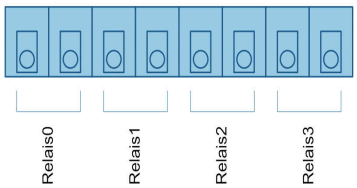
Um das ganze möglichst kompakt zu halten, konnte auf SMD Bauteile nicht verzichtet werden. Alle Teile mit Ausnahme der Treiberstufen sind SMD-Bauteile. Die Treiber sind als DIP Gehäuse ausgeführt und gesockelt. So kann man im Falle eines Defektes diese ohne Lötarbeiten austauschen. Alle extern zugänglichen Signale wurden auf der Platine herausgeführt. Die digitalen Ein-/Ausgänge sind als 2x5 polige Stiftleiste ausgeführt. Somit kann diese z.B. mit einem 10poligen Flachbandkabel herausführen. Alle anderen Signale (A2D, IIC, SPI) wurden mittels einfachen Stiftleisten verfügbar gemacht. Zudem befinden sich mehrere Testpunkte auf der Platine, an denen sich bei der Inbetriebnahme einfach zugänglich alle Versorgungsspannungen messen lassen. Auf Abbildung 4 kann man erkennen, wie die kleine Platine rechts zur Befestigung der Schraubklemme dient sowie die Verkabelung mit Flachbandkabeln.

Gehäuse

Als Gehäuse kann z.B. wie bereits vorgeschlagen ein TEKO WALL 3 zum Einsatz kommen. Es müssen jedoch z.B. mit einem Dremel Öffnungen für die USB Buchse sowie für die restlichen Ein- und Ausgänge hinein gefräst werden. Eine mögliche Realisierung (die ich gewählt habe) besteht z.B. darin, alle Signale über D-Sub Buchsen herauszuführen. Auf der schmalen Seite des verwendeten Teko Gehäuse, ist gerade genug Platz um nebeneinander sechs D-Sub Buchsen einzubauen. Wenn man welche nimmt, in die man direkt Flachbandkabel einpressen lassen, kann man so ohne großen Aufwand alle Signale nach außen führen. Falls die Variante mit den D-Sub Buchsen gewählt wird, kann z.B. die Vorlage in Abbildung 5 als Beschriftung für das Gehäuse verwendet werden.

Abb.5: Gehäusebeschriftung

	Pins	IN0-IN7	IN8-IN15	OUT0-OUT7	OUT8-OUT15	A2D	TWI/I²C
D-SUB		9 IN0 8 IN1 7 IN2 6 IN3 5 IN4 4 IN5 3 IN6 2 IN7 1 GND	9 IN8 8 IN9 7 IN10 6 IN11 5 IN12 4 IN13 3 IN14 2 IN15 1 GND	9 OUT0 8 OUT1 7 OUT2 6 OUT3 5 OUT4 4 OUT5 3 OUT6 2 OUT7 1 GND	9 OUT8 8 OUT9 7 OUT10 6 OUT11 5 OUT12 4 OUT13 3 OUT14 2 OUT15 1 GND	9 ADC3 8 ADC2 7 ADC1 6 ADC0 5 GND 4 GND 3 GND 2 GND 1 GND	MISO GND SCL GND  SCK MOSI SDA GND VCC



Software - Firmware im AVR

Allgemeines

Die Firmware ist komplett in C programmiert. Als Compiler kam der gcc zum Einsatz. Die Firmware ist in verschiedene Module unterteilt. Jedes dieser Module ist für eine bestimmte Funktion bestimmt. Da die meisten Module sehr simpel aufgebaut sind, soll hier nur ein grober Überblick über die Funktionsweise gegeben werden. Jede Funktion im Quellcode ist kommentiert, so dass es relativ leicht sein sollte, die Funktionalität nachzuvollziehen.

```
case CMD_NEW_COMMAND: //entprechenden Befehl einsetzen
{
        if(RingBuffer_Size(&rx_buffer) >= 5)//je nach Anzahl der Bytes, die zu einem Befehl gehören, anpassen
        {
                uint8_t data = RingBuffer_Pop(&rx_buffer); //empfangene Daten in Variable einlesen
                uint8_t data2 = 45;
                RingBuffer_Push(&tx_buffer,data2); //Daten senden
                UCSRB |= (1<<UDRIE); //Interrupt aktivieren, damit Daten gesendet werden
                cmd = CMD_NO_COMMAND; //notwendig damit der Befehl nur einmal ausgeführt wird
        }
}break;
```

Abb.6: Erweitertes switch (cmd) Statement

ShiftRegister Modul

Dieses Modul dient der Ansteuerung der Schieberegister, also dem Einlesen der digitalen Eingänge sowie dem Ausgeben von Werten auf den digitalen Ausgängen. Die Ansteuerung der Schieberegister wird komplett in Software erledigt. Ein Verwenden der Hardware-SPI-Schnittstelle wäre zwar etwas schneller, aber da in diesem Fall die Geschwindigkeit keine Rolle spielt, wurde das SPI-Interface freigehalten.

ADC Modul

Diese Modul stellt Funktion bereit, mit denen man Werte des ADC einlesen kann. Die Read-Funktion blockiert solange, bis der ADC mit dem Einlesevorgang fertig ist. Da auch hier keine hohen Sampleraten erreicht werden sollen, spielt das keine Rolle, so das ein Abfragen über Interrupts keinen Sinn macht (der AVR würde in der Zwischenzeit sowieso nur warten).

Relais Modul

Dieses sehr simple Modul dient einzig dazu, die 4 Relais anzusteuern. Da der AVR die Relais direkt über Treibertransistoren ansteuert, beschränkt sich die Funktionalität hier auf das setzten bzw löschen von Port-Pins.

SPI Modul

Das SPI-Modul kapselt den Zugriff auf die AVR-interne SPI Schnittstelle. Es erlaubt das senden von Daten sowie die Konfiguration der Schnittstelle. Auch hier blockieren die Sende-Routinen bis der Vorgang beendet ist, da keine hohen Anforderungen an die Geschwindigkeit gestellt werden.

RingBuffer Modul

Das Ringbuffer Modul steuert keine externe Peripherie, sondern stellt Funktionen zur Verfügung, um einen Ring-Puffer zu verwalten. Dies erleichtert die Kommunikation mit dem PC erheblich. Ein Ring-Puffer ist eine Struktur die nach dem FIFO-Prinzip arbeitet (First-In-First-Out).

Parser Modul

Dies ist der eigentliche Kern der Software. Es speichert die vom PC empfangenen Daten in einem Ring-Puffer und führt eventuell erhaltene Befehle aus. Auf Grund der doch etwas komplexeren Funktionsweise soll dieses Modul hier etwas genauer erläutert werden: Die Kernfunktionaliät ist in der Funktion Parser_Process implementiert. Diese wird in einer Endlosschleife aufgerufen und stellt den eigentlichen Parser dar. Zuerst wird geprüft, ob sich im Eingangspuffer ein Byte befindet, was einem Befehl entspricht. Falls dies der Fall ist, wird im folgenden switch Statement der entsprechende Befehl ausgeführt. Etwas mehr Aufwand bedürfen die Befehle, die Timer-gesteuert arbeiten (Pattern ausgeben bzw mehrere digitale Werte samplen). Dort muss gewartet werden, bis alle benötigten Daten vom PC empfangen wurden. Anschließend wird ein Timer gestartet, der zeitgesteuert die entsprechende Aktion ausführt. Während dieser Zeit wird die UART deaktiviert, um ein Überlaufen des Empfangspuffer zu verhindern, da teilweise direkt aus diesem gelesen wird.

Eigene Erweiterungen

Um dem Parser neue Befehle beizubringen, muss nur in der Funktion Parser_Process() das switch(cmd) Statement nach Schema in Abbildung 6 erweitert werden.

Compilieren

Die Software compiliert mit dem AVR-GCC Compiler (WinAVR 20080610 getestet). Für die Entwicklungsumgebung Codeblocks wird wieder eine Projektdatei mitgeliefert. Diese ist aber nicht nötig. Einfach alles in einen Ordner entpacken, und make aufrufen (ein Makefile wird mitgeliefert).

Steuersoftware auf dem PC

Allgemeines

Die Software wurde komplett in C++ implementiert und verwendet die Qt Bibliothek. Ursprünglich wurde sie für Windows geschrieben, auf Grund der Plattformunabhängigkeit der Qt Bibliothek sollte sie aber auch unter Linux compilieren, da keine Windows-spezifischen Funktionen verwendet wurden (jedoch nicht getestet). Zusätzlich wurde die FTD2XX.dll Bibliothek von FTDI verwendet, die zur Ansteuerung des verwendeten USB Chips dient. Es wurde hier auf die Verwendung eines virtuellen COM-Ports verzichtet, und stattdessen der FTD2XX Treiber verwendet, da dieser auch unter Linux zur Verfügung steht. Ansonsten hätte eine plattformunabhängige Lösung gefunden werden müssen, um COM Ports zu verwalten. Außerdem vereinfacht die gewählte Lösung die Handhabung des Gerätes, da die Software ohne Auswahl eines COM-Ports das Gerät stets findet. Hier soll nun ein grober Überblick über den Aufbau gegeben werden.

GUI

Die GUI besteht zunächst einmal nur aus einem Hauptfenster, das unten ein Info-Feld sowie Informationen zu Verbindung bereitstellt. Im oberen Bereich befindet sich ein Tab-Fenster, in das dann die verschiedenen Funktionen als eigenständige Tabs integriert werden können.

Pattern

Oben rechts befinden sich zwei Buttons, um ein Pattern zu definieren und um den eingebauten Trigger zu konfigurieren. Das Pattern wird auf Wunsch dann am Ausgang ausgegeben. Hierzu dient unten rechts der „Samplen"-Button. Mit den beiden Checkboxen kann ausgewählt werden, ob ein Pattern ausgegeben werden soll („output") bzw der Eingang gepollt werden soll („input"). Auch eine Kombination ist möglich. Falls ein Trigger definiert ist, startet die Ein-/Ausgabe erst, falls die Triggerbedinung erfüllt ist. Die „Anzahl der Samples"-Box wird nur ausgewertet, falls kein Output-Pattern definiert ist. Ansonsten wird die Anzahl der Samples an Hand des definierten Pattern bestimmt.

Digitalen Ein-/Ausgänge, Relais und A2D

Im unteren Bereich kann festgelegt werden, ob die Ein-/Ausgänge automatisch aktualisiert werden sollen (eventuell mit dem Intervall, mit dem die Eingänge abgefragt werden). Ansonsten muss jede Zustandsänderung explizit von Hand angestoßen werden.

SPI Kommunikation

Im oberen Bereich kann die SPI Schnittstelle konfiguriert werden. Das Datenfeld unten bietet verschiedene Anzeigemodi (Binär, Dezimal, Hex), die mit den Checkboxen ausgewählt werden können. Die zu sendenden Daten werden byteweise binär gewandelt und über die SPI Schnittstelle gesendet. Als Trennzeichen zwischen den Bytes dient ein Leerzeichen.

USB Kommunikation

Für die Kommunikation mit dem Gerät ist die Klasse CIOExpanderCommunicationModule zuständig. Diese besitzt mehrere Slots, mittels deren andere Programmteile mitteilen, falls Daten gesendet werden sollen. Über den Empfang von neuen Daten vom Gerät informiert die Klasse mit Hilfe von Signalen. Die Kommunikation mit dem Gerät wird in einem eigenen Thread abgearbeitet, um währenddessen die GUI nicht zu blockieren. Die Senderoutinen verwenden einen FIFO-Puffer um die Daten vor dem Senden zwischenzuspeichern. Falls ein Befehl eine Antwort erwartet, wird eine Variable vom Typ SCommand in einer Warteschlange gespeichert. Damit kann der Parser jedes empfangene Datenbyte eindeutig einem vorher gesendeten Befehl zuordnen.

