embedded projects
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Eine Open-Source Zeitschrift zum Mitmachen!

embedded ohne fesseln

[PROJECTS]

- Drahtlos steuerbares Sensorluftschiff
- Fahrzeugdiagnose für die Hosentasche
- Ein Mini-Mikroprozessor
- Ebook-Reader ohne Fußfesseln
- Typical behavior of the I2C Bus
- Hands-on Workshop Bestückung BGA-Bausteinen
- BTM-222 - Kommunikation mit Bluetooth

Einleitung

Ausgabe 02/2014

Embedded Projects Journal - Ausgabe No. 21

Einleitung

Was lange währt, wird endlich gut? in diesem Sinne freuen wir uns immer wieder wenn wir eine Ausgabe fertig haben. So ist es jetzt auch mit Ausgabe 21. Wir haben wieder von vielen Lesern spannende Artikel erhalten - Danke dafür! An dieser Stelle folgt wieder unser obligatorische Aufruf. Sendet uns Artikel, Themen, Ideen, Links, Vorschläge jederzeit gerne zu. Wir freuen uns auf jeden Hinweis!

Oft schlummern viele spannende Themen in den Schubladen. Oft ist es keine große Arbeit diese tollen Ideen in einem eigenen Artikel zu verewigen.

Gerne ermutige ich auch Schüler und Studenten dazu, so früh wie möglich mit dem Schreiben üben zu starten. Viel kann man nicht falsch machen. Nichts schreiben wäre falsch :-)

Benedikt Sauter und

das embeddded projects Team

Learn to Program

Learn to Program ist ein einfaches Spiel, dass - im Stil der Anfangsjahre des Computerzeitalters - das Programmieren mit nur vier Tasten erlaubt. Auf dem Spielbrett befinden sich alle nötigen Ein-/Ausgabeeinheiten, wie etwa eine LED-Ampel, ein Helligkeitssensor und ein Piezopiepser. Damit lassen sich verschiedene Programme entwickeln, wie zum Beispiel eine einfache Eieruhr, ein Wecker (der morgens wie ein Hahn Kikeriki ruft) oder auch einfache Reaktionsspiele.

Im Vordergrund steht, dass man sich nur mit dem Handbuch und Spielbrett - komplett abgekapselt von allen anderen technischen Geräten und Hilfsmitteln - in so ein einfaches Programmiersystem eindenken kann - so wie man es machen musste, als es mit dem Computer langsam los ging.

Mehr Infos: www.embedded-projects.net/learntoprogram

wawision.embedded-projects.net

waWision -
die Steuerzentrale für Ihre Firma

Verwal-tung

Plug & Play

Waren-eingang

Marke-ting

FiBu

Produk-tion

automa-tisches Lager

Online-Shops

EIN SYSTEM AUS EINER HAND

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

DEMOVERSION

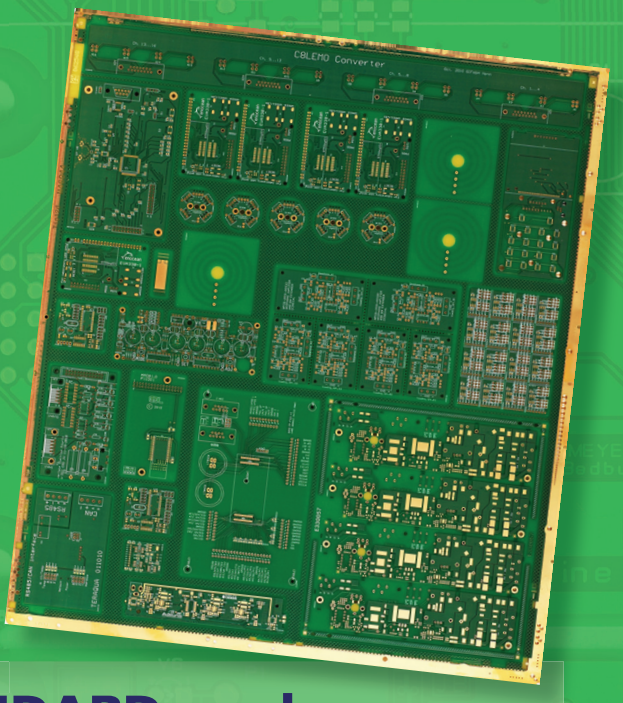
weitere Infos finden Sie auf unserer Internetseite

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

embedded projects GmbH
HARDWARE FOR PROJECTS

EURO CIRCUITS

Die europäische Referenz für PCB Prototypen und Kleinserien



Kostensenkung durch Online-Pooling

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

Zeitgewinn durch neue Online-Daten-Dienste

PCB Visualizer Sofort Design-Rule-Check

PCB Checker sofortige Anzeige von DRC Fehlern im Layout

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

PCB proto

spezieller Prototypen-Service für Entwickler, preiswert und schnell

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

STANDARD pool

die größte Auswahl an Eurocircuits Pooling Optionen

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

RF pool

alle Pooling-Vorzüge mit Hochfrequenz-Material

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

IMS pool

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

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

www.eurocircuits.de

PCB Visualizer PCB Checker PCB Configurator

The new EAGLE has landed!