Abb.9: Konfiguration der SPI-Schnittstelle

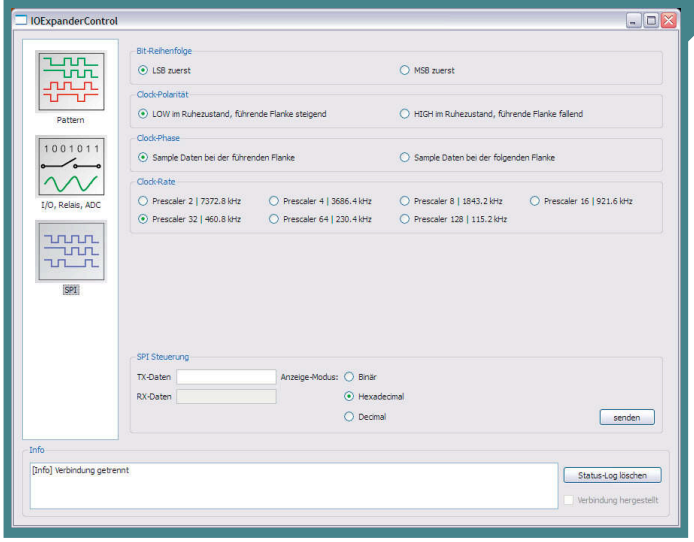


Abb.7: Pattern definieren

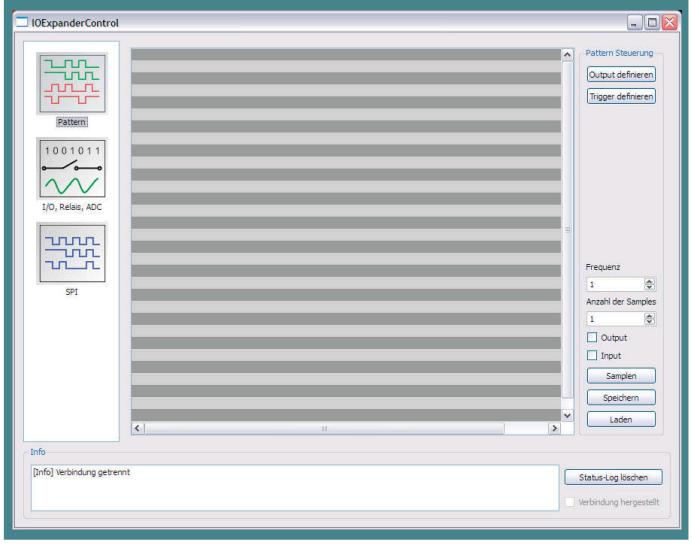
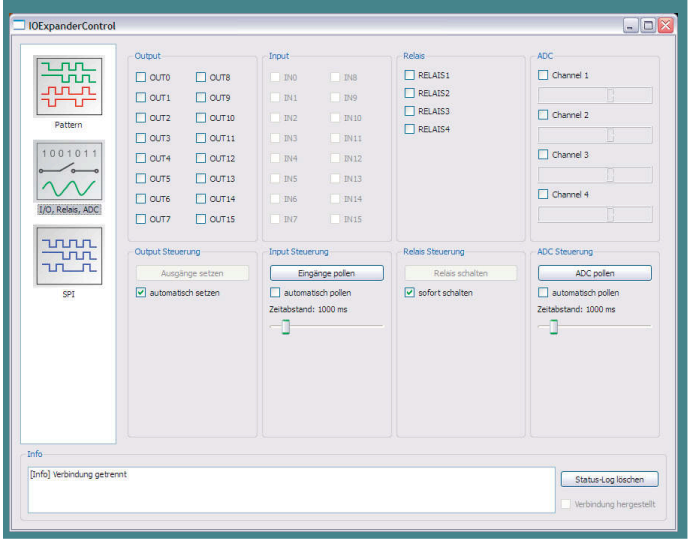


Abb.8: Automatisierung von Ein-/Ausgänge



Eigene Erweiterungen

Um der Programmoberfläche eigene Tabs hinzuzufügen, muss für jedes neue Tab eine eigene Klasse erstellt werden. Diese muss von QWidget abgeleitet sein. Ein Objekt dieser Klasse muss in CIOExpanderControlWindow instanziert werden, und in der Funktion CIOExpanderControlWindow::SetupLayout() in das Layout ein gefügt werden.

```
m_neuesWidget = new CNewTabWidget(this); //je nach verwendetem Name anpassen
m_tabs->addTab(m_neuesWidget,"Name des neuen Tabs");
```

Die Kommunikation mit CIOExpanderCommunicationModule muss über den Qt internen Signals/Slots Mechanismus abgewickelt werden. Hier zu muss die CIOExpanderCommunicationModule Klasse gegebenefalls um weitere Signale/Slots ergänzt werden. Pro Befehl muss der Enum ein weiterer Eintrag hinzugefügt werden.

```
enum CIOExpanderCommunicationModule::ECommands
```

Um Daten zu senden, muss pro Befehl ein Slot definiert werden, in dem die zu sendenden Daten einfach mit Hilfe des folgenden Code gesendet werden.

```
//sende 1,2,3 über die USB Schnittstelle.
        m_transmissionData << 1;
        m_transmissionData << 2;
        m_transmissionData << 3;
```

Falls ein Befehl eingebaut werden soll, der eine Antwort über den USB Bus erwartet, muss dies folgendermaßen implementiert werden:

```
SCommand cmd;
       cmd.type = CMD_NEW_COMMAND; //Typ angeben
       cmd.bytesWaited = 4; //Anzahl der Bytes für eine korrekte Antwort
       cmd.data = 7; //kann für verschiedene Zwecke benutzt werden, nicht notwendig
       m_commandList.append(cmd); //Befehl speichern
```

Die Verarbeitung von Daten, die über USB empfangen wurden, muss in der Funktion CIOExpanderCommunicationModule:: ProcessReceivedData() ergänzt werden. Es ist sichergestellt, dass mindestens so viele Bytes empfangen wurden, wie benötigt werden.

```
switch(cmd.type) //bereits vorhanden
       {
              case CMD_NEW_COMMAND: //je nach Namen anpassen
              {
                     //Hier entsprechende Aktion ausführen
                     emit NewCommandSignal(); //zugehöriges Signal
              }break;
       };
```

Compilieren

Um das Programm zu compilieren, wird eine funktionierende Qt Installation auf dem Rechner vorrausgesetzt (Version 4.4.0 getestet, eine ältere 4.x Version sollte auch funktionieren). Das Programm wurde - wie die Software für den AVR - mit der Entwicklungsumgebung Codeblocks geschrieben, diese ist aber nicht notwendig, um das Programm zu übersetzen (es wird aber wieder eine Projektdatei mitgeliefert). Das Programm compiliert mit dem gcc Compiler (MinGW) unter Windows, sollte aber auch mit Visual Studio bzw unter Linux compilierbar sein (nicht getestet). Um das Programm zu erstellen muss man nur einmal im Projektordner ein „qmake“ ausführen und kann anschließend den Compiliervorgang mit einem „make“ starten. In der zip-Datei befindet sich eine QT-Projektdatei (IO-ExpanderControl.pro). Dort müssen auch bei Änderungen am Programm eventuell neue Sourcedateien hinzugefügt werden. „qmake“ erzeugt daraus schließlich das Makefile, mit dem „make“ arbeitet. Um das Programm zu starten, muss der FTDI Treiber auf dem Rechner installiert sein. Ansonsten erscheint ein Fehler wegen einer fehlenden DLL (FTD2XX.dll).

Achtung (nur Windows): Die auf der FTDI Seite erhältliche FTD2XX.lib arbeitet nicht ohne weiteres mit dem GCC Compiler zusammen, da diese für den Microsoft Compiler gedacht ist. Diese muss man erst in eine .a Datei konvertieren. Dazu wird das dlltool benötigt (bei MinGW dabei). Man kopiert die FTD2XX. dll sowie die zugehörige .lib und .def Datei in einen Ordner und tippt folgendes in eine Konsole:

```
dlltool --input-def FTD2XX.def --dllname
FTD2XX.dll --output-lib FTD2XX.a -k
```

Falls man anschließend compiliert, erhält man wahrscheinlich mehrere Linkerfehler in der Art „undefined Reference to XXX@n“, wobei XXX eine Funktion ist und n eine ganze Zahl. Um hier Abhilfe zu schaffen muss in der FTD2XX.def Datei jedes vorkommen von XXX durch XXX@n ersetzt werden und die .lib Datei erneut konvertiert werden. Dies muss für jeden Linkerfehler wiederholt werden. Eine FTD2XX.a Datei ist bereits im Paket enthalten, evtl. kann diese aber mit neueren Treibern nicht mehr verwendet werden.

Achtung (nur Linux): Der FTD2XX Header ist für Windows gedacht. Auf der FTDI Seite gibt es einen angepassten Header für Linux (im Treiber enthalten). Falls das Programm unter Linux verwendet werden soll, muss natürlich dieser verwendet werden. Außerdem muss man in der Datei „USBCommunicationModule.h“ folgenden Code entfernen:

```
#include <windows.h>
```

Dieser dient nur dazu, unter Windows dem FTDI Header verschiedene Variablentypen bekannt zu machen.

Definitionen der Befehle

Grundsätzlich ist jeder Befehl gleich aufgebaut. Das erste Byte kennzeichnet den eigentlichen Befehl, alle anderen (optionalen) Bytes haben bei jedem Befehl eine unterschiedliche Bedeutung. Hier ist eine Übersicht über alle momentan implementierten Befehle:

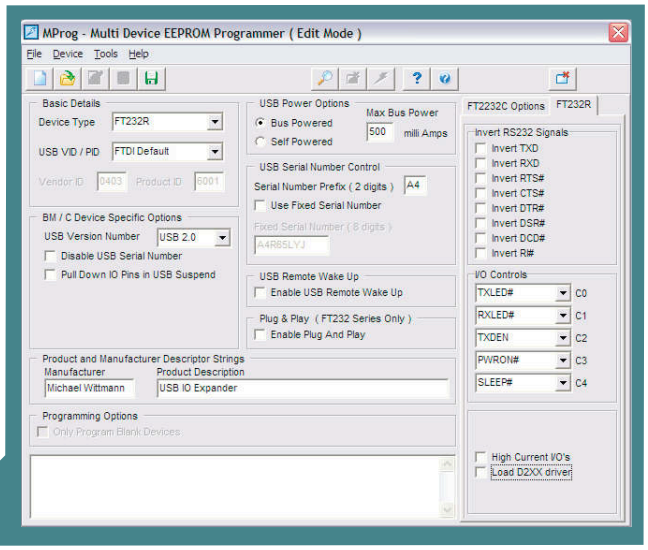
Funktion	1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6+ Bytes
Lese ADC Wert	‚a'	Kanal				
Setze Timer Frequenz	‚f'	Prescaler	OCR-Wert			
Schreibe Wert auf digitale Ausgänge	‚o'	Pin 0-7	Pin 8-15			
Lese digitale Eingänge	‚i'					
Schreibe Muster auf digitale Ausgänge	‚p'	Anzahl	Samples	Samples	Samples	Samples
Lese digitale Eingänge wiederholt	‚s'	Anzahl				
Kombination aus ‚s' und ‚p'	‚x'	Anzahl	Samples	Samples	Samples	Samples
Setzte Relais Status	‚r'	Relais Zustand				
Sende Daten über SPI	‚t'	Daten				
Konfiguriere SPI	‚c'	SPCR Register	SPSR Register			
Trigger setzen	‚z'	Trigger1	Trigger1	Trigger2	Trigger2	

Ein paar Worte zu den verwendeten Bauteilen

Ich habe hier bewusst keine Liste eingestellt, in der die genauen Typbezeichnungen und Menge jedes Bauteils stehen (das macht Eagle sowieso automatisch und den Schaltplan gibt es ja hier zum Download). Bei einigen Teilen bin ich nämlich noch auf der Suche nach der optimalen Variante. Zum einen wären das die beiden Optokoppler. Wie sich herausgestellt hat, ist es gar nicht so einfach, schnelle SMD Typen zu finden. Deshalb habe ich im Moment „nur" Standardtypen verbaut, die jedoch die UART Geschwindigkeit begrenzen. Die Übertragung läuft im Moment nur mit 19,2kBaud. Im Prinzip ist das jedoch ausreichend, da eine Echtzeitdatenübertragung von Anfang an nicht gefordert war. Zum anderen habe ich im Moment für den MosFET Q5 (über den der FT232 die Stromversorgung schaltet) einen Typen im TO220 Gehäuse gewählt, der nicht ganz ins Layout passt. Die einzigen Kriterien, die er erfüllen muss, sind bei 5V Gate-Spannung durchzuschalten (Logic-Level p-Kanal Typ) und einen maximalen Strom von mindestens 500mA schalten zu können. Eine Alternative dazu wäre ein FDS4435 im SO-8 Gehäuse. Raum für Verbesserungen bietet auch der DC/DC Wandler (momentan verwendet: SIM2-0505S SIL7 von Reichelt). Dieser liefert zwar eine ziemlich ungenaue Ausgangsspannug (evtl. wie weiter unten beschrieben mit einem zusätzlichen Lastwiderstand nachhelfen), funktioniert aber bis jetzt ohne Probleme.