Version 7

jetzt
erhältlich

CS
EAGLE

Weitere Informationen unter www.cadsoft.de

Drahtlos steuerbares Sensorluftschiff

Ein Zweitsemesterprojekt an der Siemens Technik Akademie Berlin

Gökler Özkan, Hagen Böttcher, Robby Kozok und Pascal Kahlert

Die Siemens Technik Akademie in Berlin bildet junge Leute im Studiengang Bachelor of Engineering aus. Im zweiten Semester findet neben den normalen Vorlesungseinheiten das Zweitsemesterprojekt statt. Die Studierenden arbeiten dort in 3er oder 4er Gruppen an Projekten, in denen es um praxisnahe Hard- und Softwareentwicklungen geht.

Einleitung

Im Rahmen unseres Zweitsemesterprojekts an der STA Berlin hatten wir die Aufgabe, ein Flugobjekt zu entwickeln, welches zum einen drahtlos steuerbar ist und zum anderen diverse Umgebungsdaten (z.B. Temperatur, Luftfeuchtigkeit usw.) misst, um diese dann zu speichern. Es war einem echten Projekt nachempfunden, so dass wir uns auch in die Rolle eines Kunden versetzten mussten. Zum Abschluss des Projektes erfolgte die Projektübergabe an den virtuellen Kunden sowie eine Abschlusspräsentation.

Wir sahen uns in diesem Projekt mit Problemen und Herausforderungen in verschiedenen Bereichen, wie Projektmanagement, Elektronik, Software Development und Mechanik, konfrontiert.

Um diese Herausforderung zu meistern, haben wir uns in unserem ersten Team-Meeting dazu entschlossen, uns in 4 Kerngebiete aufzuteilen. Daraus ergab sich folgende Einteilung:

- Projektmanagement – Gökler Özkan
- Elektronik – Robby Kozok
- Mechanik – Hagen Böttcher
- Software Development – Pascal Kahlert

Projektmanagement

Gökler Özkan: Ich übernahm die Rolle des Projektleiters während des gesamten Zweitsemesterprojekts. Zu meinen Aufgaben gehörten neben der Planung, Verwaltung, Organisation und Leitung auch Bestellungen und Kostenrechnung. Zudem war ich zuständig für die Kommunikation mit dem Kunden. Dabei war mein Ziel als Projektleiter, alle Kundenanforderungen unter den gegebenen Bedingungen so gut wie möglich zu erfüllen und meinem Projektteam einen reibungslosen und angenehmen Projektablauf zu gewährleisten. Auf dem Weg zum Ziel stellten sich wie üblich einige Probleme.

Das erste Problem bestand darin, eine geeignete Gruppenaufteilung zu finden und den Gruppenmitgliedern dementsprechend Aufgabenbereiche zuzuordnen. Dies stellte sich als relativ einfach heraus, zumal jeder ein Spezialist in je einem projektrelevanten Thema war.

Anschließend folgte ein interner Vertrag innerhalb des Projektteams, der die Rechte und Regeln jedes einzelnen Teammitglieds in Bezug auf das Projekt festlegte.

Eine Gruppenaufteilung und ein interner Gruppenvertrag sind jedoch nicht genug, um solch ein Projekt zu bearbeiten. Was fehlt, sind genaue Anforderungen des Kunden an das Endprodukt. Diese erfolgten in Form von wöchentlichen Kundengesprächen. Von Woche zu Woche wurden die Anforderungen des Kunden konkretisiert und entsprechend der gegebenen Möglichkeiten des Projektteams angepasst. Am Ende dann schließlich wurde das Lasten- und Pflichtenheft vom Kunden und mir unterschrieben. Dabei kam es manchmal zu Missverständnissen zwischen dem Kunden und uns, denn wir gingen mit unserer noch sehr geringen Erfahrung davon aus, dass der Kunde feste Vorstellungen von seinem gewünschten Produkt hat und diese auch nicht von Woche zu Woche ändern würde. Diese Missverständnisse konnten wir dann durch die Erstellung von Protokollen während des Kundengespräches gezielt auf ein Minimum verringern. Dies half auch bei der finalen Erstellung des Vertrags zwischen dem Kunden und unserer Gruppe, der ohne jegliche Missverständnisse oder Änderungen unterschrieben wurde.

Abb. 1: Bild des fertigen Luftschiffs

Nach der Erstellung und Unterzeichnung des Lasten- und Pflichtenhefts folgte eine genaue Zeitplanung und Arbeitsaufteilung in Form eines Meilenstein- und Projektstrukturplans. Das Problem hierbei bestand darin, das Projekt so vorrausschauend und genau wie möglich zeitlich in einzelne Arbeitselemente zu unterteilen, dass alle Anforderungen an das Produkt bis zum Übergabetermin erfüllt werden konnten. Dies stellte sich nicht als allzu einfach dar, da wir nicht wussten, wie lange die Bestellungen von Teilen dauern würde. Am Ende führte dies dann

auch dazu, dass wir die abschließende Testphase falsch eingeplant hatten und so in unsere Herbstferien verschieben mussten.

Nach der Fertigstellung des Projektstrukturplans folgte die Erstellung einer Stückliste mit allen benötigten Materialien für die Entwicklung und den Aufbau des Flugobjekts. Hierbei haben wir versucht, so gut wie möglich vorauszuplanen und erst einmal geschaut, was bereits an Standardbauteilen und -komponenten zur Verfügung stand.

Für unsere erste Bestellung von Bauteilen musste eine standardisierte Bestellliste, die speziell für dieses Projekt entwickelt und für jeden einzelnen Anbieter separat erstellt wurde, verwendet werden. Das Budget von 250 € durfte nicht überschritten werden, was sich manchmal als sehr schwer herausstellte. Die Überlegung, ob wir eine Lochrasterplatine oder eine gefertigte Leiterplatte verwenden, wurde nach längerer Abwägung mit Robby, unserem Elektroniker, zugunsten einer doppelseitigen Leiterplatte entschieden. Dies, obwohl wir damit sehr an die Grenzen unseres Budgets stießen.

Elektronik

Robby Kozok: Ich übernahm in diesem Projekt die Funktion des Spezialisten für die Elektronik. Zu meinen Aufgaben gehörten alle elektrotechnischen Themen wie die Stromlaufplanerstellung, Schaltungserprobung, die Erstellung eines Leiterplattenlayouts sowie Bestückung der Leiterplatte mit anschließender Testphase.

Da mir die Kundenanforderung viel kreativen Freiraum ließ, richtete sich mein Hauptaugenmerk bei der Stromlaufplanerstellung auf die Funktionalität. Dennoch war es notwendig, ständige Rücksprache im Team bzgl. der Mechanik, Software, aber auch über Änderungswünsche mit dem Kunden zu halten.

Nach einer Aufstellung der benötigten Funktionseinheiten entstand das Blockdiagramm in Abbildung 2. Im Zuge dieser Recherche stellte ich die elektronischen Komponenten für die Schaltung zusammen. Hierbei flossen auch Erfahrungen unserer Dozenten mit ein. So etwa entschloss ich mich, das Bluetooth Modul BTM-222 für die drahtlose Datenübertragung und für die Kommunikation zwischen den Sensoren den I2C-Bus zu verwenden. Als Stromversorgung wurde ein 2 Zellen Lithium Polymer Akku mit einer Nennspannung von 7,4 V und 900 mAh Kapazität eingesetzt. Da das Bluetooth Modul mit einer Spannung von 3,3 V arbeitet und ich nur einen Spannungslevel für die gesamte Schaltung nutzen wollte, wurden alle Komponenten auf Basis von 3,3 V mittels Schaltregler (L5973) versorgt.

Nachdem alle Bauteile vorhanden waren, errechnete ich einen maximalen Stromverbrauch der Schaltung von 1,6 A. Mit diesen Informationen konnte ich nun eine geeignete Spannungsversorgung heraussuchen. Hierbei gab es zwei Alternativen: Entweder man verwendet einen Linear- oder Schaltregler. Aufgrund der besseren Effizienz in diesem Fall (7,4 V auf 3,3 V) eines Schaltreglers entschied ich mich für solchen. In Abbildung 3 ist die Spannungsversorgung abgebildet.

An VCC wird über eine Sicherung und einen Schalter die Spannung der Batterie angelegt. Für die geforderte 200µH wurden zwei 100µH-Induktivitäten verwendet, da keine größere Induktivität mit 2A Stromfestigkeit verfügbar war.