Inbetriebnahme

Es biete sich an, zuerst den FTDI Chip nebst Peripherie sowie den DC/DC Wandler zu bestücken. So kann die korrekte Spannungsversorgung mit Hilfe der auf der Platine befindlichen Testpunkte geprüft werden. Bei einigen DC/DC Wandlern ist die Leerlaufspannung höher als die angegebene Spannung (gemessen bis zu 7V statt der angegebenen 5V). Unter Last geht sie auf das angegebene Niveau zurück. Deshalb kann es sein, dass man den DC/DC Wandler mit einem zusätzlichen Widerstand belasten muss, um die Spannung in einem verträglichen Bereich zu halten (max 5,5V). Der genaue Wert hängt vom Wandler ab und läst sich am einfachsten durch Ausprobieren herausfinden. Wenn die Spannung OK ist, kann man die restliche Platine gefahrlos bestücken. Wenn die Platine korrekt aufgebaut ist, und man sie über ein USB Kabel mit dem Rechner verbindet, sollte ein USB-serial-Converter erkannt werden. Dann muss mit dem Programm MProg (bei FTDI erhältlich) das EEPROM des FT232R programmiert werden. Die vorzunehmenden Einstellungen kann man hier sehen. Wirklich wichtig sind nur die „Product Description", da die Software an Hand derer das Gerät erkennt, sowie die 500mA und die Auswahl von „Bus Powered". Der „Load D2XX Driver" Schalter hat laut Dokumentation beim FT232R keinen Effekt, anscheinend wird immer sowohl der D2XX als auch der VCP Treiber geladen.



Problembehandlung

FT232 Ich habe beim Testen beobachtet, dass der FTDI-Treiber Probleme macht, falls man eine zu alte Version verwendet. Am sichersten ist es, den neuesten Treiber von der Homepage zu installieren und mit der zugehörigen FTD2XX.lib/FTD2XX.h das Programm zu compilieren. Ansonsten kann es vorkommen, dass während des Betrieb die Verbindung abreißt (also bei einem Treiberupgrade am besten das Programm neu compilieren).

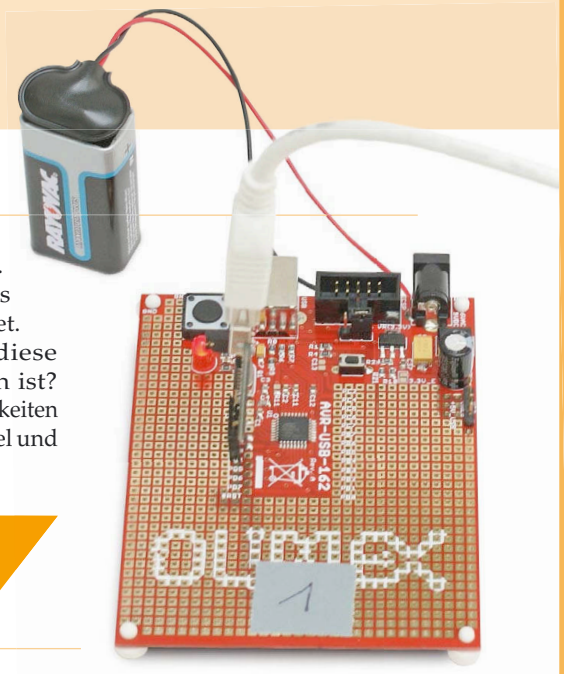
AVR Falls der Controller nach Aufspielen der Software keinen Mucks von sich gibt, wurden eventuell die Fusebits falsch gesetzt. Der Controller muss so eingestellt werden, dass er mit einem externen Quarz arbeitet (kein Oszillator). Die restlichen Fusebits können in der Standardeinstellung belassen werden.

USB Kommunikation Als erstes sollte man prüfen, ob der FTDI-Treiber in der neuesten Version installiert ist und ob die MProg Einstellungen korrekt sind. Ansonsten noch einmal die Löstellen am FT232R überprüfen (eventuell Lupe?!) bzw kontrollieren ob der Quarz schwingt.

Digitale Ein-/Ausgänge Eventuell sind die 74AC245 Treiber falsch herum in die Sockel eingesetzt. Sind die Lötstellen an den Schiebregistern (SMD) korrekt?

UART mal ohne Kabel...

Michael Hartmann <michael@speicherleck.de>



RS232 in Zeiten von USB

Obwohl UART eigentlich längst von USB abgelöst worden ist, erfreut sich diese Art der Datenübertragung innerhalb der Elektronik- und Microcontroller-Community noch immer großer Beliebtheit. Die Schaltung ist schnell aufgebaut und preiswert, denn neben einer Hand voll Kondensatoren und einer Buchse braucht man nur einen einfachen Pegelwandler.
Allerdings besitzen, gerade auf Grund der Dominanz von USB, nicht mehr alle Computer einen seriellen RS232 Anschluss. Vor allem bei Notebooks wird aus Platzgründen gerne darauf verzichtet. Was aber tun, wenn man auf diese einfache Schnittstelle angewiesen ist? In diesem Artikel werden zwei Möglichkeiten für eine moderne Anbindung per Kabel und Funk über USB an RS232 gezeigt.

Abb.1: Serielle Verbindung mit USB-zu-RS232 Wandler

Kabelgebundene Lösung

Abhilfe für das oben genannte Problem schafft hier ein USB-zu-RS232 Wandler. Die UART-Bridge wird zum PC hin per USB verbunden. Für die Verbindung zum Microcontroller verwendet man die Pins VCC, GND, TX und RX. Für eine serielle Kommunikation genügt es, den GND Pin eines USB-zu-RS232-Wandlers mit GND der Schaltung zu verbinden, VCC mit VCC, RX des Wandlers mit TX und TX des Wandlers mit RX des Microcontrollers. Der Adapter und das später beschriebene Funkmodul, welche in diesem Artikel verwendet werden, findet man hier [1]. Es gibt jedoch viele andere weitere Wandler, die dasselbe können.

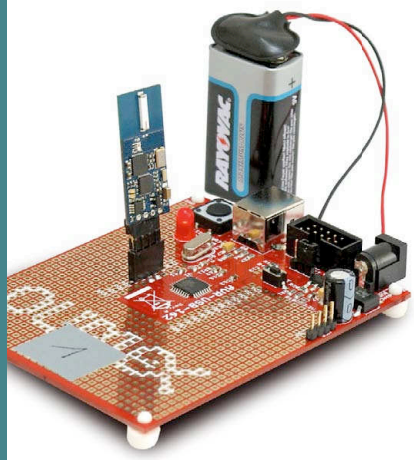
Mögliche USB zu RS232 Wandler:
- USB-zu-RS232 Wandler (von http:// eproo.de)
- TTL-232 Kabel von FTDI
- OpenOCD USB
- CP2102 (USB zu RS232 Wandler-Baustein)
- FT232 (USB zu RS232 Wandler-Baustein)

Durch diesen Aufbau kann man auf die Platz raubenden Elemente der klassischen Schaltung, nämlich den Pegelwandler (meist einen max232), die Buchse und die Kondensatoren verzichten. Auf dem Board selbst benötigt man nur noch eine 4-polige Stiftleiste, deren Pins mit GND, TX, RX und VCC verbunden sind. Die Schaltung wird dadurch kompakter und ein Pegelwandler ist nicht mehr notwendig. Ein USB-zu-RS232-Wandler kostet in etwa 15 Euro.

Unter Linux kann man weiterhin das gewohnte Terminal verwenden. Nach dem Einstecken wird bei aktuellen Distributionen eine Datei unterhalb von /dev - oftmals /dev/ttyUSBN - erstellt. Für Windows gibt es Treiber, so dass auch hier das alte Terminal weiter verwendet werden kann.

Lösung per Funk

Anstatt den USB-zu-RS232-Wandler direkt per Kabel mit dem Microcontroller zu verbinden, kann man auch Funkmodule verwenden. Dafür braucht man zwei Funkmodule: Eines, das mit dem Microcontroller verbunden ist und ein Weiteres, das auf dem USB-zu-RS232-Wandler steckt. Ein Funkmodul ist mit ca. 30 Euro pro Stück relativ teuer. Auf Grund seiner Reichweite von 200 Metern ist es aber deutlich besser für einen Einsatz im Freien geeignet, als die Kabellösung. Und sogar innerhalb von Häusern können Funkmodule durchaus ihre Vorteile haben, vor allem dann, wenn sperrige Wände im Weg sind. Leider schrauben sie jedoch die große Reichweite der Funkmodule wieder stark zurück.



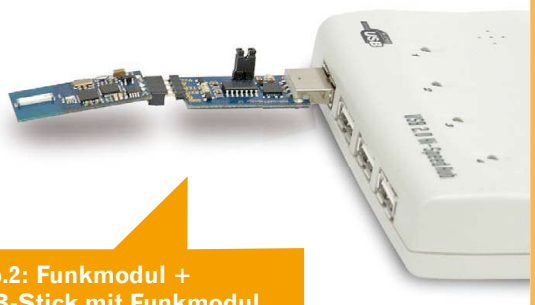


Abb.2: Funkmodul + USB-Stick mit Funkmodul

Downloads

Die Platinen als Eagle Dateien
Die Software für den PC
Die Software für den AVR

http://www.mikrocontroller.net/articles/USB_IO_Expander

Links

Codeblocks
(damit wurde dieses Projekt erstellt)
http://www.codeblocks.org/
Qt Bibliothek
http://www.qtsoftware.com/downloads/
Hier gibts alles für den USB-seriell Wandler
http://www.ftdichip.com/

Lösung per Funk

Der USB zu RS232 Wandler und das Funkmodul sind komplett kompatibel zu einer UART-Umsetzung mit Pegelwandler und Kabel. Der bestehende Code muss also nicht angepasst werden. Der PC empfängt Text, den der Microcontroller per UART und Funk schickt (siehe Abbilung 4).