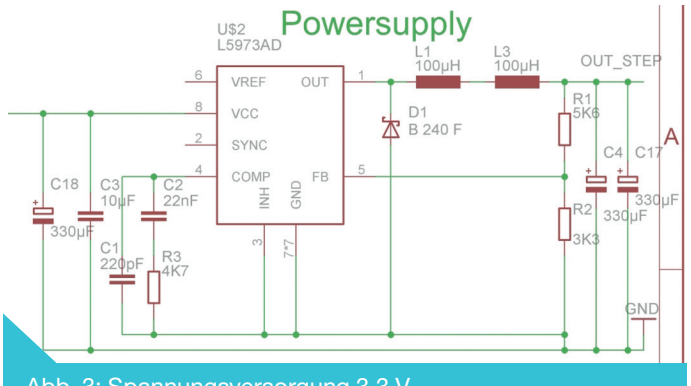


Abb. 3: Spannungsversorgung 3,3 V

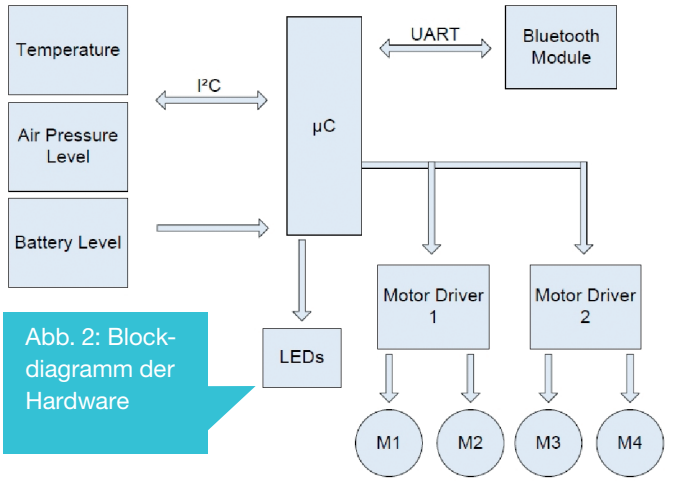


Abb. 2: Blockdiagramm der Hardware

Es stellte sich heraus, dass die Spannungsversorgung einige Probleme machte. Zum Testen war diese auf einer Lochrasterplatine gelötet, und unter Last brach die Spannung von 3,3 V auf 3 V zusammen. Die Überarbeitung der Werte für die Induktivität und der Kapazitäten brachte nur mäßigen Erfolg. Später stellte sich heraus, dass es an der wild aufgebauten Schaltung lag, denn auf der fertigen Leiterplatte blieb die Spannung stabil bei 3,28 – 3,30 V.

In Absprache mit unserem Programmierer Pascal Kahlert haben wir uns für einen Mikrokontroller von Atmel (Atmega328) entschieden. Um diesen einfach in die Schaltung zu implementieren, nutzte ich ein Applikationsboard von Anvilex (Abbildung 4). Dieses Board erwies sich als sehr zuverlässig, preisgünstig und modular.

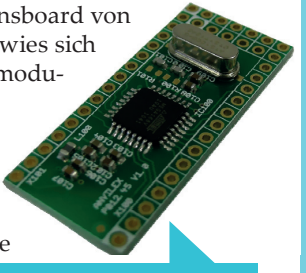


Abb. 4: Atmega328

Um unsere Motoren (Mikro Glockenanker Motoren) anzusteuern, wurden Schrittmotortreiber vom Typ BA6845FS eingesetzt, da diese über eine integrierte Logik verfügen und es so sehr einfach war, die Motoren anzusteuern. Für jeden Motor gab es zwei digitale Eingänge. Über einen konnte die Drehrichtung gesteuert werden, an den anderen wurde mittels PWM (Pulsweitenmodulation) die Drehzahl eingestellt. In Abbildung 5 und 6 sind die Motortreiber zu sehen.

Das für die Bluetooth-Kommunikation verwendete BTM-222 Modul erwies sich als sehr stabil und einfach zu handhaben. Als externe Beschaltung waren lediglich 3 Status LEDs, ein Reset-Taster mit Pullup-Widerstand sowie eine Antenne notwendig. Das Model überzeugte mit seiner hohen Reichweite, dafür mussten wir aber auch einen hohen Stromverbrauch von rund 100 mA hinnehmen. Abbildung 7 zeigt das Modul auf der Leiterplatte.

Links davon (Abbildung 7) ist außerdem der Temperatursensor DS1621 zu sehen. Dieser arbeitet momentan in 1 Grad-Schritten, 0.5 Grad-Schritte können aber softwareseitig noch implemen-

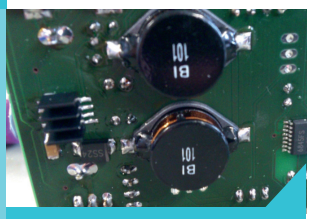


Abb. 6: Treiberschaltung

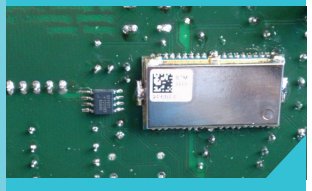


Abb. 7: BTM-222 und Temperatur Sensor

tiert werden. Die Positionierung des Sensors erwies sich aufgrund der Erwärmung des Bluetooth-Moduls als unpassend und sollte in der nächsten Version geändert werden.

Zur automatischen Höhenhaltung haben wir versucht, den Sensor BMP085 zu verwenden, d.h. mit diesem den Luftdruck zu messen und mit den gewonnenen Daten die Motoren anzusteuern. Dies gelang aber nur mit mäßigem Erfolg, und in der nächsten Version würden wir dafür einen Beschleunigungssensor verwenden. Außerdem überwachten wir über den ADC-Eingang des Mikrokontrollers die Batteriespannung. Pascal implementierte eine Füllstandsanzeige der Batterie auf dem Smartphone.

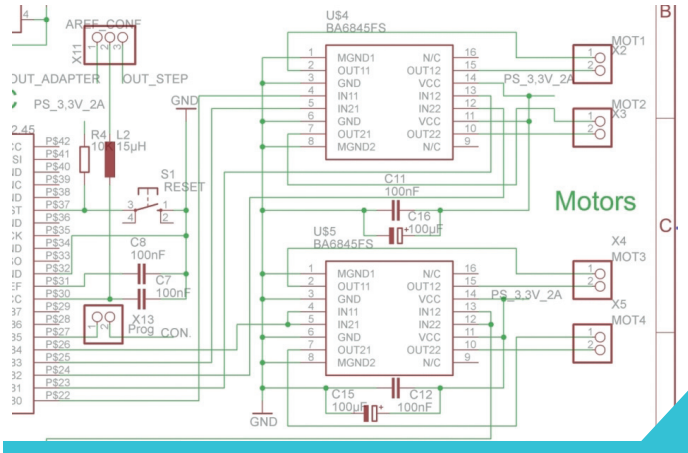


Abb. 5: Treiberschaltung für die Motoren

Nachdem die Schaltung in Eagle fertiggestellt war, begann ich mit der Layouterstellung. Viele der verwendeten Komponenten waren nicht in den Bibliotheken verfügbar, diese mussten daher selber erstellt werden. Insgesamt zog sich dieser Prozess etwa über einen Monat hin (wir mussten in der Zwischenzeit ja auch noch studieren!). Danach konnte die Leiterplatte bei einem Hersteller geordert werden. Das Leiterplattenlayout ist in Abbildung 8 zu sehen.

Nachdem die Leiterplatte geliefert wurde, konnte die Bestückung abschnittsweise erfolgen. Nach einem Tag war die Leiterplatte bestückt und getestet, alles funktionierte wie geplant, und unser Mechanikexperte Hagen Böttcher konnte beginnen, die Gondel zu bauen.

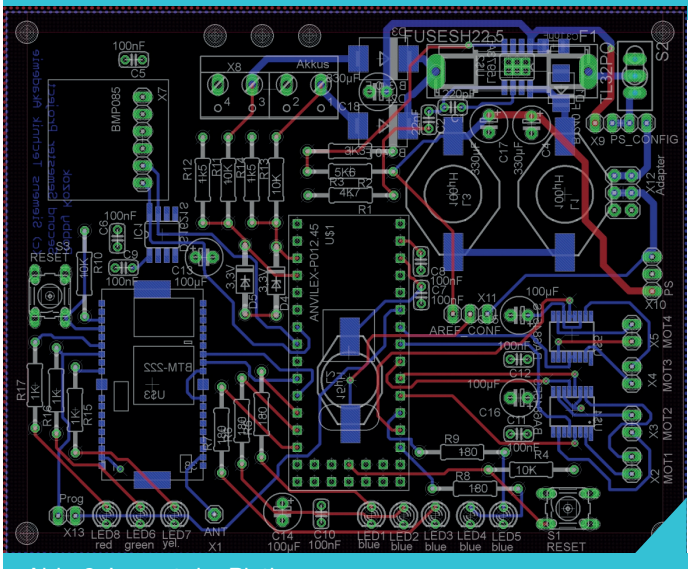


Abb. 8: Layout der Platine

Mechanik

Hagen Böttcher: Meine Kernkompetenz in diesem Projekt war die Mechanik, dazu gehörte die Konstruktion der Gondel sowie der Befestigung der Gondel an den Blimps (Prallluftthüllen).