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

int main(void)
{
   UARTInit(38400);
   while(1)
   {
        UARTSend(„Ich werde per Funk übertragen\r\n");
        _delay_ms(1000);
   }
}
```

Abb.3: Quelltext main.c

Abb.4: Terminal

Fazit

Ziel des Artikels war es kurz zu zeigen wie mit modernen USB-zu-RS232 Wandlern sich einiges an Bauteilen gespart werden kann und wie komfortabel das Arbeiten dank moderner Funkübertragungen möglich ist.

Links

http://www.eproo.de

MyCPU - A Homebrew Computer

Dennis Kuschel <dennis_k@freenet.de>

MyCPU

„MyCPU“ is a homebrew CPU that was primary developed for fun. It is a simple 8-bit processor that is completely built with simple discrete logic gates of the 74xx CMOS series. Over the years the project has grown, and now MyCPU is the brain of a real Personal Computer. The project is totally open source, and everybody is invited to participate and contribute to the project.

Currently the computer consists of the following main parts:
8-bit processor, consisting of around 65 integrated circuits placed on 5 euro cards
Interrupt Controller Board, providing 7 edge-triggered, maskable interrupts and 1 timer interrupt
Memory Unit with up to 1 MByte bank switched RAM and 160 kByte ROM
Multi-I/O-Unit with keyboard, LCD, 2 x RS232 and one parallel printer port interface.
IDE Controller Board with RTC/CMOS and two IDE ports for up to four IDE devices / harddisks
VGA Graphic Unit providing 16 colours and 80x50 character text mode and several graphic modes
Ethernet Controller with 10BaseT interface to allow simple connectivity to the Internet

The processor performance is around 1 MIPS (where an integer has 8 bit). The CPU can be clocked up to 8 MHz. The speed is limited by the slow EPROM‘s used in the CPU design and by the slow external components. Because of the cable length and mismatched line impedance the processor bus cannot be clocked higher than 4 MHz, making it to the bottleneck.

The fully equipped system has a power consumption of less than 3.5 Watt. This is due to the fact that only CMOS chips are used and the power dissipation at 8 MHz clock frequency is near zero. Most power is consumed by the VGA Unit (1.3 Watt) that runs at 25 MHz.

The MyCPU is not only a hardware but also a software project. Today much software is available. The Operating System that resides in an EPROM is completely written in assembly. The OS is a mixture of DOS, Unix and Commodore 64, and provides most of the well-known commands to manage hard disks. MyCPU has its own file system that is able to handle up to 8 disks or partitions. Drivers for additional external hardware components like the Ethernet Controller can be loaded from disk.
Furthermore MyCPU OS comes with a build-in basic interpreter that is compatible to the basic interpreter of Commodore 64. Old pure basic programs that were written for C64 can be run on MyCPU without modification.
MyCPU can be programmed in three more programming languages: For the fans of higher-level languages a „C“-Cross-Compiler is available. And hardcore freaks can write their programs directly in assembler or in the esoteric programming language „brainfuck“ (MyCPU OS comes with a build-in JIT compiler for brainfuck).
The following application software is currently available for MyCPU: Basic interpreter, brainfuck compiler, assembler, text editor, games and demo‘s, networking software like HTTP web server and a Telnet demon. And of course there are some more tools available that make life with MyCPU more convenient.

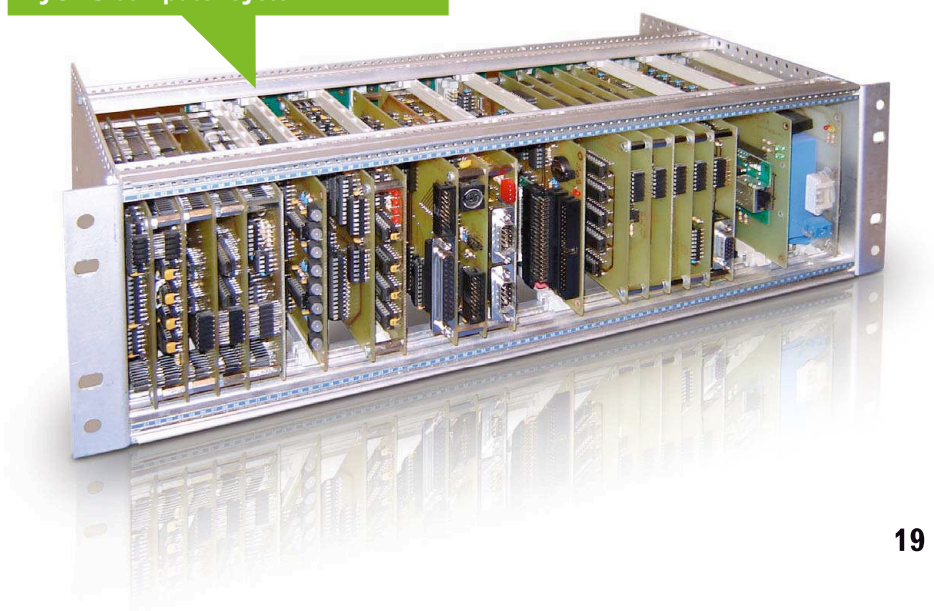
Technical data of MyCPU

- 8-bit processor, Harvard architecture
- 16 bit address bus, 2x64 kB address range
- runs up to 8 MHz
- 8-bit integer performance is around 1 MIPS
- 247 OP-Codes, 9 are reserved for future use
- 14 addressing modes
- 256 byte stack memory
- 1 non-maskable software interrupt
- 1 maskable hardware interrupt
- 65 integrated circuits on 5 boards

The Computer

Where „MyCPU“ stands for the processor itself, I use the term „MyCPU System“ for the whole home-brew computer. The fully equipped MyCPU System currently consists of the following components (compare with **figure[1]**, from left to right): MyCPU (5 boards), Interrupt Controller (1 board), Memory (2 boards), I/O (3 boards), IDE Interface (1 board), VGA Graphic (6 boards), Ethernet Controller (1 board) and the Power Supply (1 board). The system provides this interfaces to the outside world: PS/2 Keyboard, alpha-numeric LC-Display, 2 RS232 Interfaces, Parallel Printer Port (Centronics), 8 universal inputs and 8 universal outputs, VGA graphic interface (RGB, 15 pin high density Sub-D) and 10 MBit Ethernet (RJ45 twisted pair). Because I have not written the FTP server software yet, data communication with a PC is still done through a serial link via RS232 and special software. But note: You can now trash your PC since there is everything available that you need to work stand-alone with the MyCPU System ;-). For example there is an assembler program available that runs on MyCPU and produces code for MyCPU. The evolutionary step is that the assembler can assemble itself on MyCPU. So you can write software for MyCPU on MyCPU, and you are no more required to use a PC for special tasks.

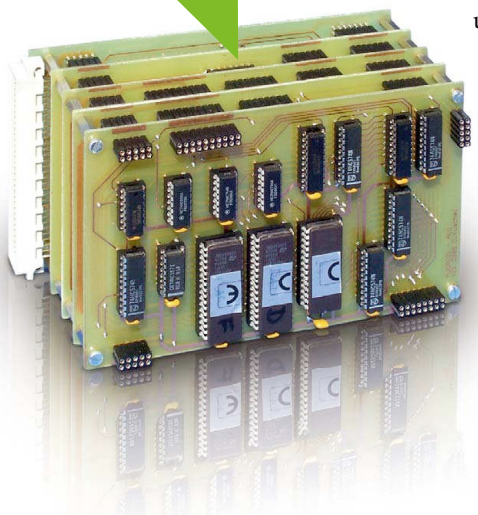
figure [1]: Fully equipped MyCPU computer system



A closer look to the processor

The MyCPU processor (depicted in **figure[2]**) is a complex microcode machine (CISC), like any modern processor today. The internals of my CPU are shown in **figure[3]**. The core of the processor is the microcode engine that controls the data flow from and to the registers and arithmetic components. The processor has only 5 universal and 1 special purpose register: A, X, Y, P and SP are universal 8-bit registers, whereas the Program Counter (PC) is a special purpose 16-bit register whose contents can be incremented by one through a dedicated signal. The P-register is only an internal register used by the microcode to store temporary data and is not visible to assembly programs. In my implementation the microcode covers many commands and addressing modes of the good old 6502 processor, but MyCPU is not binary compatible to it. The microcode can be adopted to simulate other processors like Z80, 8051 or whatever; the only limitation is the number of internal CPU registers. For Z80 or 8051 it becomes necessary to store registers in external memory. The microcode was developed with the help of a „macro language". The Microcode-Compiler is written in „C" and does also some validity checks and optimisations on the generated microcode. For example the OP-Code „CLA" (clear accumulator) is constructed from the following micro-OP's: fetch_opcode_and_increment_pc, move_const0_to_accu, update_flags, next_op. The processor design is optimised for speed. With easy to obtain parts like usual EPROM's with 100ns propagation delay and the usual 74HCxxx series of logic gates MyCPU can be run with at least 4 MHz. It is possible to clock MyCPU even higher, but than some parts must be replaced, e.g. faster EPROM's are needed and some 74HC-chips must be replaced by faster 74AC-chips. I have successfully run a MyCPU at above 8 MHz, but that is only possible with a short bus length and a reduced number of peripheral boards. My goals was to have a fast and efficient processor that requires as less parts as possible by providing a superior high processing power. The MyCPU design is a good compromise. Of course it would be nice to have a 16-bit processor with 32-bit address range, but this would double the amount of required parts.

figure [2]: MyCPU processor cubesystem



The System's Memory

The MyCPU System is equipped with 1 MB RAM and 160 kB ROM. Since MyCPU has only an address range of 16 bit (that means 64 kB of linear memory can be accessed), it is necessary to do some bank switching to allow MyCPU to access all the memory. The ROM is partitioned into chunks of 32 kB. The EPROM that contains the Operating System Kernel is always visible and is mapped to the address range 0000h - 7FFFh. The other four 32 kB ROM partitions can be mapped to 8000h - FFFFh. These ROM partitions contain the filesystem driver, the shell and the Basic Interpreter. The RAM can also be mapped into the area between 8000h and FFFFh. Thus, a program that is loaded from disk can have a maximum size of 32 kB. That is why it is not possible to run Linux on MyCPU (there are some other limitations also, e.g. Linux requires at least a 16 bit processor). There are some more special features on the memory board: The RAM can be split into 16kB chunks that can be paged into the area 4000h to 7FFFh. Furthermore the two lowest 256 byte pages, which are the zero-page and the stack-page, can also be bank-switched. This allows the use of more stack memory and enables MyCPU to run multitasking operating systems. I have successfully ported the RTOS „PicoJOS" **link[2]** to MyCPU. OK, that is only half of the truth. Remember MyCPU has a Harvard like architecture, except that there are no two separate buses to access program and data memory. It's more like the 8051; MyCPU supports two different read signals, one for code- and one for data-memory. If you are interested in how memory bank switching really works please read the Memory-Selfbuild-Guide that can be found on my website **link[1]**. Physically the Memory Unit consists of two boards: The baseboard provides 512 kB RAM and 160 kB ROM. The optional extension board adds 512 kB more RAM and a second paging unit.

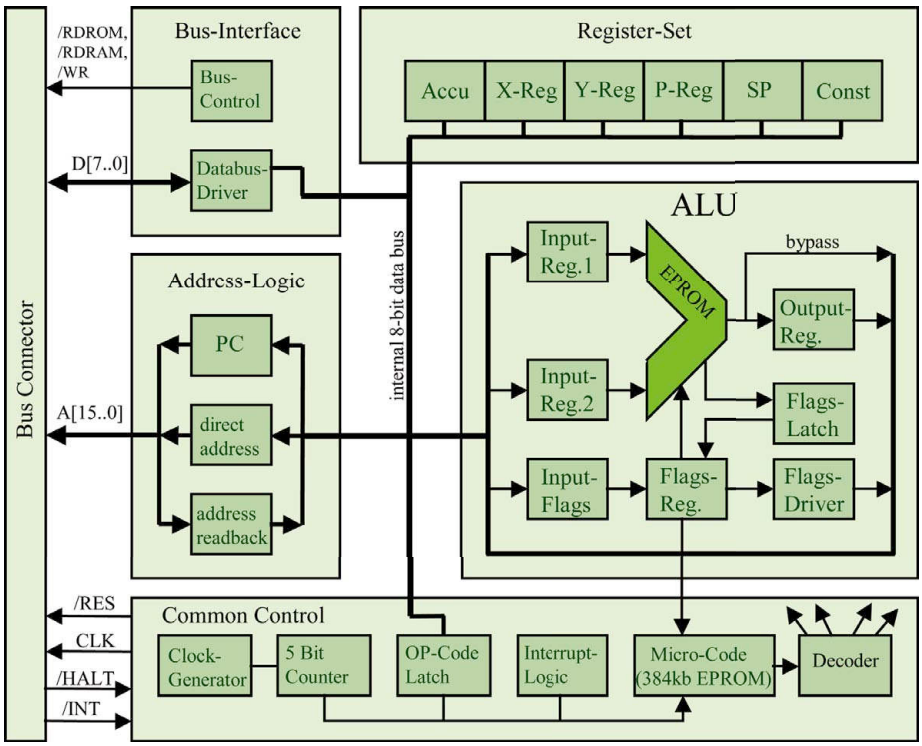


figure [3]: Internal architecture of MyCPU

A minimalist Computer System

figure[4] shows how a minimalist computer system with MyCPU looks like. At the left side you see the MyCPU cube. The two other boards are the memory baseboard with 512 kB RAM and the RS232 interface board that mainly consists of two commercial 16C550 UART's and some glue-logic. User communication is done with a terminal program like Hyperterm on MS Windows or minicom on Linux. A second serial link is used for file transfer between MyCPU and a PC. The PC must run special server software for this purpose. When the System is powered on it searches for boot files on the PC. The user can choose one of four software packages that is then loaded into a RAM-disk on MyCPU. With this minimalist computer system the user can write and save texts, write Basic programs, write and assemble Assembler programs, play games and run network services like the http server and Telnet over a SLIP link.

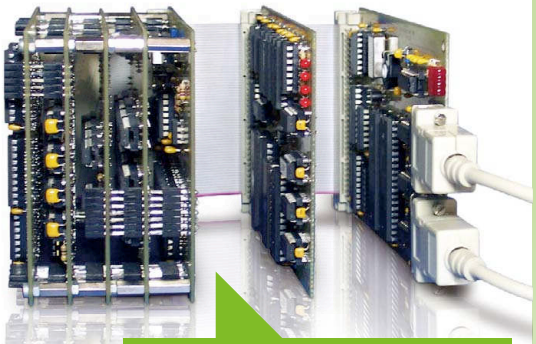


figure [4]: Smallest possible MyCPU computer system

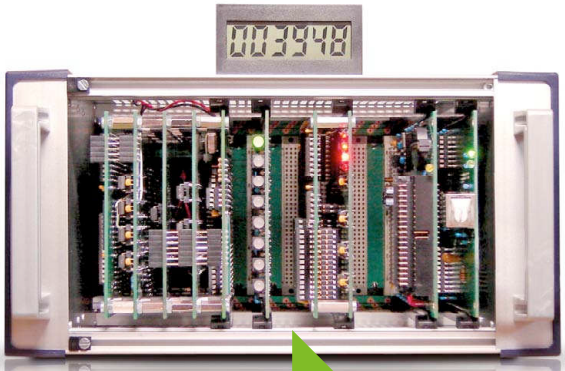


figure [5]: The MyCPU-Webserver provides the content for the domain www.mycpu.eu. The website is stored on a CF-Cardand the server is managed via Telnet. On top is the page-hit-counter.