Das Hauptproblem war das Gewicht, welches durch die Blimps zu befördern war. Aus diesem Grund musste abgewartet werden, bis die Leiterplatte fertig bestückt und getestet war. Im Vorfeld konnte ich nur mit einem grob geschätzten Gewicht meine Recherche nach geeigneten Hüllen tätigen. In Folge dieser Recherche ergab es sich, dass praktisch keine Hülle auf dem Markt verfügbar war, um das zu befördernde Gewicht zu heben. Es waren lediglich kleinere Hüllen erhältlich. Dieser Punkt wurde Hauptthema in einer unserer Teammeetings, und wir haben darüber diskutiert, ob es möglich ist, selber eine Hülle zu bauen oder ob wir mehrere Hüllen zu einer verbinden sollten.

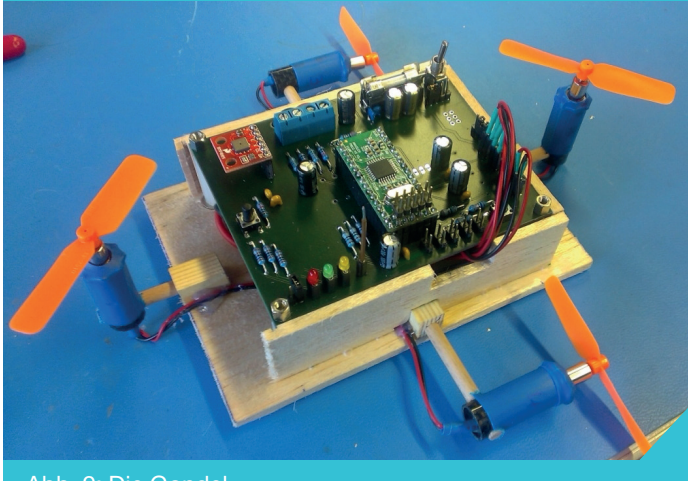


Abb. 9: Die Gondel

Wir haben uns für die zweite Variante entschieden, da es unserer Meinung nach ohne Erfahrung nicht möglich ist, eine dichte Hülle selber zu bauen. Nach einer weiteren Recherche wurden Blimps der Firma Good Year aus den USA bestellt. Während wir auf die Lieferung warteten, habe ich mir über die Gondel Gedanken gemacht. Diese sollte leicht sein, aber dennoch zuverlässig die Komponenten wie Leiterplatte und Motoren am Blimp befestigen. Der erste Prototyp der Gondel, eine Konstruktion aus Draht, war jedoch zum einen zu schwer, und zum anderen konnte die Leiterplatte darin nicht zuverlässig fixiert werden. Daraufhin entschied ich mich für ein anderes Material, Balsaholz, welches auch im Modellbau häufig genutzt wird. Nach zwei Tagen war die neue Gondel fertig, und die Leiterplatte konnte darauf montiert werden. In Abbildung 9 ist die Gondel mit Leiterplatte und Motoren zu sehen. Unter der Leiterplatte habe ich auch eine Batteriebefestigung gefertigt. Um die Gondel zu schließen, wurde noch ein Einschub mit Feder und Nutsystem realisiert. Anschließend wurde noch alles weiß lackiert.

Software Development

Pascal Kahlert: Ich habe im Projekt die Kernkompetenz des Software Engineerings übernommen. Das waren im Grunde die Programmierung des Mikrocontrollers (µC) in ANSI–C und die Programmierung meines Smartphones in Java. Angefangen hat allerdings alles mit Planung, wie die Kommunikation funktionieren könnte. Es standen mehrere Optionen im Raum:

- WLAN
- Funk (433 oder 868 MHz)
- Bluetooth (2,4 GHz, SPP)

Abb. 10: GUI der App

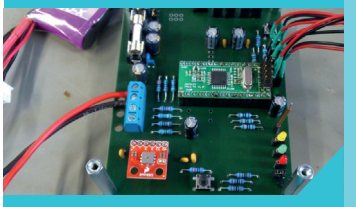


Abb. 11: Mainboard

WLAN mussten wir aus Gewichts- und Energiegründen streichen. Eine Funkverbindung war nicht geeignet, da wir eine Verbindung entweder über PC oder Smartphone realisieren wollten und das Smartphone die 433/868 MHz Funksysteme nicht unterstützt. Deshalb haben wir uns für Bluetooth entschieden; gute Module waren günstig zu kaufen und lieferten eine Reichweite von 100 m.

Wir entschieden uns für eine Steuerung über ein Smartphone. Wir sahen es als Herausforderung, mit dieser modernen Technik zu arbeiten. Vieles sprach außerdem dafür, dass Bluetooth sehr einfach zu realisieren war. Die größten Schwierigkeiten beim Entwerfen der Smartphone App am Anfang war die stabile und übertragungssichere Bluetooth-Kommunikation. Ausfälle hätten später die Steuerung empfindlich stören können.

Die grundsätzliche Strategie war, nicht einzelne Bytes zu setzen, sondern immer 5 Bytes hintereinander zu senden und das ganze gleich dreifach. Damit dabei immer noch klar unterschieden werden konnte, wo ein Byte beginnt und ein anderes endet, habe ich jeweils Stopp- und Start-Bytes benutzt, welches Zahlen waren, die nicht durch das Programm entstehen konnten. Damit war die Übertragung so stabil, dass uns keine Fehlübertragungen aufgefallen sind. Das nächste Problem war der grundsätzliche Umgang mit Threads und den GUI-Elementen auf dem Android Smartphone. Wie auch die Swing GUI-Elemente, sind die GUI-Elemente in Android auch nicht Thread sicher, allerdings stellt ja Android dafür sogenannte Handler zur Verfügung, die es erlauben, Änderungen an GUI-Elementen aus anderen Threads durchzuführen. Nachdem der Großteil der Hardware geliefert und aufgebaut war, fing ich an, den Atmega328 zu programmieren. Dabei musste zunächst die Übertragungsgeschwindigkeit von 19200 Baud angepasst werden.

Abb. 12: Arbeit im Elektroniklabor

Das nächste zunächst sehr komplexe Thema war der I²C Bus, da wir diese Art der Kommunikation noch nie benutzt hatten. Allerdings stellte sich sehr schnell heraus, dass es im Grunde immer dieselbe Herangehensweise war und wir die Arbeit somit nur für einen Sensor hatten. Daher konnten wir ohne Probleme die auf andere Sensoren übertragen.

Bis zu diesem Punkt war allerdings noch nicht klar, ob das Zusammenspiel aus Smartphone, Mikrocontroller und Elektronik funktionieren würde. Daher haben wir zunächst die Motoren an Pappboxen befestigt, um zu sehen, ob die Steuerung so wie vorgesehen funktioniert. Die Sensoren wurden auf Steckbretter gesteckt und mit Kältespray oder Heißluftföhn bearbeitet, um eine Temperaturveränderung sichtbar zu machen.

Als wir uns sicher waren, dass alles funktioniert, haben wir das erste Mal Helium in unsere Hüllen gefüllt und unseren Blimp fliegen lassen. Nach den ersten Versuchen stellte sich heraus, dass die Steuerung funktioniert, aber zu kompliziert war, um unseren trägen Blimp ordentlich steuern zu können. Daher war die Idee, das Gyroskop des Smartphones zu nutzen und alle horizontalen Richtungsveränderungen sowie die Motorleistung über den Grad der Auslenkung des Handys zu lenken. Gleichzeitig sollten die Motoren für die vertikalen Richtungsänderungen weiterhin über das Handydisplay laufen, so dass wir den Blimp gleichzeitig hoch bzw. runter und links/rechts/vorwärts/rückwärts fliegen lassen konnten. Das erlaubte uns, nach einer kleinen Eingewöhnung eine sichere Bewegung, selbst auf kleinem Raum.

Mögliche Verbesserungen

Die Erfahrungen aus diesem Projekt ermöglichen uns dies weiter zu verbessern. Im Einzelnen wären das folgende Punkte:

Um Platz und Gewicht zu sparen, könnte man alle Bauteile in SMD ausführen und auch den µC direkt auf das Mainboard platzieren. Da der Stepdown-Converter funktioniert, kann auf einige Steckverbinder und Klemmen verzichtet werden. Alle Anschlussklemmen verkleinern und div. Teile, die nicht notwendig sind, entfernen, um Gewicht zu sparen. Ersetzen der zwei in Reihe geschalteten Spulen durch eine Spule. Der BMP085 Luftdrucksensor gegen Beschleunigungssensor tauschen und am besten alles mit TWI (I2C) anbinden. Um die Temperatur richtig zu messen, würde eine Umplatzierung des Temperatursensor DS1621S hilfreich sein, um Wärmestau durch den BTM-222 zu vermeiden. Um Gewicht zu sparen, neues, leichteres Material für die Gondel verwenden, z.B. 3D-Druck oder dünner Kunststoff.