Want to build your own MyCPU Computer?

Since the MyCPU Computer is completely open source you can get all the schematics and software from my website link[1]. You should start with building the MyCPU itself and the memory board. For this purpose you need to download the „MyCPU Selfbuild Guide" and the „Memory Selfbuild Guide". When the set-up of CPU and memory works as expected, you can continue to build the serial interface board. The fourth board would be the Interrupt Controller which is required to be able to use the next boards that are: Keyboard/LCD, IDE Controller, VGA and Ethernet (in any order). Are you interested? If yes, you may also try out the MyCPU Emulator that can be downloaded from my website (go to „all downloads" and scroll down).

About the history of MyCPU

It all started in February 2001.
I was learning for my diploma examination in digital electronics and got bored. I wanted to do something productive with the knowledge I just obtained. So I decided to construct an electronic calculator machine by using only fundamental digital gates of the 74-TTL series from which I still had some hands full in a drawer. At this point it was still nothing special to construct a calculator, many people would have done this before. It was just an education project for me.

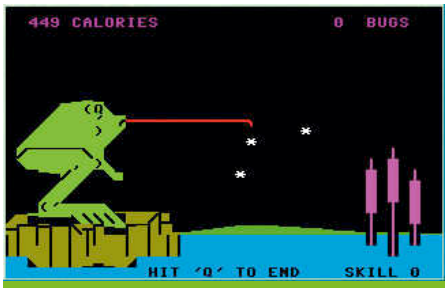


figure [6]: MyCPU running an old Basic game by Commodore.

Which way to go?
Because I already had some experience with the very simple but powerful 6502 CPU, the idea was not to implement the calculator algorithms in hardware but to construct some programmable machine that should be similar to the 6502. If I could port the Commodore 64 Basic Interpreter to my machine I would get the floating-point routines for my calculator for free.

8 bit are enough!
To keep the design as simple as possible 8 bit should be enough for my machine. Maybe you still remember the Commodore 64 and the powerful games and demos that ran on it. And this with only 64 kB RAM and less than 1 MHz clock frequency. That should be possible for my machine, too.
My philosophy was and is until today to use only the simple gate parts. No higher integrated components like GAL's, PAL's and FPGA's are allowed in my design (this would be too easy and was already done by many other people). The only higher integrated parts that are permitted are EPROM's because I needed them to store the calculator program anyway.

How to start?
I sketched the schematics on a piece of paper. The draft of the CPU was finished within two weeks (while I was still learning for my examination). My idea was to have a micro code based machine. The advantage is that only a few components are required. The core logic would reside in a re-programmable EPROM. And because EPROM's are so universal I also constructed the ALU (Arithmetic Logic Unit) with some big EPROM's that are simply used as look-up tables. Thus an 8 x 8 -bit multiplication can be done within only one clock cycle.

Simulation!
But before I would start constructing the processor in hardware (what would be an exhausting task) I wanted to test my design. My approach was to write a CPU emulation in the programming language „C". Version 1 of the emulator simulated every single gate of the processor, including the timing constraints. The result was that the design was functional!

What's with software?
The first emulator simulated the processor (gate by gate), 32 kB ROM and 32 kB RAM. User interaction was done by a simulated serial terminal. The test software simply read some input from the terminal and outputted it immediately again. The first „stupid" version of my today's kShell was born! (This was around April 2001; I already finished my diploma with best degree).
And now: Hardware! Because all went so fine I started constructing „my CPU" in hardware. The first set-up consisted of the CPU, a memory board and a couple of LED's for I/O. As next I constructed the interrupt controller and the RS232 I/O-board. The keyboard interface and the VGA Unit followed, and at the beginning of 2002 I had a fully featured „Personal Computer" on my desk. A friend of mine was so enthused of my project that he recommended me to share the project with others. So I started my website yet in September 2001.

In the following years
I improved the design and wrote the self-build guides. I spent much time with developing the application software. Even today I am not finished with the CPU design, and I will possibly never be. There are still too much things to improve. My slogan is: „Others have a model railway in their cellar, I have my CPU instead."

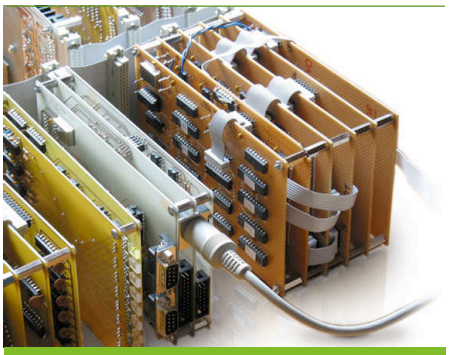


figure [7]: MyCPU in an early phase. You see the prototype of the VGA Graphic Unit and besides the Multi-I/O-Unit and the Interrupt Controller.

Links

[1] The MyCPU project website:
http://www.mycpu.eu
[2] Pico]OS real-time OS:
http://picoos.sourceforge.net

[IMPRESSUM]

embedded projects GmbH
Holzbachstrasse 4
D-86152 Augsburg
Telefon: +49 821 31946-23
Telefax: +49 821 31946-24
Mail: journal@embedded-projects.net

Anzeigemöglichkeiten und Preisliste auf Anfrage via Mail.

Herausgeber: Benedikt Sauter
Gestaltung/Satz:Das-Medienkollektiv.de

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

Dies ist ein Open-Source Projekt.
Informationen zum ABO - www.embedded-projects.net/journal

EPJ Fotonachweise
Titel: starJumper@fotolia.com
EPJ Anzeige: Ramona Heim, Nicemonkey, dip,...@fotolia.com

Programmierung Grasshopper

Dokumentation für ein Beispielprojekt mit Standard C Programmierung des Grasshopperboards (ATMEL AVR32 AP7000 CPU) Stand: 06.04.2008

Autor unbekannt, Bearbeitung Benedikt Sauter <sauter@embedded-projects.net>

Einleitung

Dieses Dokument soll ein Beispiel C-Projekt für das Grasshopperboard mit der AVR32 Toolchain und AVR32 Studionutzung unter Windows XP beschreiben.
Es soll beschrieben werden, wie mit Standard C eine Anwendung programmiert und dann auf das Board im Flash gespeichert wird. Dazu wird **NICHT** der Bootloader U-Boot genutzt. **Das komplette Flash wird gelöscht, d.h. der Bootloader U-Boot und auch das vorinstallierte Linux gehen verloren.** Man kann sie später wieder mit dem JTAG ICE MKII auf das Boardprogrammieren.

Benötigte Hardware
· Grasshopper Board (logo)
· JTAG ICE MKII von ATMEL zum programmieren
· Zum Nutzen der Ausgabe von printf() auf dem 3.3V zu RS232 benötigt. Eine UART-Schnittstelle einen 4-poligen Pfostenverbinder gelegt (zu finden TAG Pfostenverbinder). Für den Pegelwandler kann man z.B. einen MAX3232 verwenden. **ACHTUNG NICHT MAX232!!**

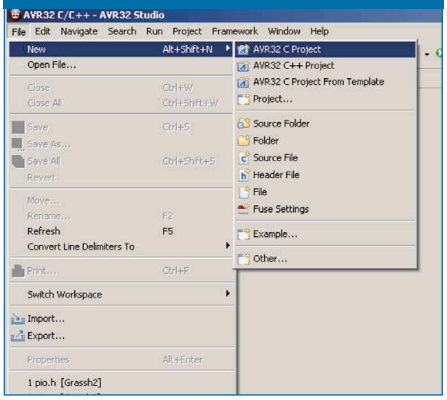
Notwendige Software Pakete von ATMEL
Im Moment noch zu zu finden unter: **http://www.atmel.no/beta_ware** Ich habe die „stable" Versionen nicht probiert. Diese Versionen hier liefen bei mir aber ohne Problem.
- AVR32 Studio 2.0 second release candidate
- GNU Toolchain 2.0 second release candidate

RTFM -> Empfohlene Docs:
Leider habe ich keine brauchbare Dokumentation über die Implementierung der Newlib (libc/libm) gefunden. Ich habe teilweise den Toolchain-Source genutzt, zu finden unter **http://www.atmel.com/dyn/resources/prod_documents/avr32-gnu-toolchain-1.3.2-0.exe**

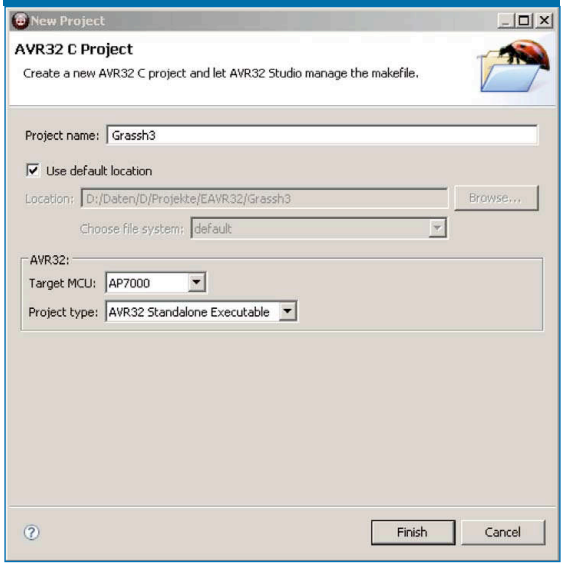
Es ist scheinbar einiges implementiert aber leider nicht dokumentiert. Hilfreich ist auch das Wiki bei **http://www.avrfreaks.com** Die BSP (Board Suppport Packages) zum STK1000 und NGW von ATMEL helfen auch weiter und natürlich die Application Notes (als Dokument oder auch als Beispielprogramm), ebenfalls von ATMEL.

Folgende Dokumente sind hilfreich:
- ATMEL: AVR32015: AVR32 Studio getting started
- ATMEL: AVR32006 : Getting started with GCC for AVR32
- ATMEL: AVR32000: Introduction to AVR32 Headerfiles
- ATMEL: AT32AP7000 AP7000 Manual
- ATMEL: AVR32 Architecture Document
- ATMEL: AVR32 AP Technical Reference Manual

1: Beispielprojekt erstellen



2: Eintrag Projektnamen & Typ



Toolchain Installation

AVR32-Studio und die Toolchain sind beides ausführbare EXE-Dateien, die wie gewohnt starten und installieren. Wie man dann mit dem AVR32-Studio ein Projekt erstellt, kompiliert und mit dem JTAG ICE testet, ist im oben genannten Dokument „AVR32015: AVR32 Studio getting started" beschrieben. Im Folgenden auch mehr oder weniger genau:

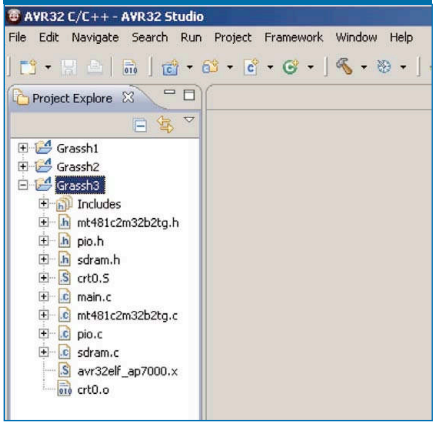
Beispielprojekt erstellen

1 Im AVR32-Studio mit File-New-AVR32 C Project ein neues Projekt erstellen.

2 Hier den Projektnamen eintragen, die CPU AP7000 und den Projecttype „AVR32 Standalone Executable" einstellen.
Jetzt müssen die mitgelieferten Beispiel C-Sourcen importiert werden. Dazu rechter Mausklick auf das Projekt und „Import" wählen. Dann in dem Fenster die Option „General- FileSystem" wählen, die Directory auswählen, in der die Dateien liegen und die *.c, *.h *.S *.o Files importieren.

3 Der Projektbaum im AVR32-Studio sollte dann so ähnlich aussehen (Projekt Grassh3).

3: Projektbaum



4 Jetzt müssen noch die Project-Properties eingestellt werden. Das die Einstellungen für den Compiler und den Linker. Dazu das Projekt anklicken (markieren) und im Menu unter Project-Properties die Property-Seite öffnen. Jetzt unter C/C++ Build das Menu „Settings“ öffnen. Das sieht so aus wie die Grafik oben. Hier kann man jetzt die Compiler und Linkereinstellungen vornehmen.

5 Es müssen in unserem Fall nur Linkereinstellungen vorgenommen werden, der Rest stimmt mit den Default-Einstellungen. Dazu unter: AVR32/GNU C Linker

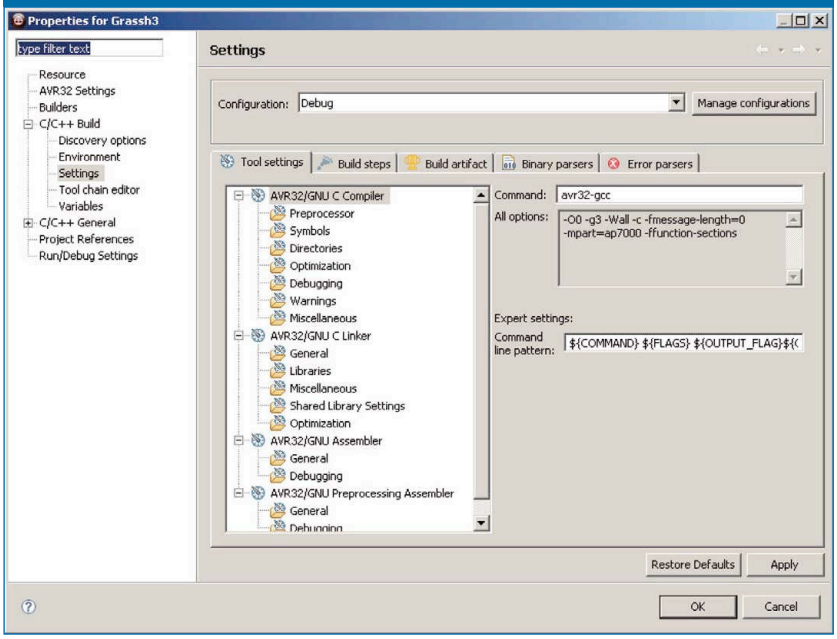
- General: „Do not use standard startfiles“ markieren, alles andere darf nicht markiert sein.
- Miscellaneous: Other Options (-Xlinker) zwei Einträge erzeugen: 1) nur ein „-T“ und 2) die Datei „avr32elf_ap7000.x“ inkl. Pfad eintragen (Grafik 5).
- Miscellaneous: Other Objects: die Datei crt0.o inkl. Pfad eintragen (Grafik 5).

Jetzt im Projektbaum die Datei „avr32elf_ap7000.x“ mit der rechten Maustste anklicken und Properties öffnen. Hier können für jeden Sourcefile spezielle Einstellungen vorgenommen werden. Für diese Datei muß „Exclude resource from build“ unter C(C++ Build-Settings markiert werden, da das Studio diese Datei sonst kompilieren will (warum das so ist weiß ich nicht).

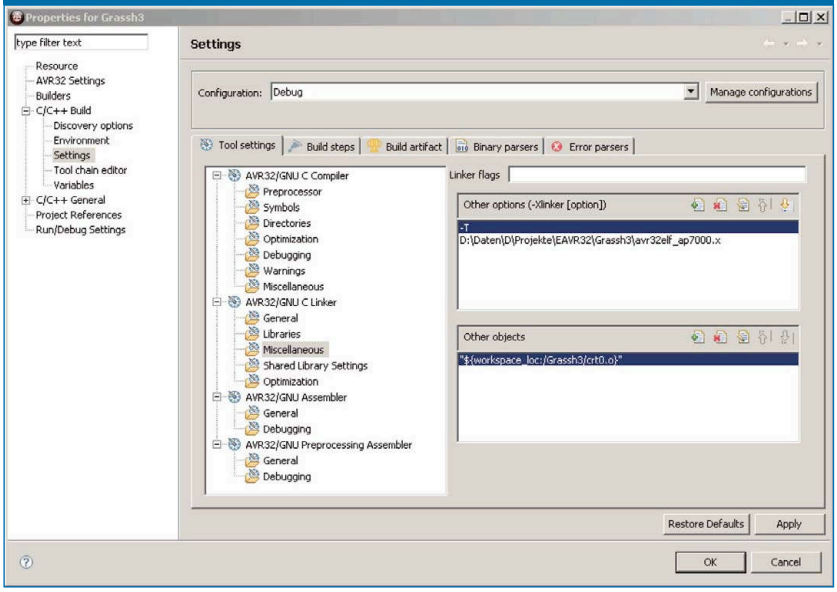
6 Beispielprojekt kompilieren

Jetzt kann man das Projekt kompilieren. Dazu unter Projekt „Build Project“ anklicken. Unter den Studio Views (Window-Show Views) „Problems“ bzw. in dem View „Console“ kann man sehen, ob alles erfolgreich war. Bei mir sieht die Consolenausgabe im Studio so aus:

4: Property-Seite



5: Linkereinstellungen



6: Kompilierung

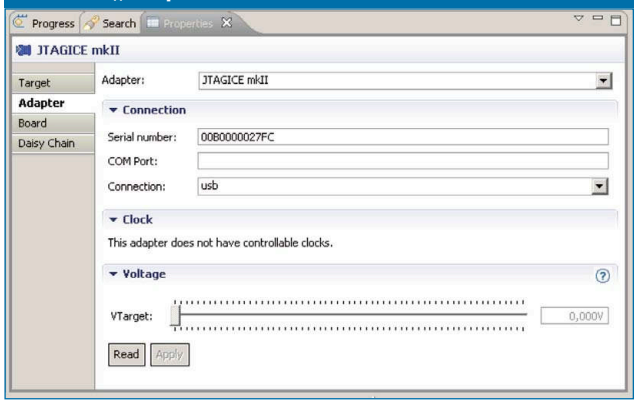
```
**** Rebuild of configuration Debug for project Grassh3 ****

**** Internal Builder is used for build ****
avr32-gcc -O0 -g3 -Wall -c -fmessage-length=0 -mpart=ap7000 -ffunctionsections
-osdram.o ..\sdram.c
avr32-gcc -O0 -g3 -Wall -c -fmessage-length=0 -mpart=ap7000 -ffunctionsections
-omain.o ..\main.c
avr32-gcc -O0 -g3 -Wall -c -fmessage-length=0 -mpart=ap7000 -ffunctionsections
-omt481c2m32b2tg.o ..\mt481c2m32b2tg.c
avr32-gcc -O0 -g3 -Wall -c -fmessage-length=0 -mpart=ap7000 -ffunctionsections
-opio.o ..\pio.c
avr32-gcc -nostartfiles -Xlinker -T -Xlinker
D:\Daten\D\Projekte\EAVR32\Grassh3\avr32elf_ap7000.x -mpart=ap7000 -Wl,--
gc-sections -oGrassh3.elf sdram.o pio.o mt481c2m32b2tg.o main.o
D:\Daten\D\Projekte\EAVR32\Grassh3\crt0.o
Build complete for project Grassh3
Time consumed: 3219 ms.
```

Beispielprojekt mittels JTAG ICE MKII auf das Grasshopperboard bringen:

Ich gehe von vorhandenen Hardwareverbindungen aus, also PC, JTAG ICE und Grasshopper sind verbunden und eingeschaltet. Auch die serielle Schnittstelle des Grasshoppers ist mit dem PC über den Pegelwandler verbunden und ein Terminalprogramm (115200 Baud, 8 Datenbit, no parity, 1 Stop bit) ist aktiv. Von richtig installiertem AVR32-Studio inkl. des USB-Treibers für dem ICE gehe ich auch aus.

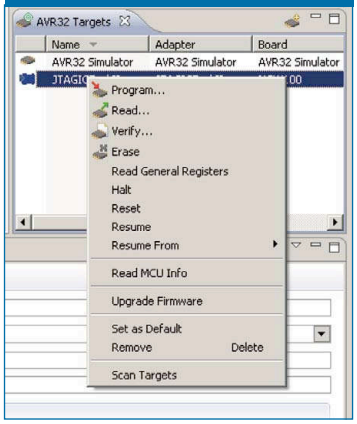
7 Im Studio jetzt in der View „AVR32 Targets“ rechter Mausklick und „Scan Targets“ aufrufen wenn der JTAG ICE nicht schon als Target vorhanden ist. Er sollte den ICE dann finden. Jetzt den ICE anklicken. Im View „Properties“ unter Adapter connection =”USB” eintragen. Unter Board das “NGW100” eintragen. MCU = „AP7000“. Da das Studio den Grasshopper nicht kennt (ich weiß auch nicht, wie man es ihm beibringt) wird das NGW100 eingetragen, da es vom Flash und er Adressierung des Flashs mit dem Grasshopper kompatibel ist. Alle anderen Einstellungen im View „Properties“ kann man so lassen.

7: View „Properties“

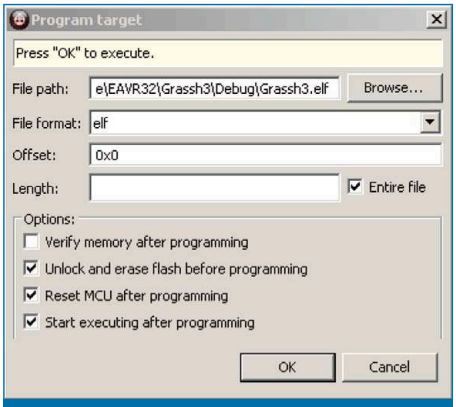


8 Jetzt kann man mit rechtem Mausklick im Fenster „AVR32 Targets“ auf den ICE ein Kontext-Menu öffnen.

8: Kontextmenü



9 Mit Klick auf Programm öffnet sich der Dialog zum Programmieren des Grasshoppers.



9: Programm - Dialogfeld

Dort die Daten wie oben angegeben eintragen. Als Filepath aus dem Workspace das Projekt wählen und dort das Directory „Debug“. Dort sollte der erzeugte ELF-File stehen. Dann OK klicken und der Grasshopper sollte programmiert werden. Wenn das beendet ist, dann sollten die rote LED und von den grünen die jeweils äußeren gleichzeitig blinken. Auf dem Terminal sollten Ausgaben erscheinen.

Wie man das Programm mit dem Debugger testet, ist hier nicht beschrieben. Dazu die Hilfe des Studios studieren oder in Foren z.B. www.avrfreaks.net lesen.
Viel Spaß.

Disko - A GUI toolkit for set-top boxes

Stefan Schwarzer

Introduction

Disko is a toolkit that can be used to put rich GUI application together and aims at set-top boxes as well as touch screen terminals. It tries to seamlessly integrate some great technologies like xine[1] or gstreamer [2] with a GUI toolkit that delivers more functionality than the usual media player overlays. It works on top of DirectFB [3] or directly on the Linux framebuffer as well as X11 [4]. To have a closer look at Disko and its capabilities you take a look at Morphine. TV [5] or 23p [6]. More information can be obtained at the Disko home page [7].

Getting Started

To start development with Disko it is recommended to use the latest development files. Get the source code from the public git repository:

```
git clone git:
//www.diskohq.org/disko.git
```

The SCons build environment is used to compile and install the toolkit, see scons *--help* for the build options.

The first application

Because this article is too short to cover all the necessary aspects and files needed to have a closer look to, I will make some references to the Disko tutorials [8] on the Disko homepage.

Initialization

Now were going to do the first steps in Disko development. This first example simply initializes an basic application and displays a simple "Hello world".

The initialization process sets Disko up for the environment it will run within. To make things a bit easier we will assume that we are in an X11 environment.

Disko is intialized using the *mmsinit()* function. This function receives the information needed to power up the application. The *MMSINIT_FLAGS are passed to mmsinit()* to specify the Disko subsystems that are needed by the application like graphics, windowing, plugins, events, inputs, or theming. In this case we confine the initialization to the windowing environment, which triggers the graphics system implicitly.

Configuration files

There are also some configuration files needed for a successful initialization. The main configuration file *diskorc.xml*, the input mapping file *inputmap.xml* and the theme file *theme.xml* namely the additional configuration files *theme.xml* and *inputmap.xml* only have to exist. Both of them can be set up with an empty root tag <theme/> and <inputmap/> respectively. The main application configuration must be populated with the proper settings for the environment. An example configuration for the X11 environment can be obtained from the Disko tutorials.

Hello world!

When the proper initialization is done and went well, we need a *Window*. A window is an entity that contains other graphical items referred as *Widgets* within Disko. These windows can be shown, hidden or placed anywhere on the screen. One specialty in Disko is the fixed z-order of these windows. That means there are 3 classes of Windows: *Root*, *Main* and *Popupwindows*. Rootwindows go to the back, Mainwindows go on top of them, and *Popupwindows* above all others. For each class of window there can only one window be shown at the same time. The windowing environment takes care that a *Window* is properly replaced, when another window of that type is to be shown. That enables independently operating plug-ins to inter-operate with the screen without leaving it a mess. If there is a more "desktop – like" approach needed, every main Window class is able to contain *Childwindows* with a dynamic z-order and the possibility to show more than one simultaneously.

To create a Window and show some content the following code should suffice:
The empty strings passed to the *Window* and the label *Widget* represent theme classes, defined in the theme file. As there are no themes defined, these strings are empty. Disko will give it a very basic look on its own, to ensure something is actually happening on the screen.

```
MMSRootWindow *win;
win = new MMSRootWindow(„“,“100%“,“100%“);

MMSLabel *lbl;
lbl = new MMSLabel(win, „“);
lbl->setFont(„./“,
       „DejaVuSansMono.ttf“,16);
lbl->setText(„Hello World“);

win->add(mylabel);
```

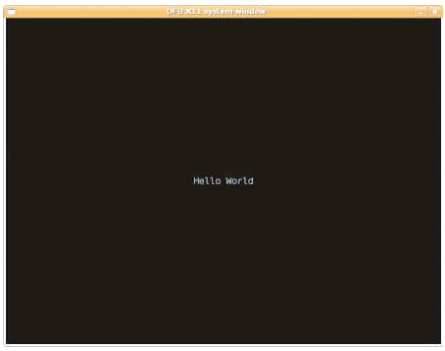
To see the results, the window needs to be shown on the screen:
To keep the application up and running,

```
win->show();
```

we have to set up an infinite loop:

```
while (1) sleep(1);
```

When every thing went fine, the result should be an X11 window having the text "Hello world" displayed centered within:



Not very spectacular but its the most basic application that can be done with Disko.

GUI possibilities

The Disko GUI engine is very powerful in its capabilities. The most powerful part is the dialog engine. It reads definitions for a Window and its contents from an XML-file, making the resulting dialog fully themeable and skinable without changing a single line of source code. Even the end-user should be capable of changing the layout to fit his needs. To achieve a clean MVC-approach, every widget that needs to have a programmed interaction can be named within the XML-file, enabling the application to utilize the dialog just by the symbolic names. Using the theme engine makes it also possible to create *Widget* or *Dialog* templates, thus bringing the opportunity to create meta-*Widgets* that can be used within the dialog files. This is an example from 23p using the different techniques showing an IPTV stream and application controls:

To fit into the embedded world, the XML files are only parsed once and are compiled into a binary format called *taff* optimized for the rendering engine to read. This is done to minimize the application startup time as well as to meet the requirements of limited storage capabilities.

Whats next?

This introduction covers only the very basic aspects of the Disko framework. The input models for remote control, touchscreen or the usage of plug-ins will be covered in the next article. Any questions can be posted on the Disko forums [9].

References

[1] http://www.xineproject.org
[2] http://www.gstreamer.net
[3] http://www.directfb.org
[4] http://www.x.org
[5] http://www.morphine.tv
[6] http://23p.diskohq.org
[7] http://www.diskohq.org
[8] http://www.diskohq.org/tutorials
[9] http://www.diskohq.org/forum

„Achtung Falschfahrer! Auf der A9 ....“

TMC-Staumeldungen auf dem NIOS II Embedded Evaluation Kit

Ludwig Schmidt <support@mixed-mode.de>

Einleitung

Jede halbe Stunde bringt der Radiosender die aktuellen Verkehrsnachrichten, nur leider nie, wenn man sie gerade hören möchte. Es fehlt die Meldung für zwischendurch.

Abhilfe verspricht der Traffic Message Channel (TMC), ein digitaler Staumeldeservice, der von nahezu allen öffentlich-rechtlichen sowie auch vielen privaten Radiosendern angeboten wird. Die Daten sind in den Radio Data System (RDS) Datenstrom des UKW Rundfunks eingebettet, der weit mehr als nur den aktuellen Sendernamen liefert.

RDS & TMC – der Aufbau

Die Übertragung der RDS-Daten findet parallel zum Audiosignal über das frequenzmodulierte Signal des Radiosenders statt. RDS-Daten sind zu Gruppen mit jeweils vier Blöcken organisiert. Jeder Block enthält 26 Bits, davon 16 Infobits und 10 Check- und Offsetbits. Eine RDS-Gruppe stellt also effektiv 64 Infobits zur Verfügung. Den grundsätzlichen Aufbau aller RDS-Gruppen zeigt Abbildung 1.

Jede dieser Gruppen hat einen bestimmten Group-Type, der zusätzlich in zwei Versionen A und B auftreten kann. Laut RDS-Spezifikation (DIN EN 62106) werden die Gruppen deshalb mit einer Hexzahl 0x00 – 0x15 und einem angehängtem A oder B bezeichnet. Dies ergibt prinzipiell 32 verschiedene Kombinationen, die für die verschiedenen Dienste eingesetzt werden. Diese Klassifizierung ist in Bit 15..12 „Group-Type“ und Bit 11 „Version“ des zweiten Blocks kodiert. Neben dem Group-Type enthält jede RDS-Group den PI- und cden PTY-Code. Der **P**rogram **I**dentification **C**ode identifiziert den Radiosender eindeutig und dient insbesondere der automatischen Suche nach Alternativfrequenzen. Mit dem Program Type Code können Radiosendung klassifiziert und vom Radioempfänger automatisch ausgewählt werden. So kann der Hörer etwa die Sparte „News“ wählen und das Radio sucht selbstständig nach

dem passendem Sender. Leider wird der PTY-Code oft nur statisch belegt und nicht dynamisch den Sendungen angepasst.

Die wichtigsten RDS-Gruppen sind 0 A&B für Sendernamen, 2 A&B für Radiotext, 4 A für Uhrzeit und Datum sowie 8 A für den Traffic Message Channel. Der Group-Type bestimmt aber nicht nur den Typ der Gruppe, sondern liefert auch die Vorschrift, wie die einzelnen Bits zu interpretieren sind. Der Radiotext wird zum Beispiel iterativ in ASCII-Klartext übertragen, die Uhrzeit in UTC-kodierter Form.

RDS stellt eine Datenrate von ~1,2 kBit/s zur Verfügung. Für TMC verbleiben davon ca. 60 Bit/s.
Eine typische Verkehrsmeldung könnte folgende Informationen beinhalten.
- Bezeichnung der Autobahn
- Bezeichnung des Abschnitts
- Betroffene Ausfahrten
- Eingetretenes Ereignis
- Voraussichtliche Gültigkeitsdauer der Störung

Um eine vollständige und sichere Übertragung der Daten zu gewährleisten wird auf redundante Datenübertragung gesetzt. Bei einem Payload von 60 Bit/s wird schnell klar dass eine Übertragung der Informationen in Klartext nicht möglich ist.

TMC wurde deshalb als listenbasiertes System entworfen (ISO 14819). Es sieht eine international gültige Event-List sowie eine national gepflegte Location-List vor. In der Event-List sind die Verkehrsereignisse kodiert, derzeit ca. 1600, die Location-List enthält die Beschreibung der bedeutenderen Straßen (in Deutschland ca. 45 000). Die Location-List für Deutschland sowie die Event-List mit deutscher Übersetzung

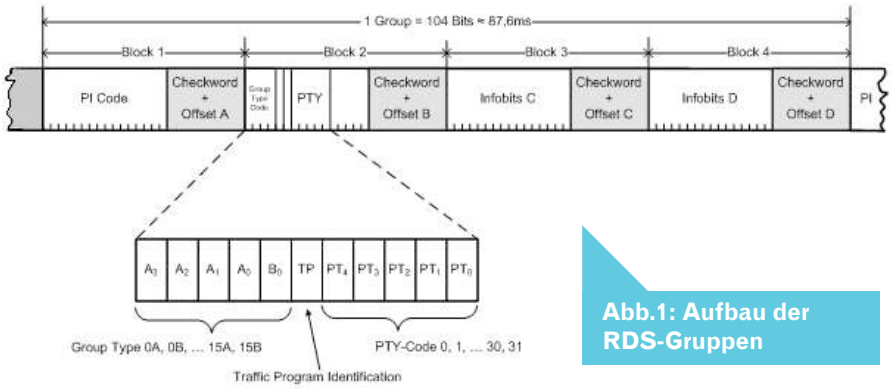


Abb.1: Aufbau der RDS-Gruppen

erhält man auf Anforderung von der Bundesanstalt für Straßenwesen [2]. Durch die Referenzierung der Listeneinträge kann so die Anzahl der zu übertragenden Bits drastisch reduziert werden. Im einfachsten Fall lässt sich eine Verkehrsmeldung als Single-Group User Message in nur einer RDS-Gruppe kodieren. Abbildung 2 zeigt eine RDS-Group, die die Daten einer TMC Single-Group User Message trägt. Neben den bereits erwähnten Codes für Event und Location und den Bits T und F in X4 und X3 besitzt die Message weitere Bestandteile zur näheren Beschreibung der Verkehrsstörung. Die Bits Y14 – Y11 dienen der Festlegung von Richtung und Ausbreitung der Störung. Der Diversion Indicator Y15 beschreibt, ob eine Umfahrung empfohlen wird oder nicht. Die Bits X2 – X0 Duration and Persistence informieren über das zeitliche Verhalten der Störung, etwa ob es sich um einen kurz anhaltenden Stau oder eine Tagesbaustelle handelt. Der Typ der TMC-Meldung, also ob es eine System oder User Message, eine Single- oder Multi-Group-Message ist, wird in den Bits X4 und X3 codiert.

Eine mögliche Meldung könnte sein: „A9, München, Richtung Nürnberg, zwischen Allershausen und Fürholzen, Unfall, 10km Stau“. Neben diesen für den User gedachten Informationen stehen für moderne TMC-Endgeräte zusätzliche Informationen in den Listen zur Verfügung. So können zum Beispiel Navigationsgeräte die geografischen Koordinaten aus der Location-List auswerten und diese in eine dynamische Routenplanung mit einfließen lassen.

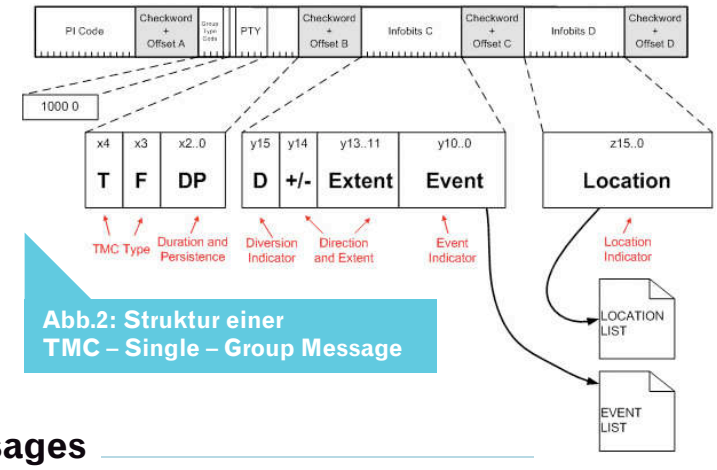


Abb.2: Struktur einer TMC – Single – Group Message

Noch mehr „Stau“ in Multi–Group Messages

Durch die Verwendung von Multi-Group Messages von bis zu fünf RDS-Gruppen können zusätzliche Informationen für eine Meldung, wie z.B. „Gefahr durch Schneeglätte“, ergänzt werden. Dabei werden zusätzlich zur ersten Gruppe, deren Struktur der einer Single-Group Message sehr ähnlich ist (vgl. **Abbildung 3**), sogenannte Subsequent Groups übertragen. Der Unterschied zwischen einer Single-Group und einer First-Group Message liegt lediglich bei den Bits X2 – X0 und Y15. X2 – X0 beinhaltet hier den Continuity Index, der eine Group mit einer bestimmten Multi-Group Message verbindet. Er wird mit der Aussendung einer jeden neuen Multi-Group Message inkrementiert und nach 110 zurück auf 000 gesetzt. Y15 ist der sogenannte First-Group-Indicator und ist nur bei der First-Group 1, bei allen Subsequent-Groups ist er auf 0 gesetzt.

Wie **Abbildung 4** zeigt, finden sich im zweiten Block einer Subsequent-Group die gleichen Merkmale wie in der First-Group wieder. Zusätzlich zum First-Group-Indicator Bit Y15, steht in

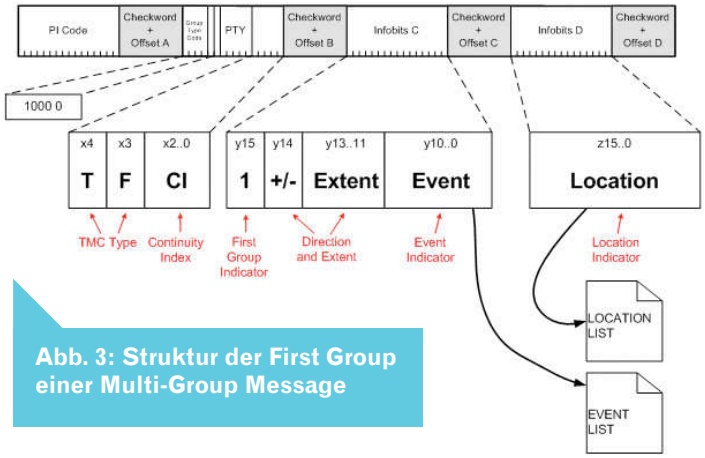


Abb. 3: Struktur der First Group einer Multi-Group Message

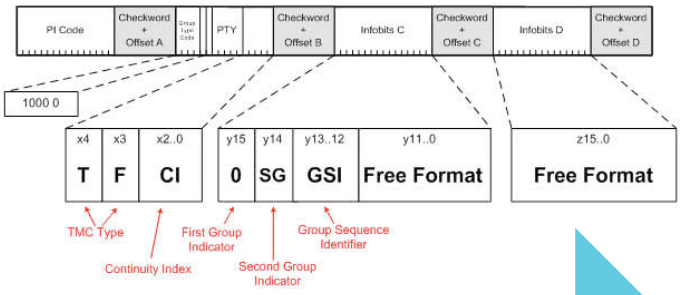


Abb.4: Struktur der Subsequent Groups einer Multi-Group Message

den Subsequent-Groups das Bit Y14 als Second-Group-Indicator bereit, der die zweite Group identifiziert. Mit den Bits Y13 und Y12 wird der Group Sequence Identifier übertragen, ein Zähler der mit jeder weiteren Group von N-2 abwärtszählt. N steht dabei für die Anzahl der Groups einer Multi-Group-Message. Neben diesem zur Zusammenführung der einzelnen Groups notwendigen Teils stehen pro nachfolgender Gruppe 28 Bit zur freien Verfügung. Über diese Free Format Bits lässt sich eine Sequenz von Label-Datafield-Paaren übertragen, wobei das Label die Länge und den Inhalt des darauf folgenden Datafields beschreibt.

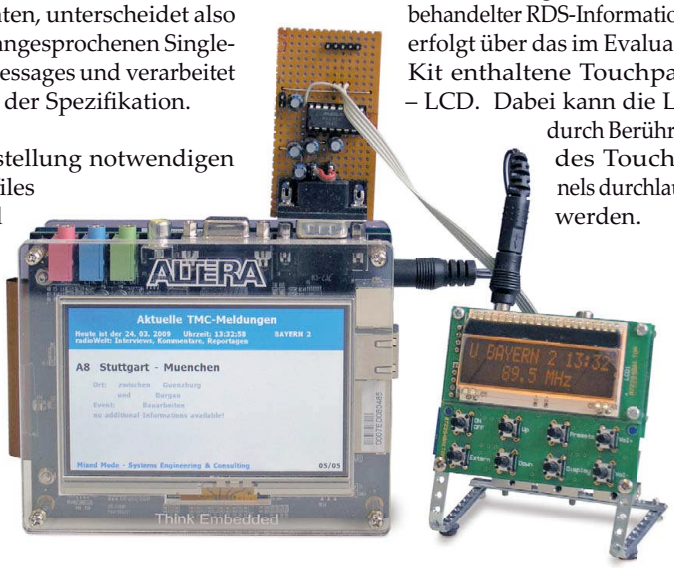
Die Realisierung

Für die Implementierung wurde ein NIOS II Embedded Evaluation Kit der Cyclone III Edition von ALTERA verwendet. Den Empfang des UKW-Signals übernimmt ein ELV Radiomodul des Typs RDS 100, dass den RDS-Datenstrom über eine RS232-Schnittstelle transparent ausgibt und über einen Pegelwandler mit dem ALTERA-Modul verbunden ist. Leider ist es mit der Original Firmware des Radios nicht möglich, diesen über die Schnittstelle zu steuern, was aber zum Beispiel für die automatische Suche nach alternativen Frequenzen wichtig wäre. Und wer sich dafür interessiert, sei auf das Radio-Projekt von Florian Kristen [3] verwiesen. Hier wurde eine alternative Firmware für das ELV-Radio entwickelt, mit der die Steuerung über RS232 möglich ist. Auf der Seite finden sich auch die Spezifikationen von RDS und TMC.

Die Dekodierung und Verarbeitung der Daten geschieht über einen im FPGA synthetisierten NIOS II Softcore. Über die UART-Schnittstelle werden die vom Radiomodul empfangenen RDS-Rohdaten übertragen. Das Programm trennt die RDS-Gruppen anhand eines vom Radiomodul eingefügten Sonderzeichens und dekodiert die RDS-Informationen. Dazu wird zunächst der Group-Type festgestellt und bei Vorhandensein von relevanten Informationen eine entsprechende Routine aufgerufen.

Neben TMC-Groups werden dabei auch RDS-Daten zu Sendernamen, Radiotext sowie Uhrzeit & Datum behandelt. Die Routine für TMC-Groups untersucht dann zunächst die Art der TMC-Daten, unterscheidet also zwischen den oben angesprochenen Single- und Multi-Group-Messages und verarbeitet diese entsprechend der Spezifikation.

Die zur Klartexterstellung notwendigen Listen sind als Textfiles im Flash abgelegt und werden über Dateizugriffsoperationen angesprochen. Ist eine TMC-Meldung fertig dekodiert, wird geprüft, ob es sich um eine neue Meldung oder um die Aktualisierung bzw. Wiederholung einer bereits vorhandenen Meldung handelt. Entsprechend dieser Überprüfung wird sie dann zu einer Liste hinzugefügt oder sie überschreibt eine vorhandene Meldung der Liste. Die Liste wird im Hintergrund periodisch durchlaufen, um die zeitliche Gültigkeit der enthaltenen Meldungen zu überprüfen und gegebenenfalls Meldungen zu löschen. Die Darstellung der in der Liste enthaltenen TMC-Meldungen und weiterer behandelter RDS-Informationen erfolgt über das im Evaluation Kit enthaltene Touchpanel – LCD. Dabei kann die Liste durch Berührung des Touchpanels durchlaufen werden.



Ausblick & Referenzen

Mit der fortschreitenden Digitalisierung des Rundfunks werden die TMC Meldungen zunehmend auch im DAB Datenstrom übertragen. Für weitere Releases ist geplant die RDS Daten aus einem DAB Signal zu dekodieren oder gar das DAB Signal mit dem NIOS oder direkt auf dem FPGA zu demodulieren.

[1]: http://de.wikipedia.org/wiki/Radio_Data_System
[2]: http://www.bast.de/cln_005/nn_42256/DE/Aufgaben/abteilung-v/referat-v2/Location-Code-List/location-code-list-start.html
[3]: https://wiki.tux-project.de/index.php/YetAnotherRDS100Software

TO OPEN YOUR SOURCE

JETZT VERÖFFENTLICHEN!

Dein Wissen - Dein Können - Dein Projekt

Dokumentiere deine Arbeit und veröffentliche im EP-JOURNAL!

DER BEITRAG FÜR DIE COMMUNITY

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

Werdet aktiv!

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

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

Regelmäßig lesen!

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

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

Sponsoren gesucht!

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

Bitte frei machen

Embedded Projects
Holzbachstraße 4
D-86152 Augsburg

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

Name / Firma

Straße / Hausnummer

PLZ / Ort

E-Mail / Telefon / Fax

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

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

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

SPEZIALISTEN GESUCHT

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

Du willst kreativ Entwicklungsprozesse mitgestalten -
werde Teil der Elektronikwelt.

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

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

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

IBV - ECHTZEIT- UND EMBEDDED GMBH & CO. KG

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

ibv
Realtime is BLUE