Durch Einbau eines Beschleunigungssensors könnte man durch ein intelligenteres Design der App eine bessere Steuerbarkeit sowie etwas mehr Funktionalität entstehen lassen. Beispiel: Messwerte direkt über die App per Email verschicken.

Fahrzeugdiagnose für die Hosentasche

Steffen Köhler <contact@oobd.org>

Hardware zur professionellen Fahrzeugdiagnose ist groß, unhandlich und teuer, und man muss ein Diplom haben, um es bedienen zu können? OOBD zeigt: Es geht auch einfach!

Die Vorgeschichte

Vor ein paar Jahren saßen ein paar Entwicklungsingenieure eines großen Automobilherstellers zusammen und fragten sich, warum sie für ihre Fahrzeugdiagnose immer noch mit Laptop, CAN-Blackbox und einer Kabelrolle durchs Werk irren mussten, wo es doch scheinbar in jedem Baumarkt kleine, handliche Fahrzeug-Diagnosetester zu kaufen gibt. Aber für Ingenieure sind Probleme halt dafür da, gelöst zu werden - und so entstand OOBD.

Was ist überhaupt Fahrzeugdiagnose?

Fahrzeugdiagnose macht man heute über den in jedem Auto vorgeschriebenen OBD-II Stecker (OBD= OnBoard Diagnostics). In diesem Stecker finden sich, je nach Automarke und Modelljahr, verschiedene serielle Anschlüsse (Busse), die intern mit den Fahrzeugmodulen verbunden sind. Die Flöte der möglichen Busse reicht von 1/2-Draht Leitungen (K(L)-Line ISO-9141, VW-Bus) über symmetrisch serielle Leitungen (V-/PWM SAE-J1850) bis hin zum richtigen CAN-Bus (SAE-J2284). Ein Teil der Pinbelegung des OBD-II Steckers ist genormt, und die Hersteller lieben es, die nicht genormten freien Pins mit eigenen Erweiterungen zu bepflastern.

Diese ganzen Busse sind von ihrer Hardware her aber alle nicht direkt kompatibel zu dem, was man normalerweise an einen PC anschließen kann, darum ist immer eine zusätzliche (intelligente) Hardware vonnöten, die die Signalpegel und z.T. auch das gesamte auf dem Bus verwendete Datenprotokoll so umsetzt, dass ein PC mit dem Bus die Daten austauschen kann. So ein Hardware-Wandler wird normalerweise als OBD-Dongle oder OBD-Box und die ganze Einheit aus Dongle, PC und Diagnosesoftware als Diagnose-Tester (OBD-II Tester) bezeichnet. Diese Bezeichnungen Dongle und Tester werden wir auch im weiteren Text so wiederfinden.

Auch wenn man als Zuschauer in der Werkstatt den Eindruck bekommt, die Daten würden von selber aus der OBD-II Schnittstelle herauspurzeln, so ist es in Wirklichkeit dann doch anders. Fahrzeugdiagnose ist statt dessen ein streng reglementiertes Frage- und Antwort-Spiel (Request <=> Response): Im Detail sieht das so aus, dass der Tester an die Moduladresse des gewünschten Steuergerätes (ECU - Electronic Control Unit) eine Bytefolge schickt, die das angesprochene Modul dann als Anfrage interpretiert und seinerseits daraufhin die Antwort an die Adresse des Testers schickt, der die Antwort entsprechend auswertet.

Wie man daran schon sehen kann, sind dafür schon einige Detailkenntnisse notwendig: Man muss die Modul-Adresse kennen, wissen, wie die gewünschte Frage in Byteform zu formulieren ist und wie man die empfangenen Bytes wieder richtig in eine Klartext-Antwort umrechnen bzw. darstellen muss. Und auch hier hat jeder Automobilhersteller wieder seine eigene Philosophie, aber glücklicherweise ist wenigstens das Protokoll standardisiert, mit dem die Bytes über den Bus huschen, und genau da setzt OOBD an. OOBD steht übrigens für OpenOnBoardDiagnostics wobei das Open für Opensource steht. Aus diesem Grund befinden sich auch alle Hard-und Softwaresourcen des OOBD-Projektes auf OOBD.org. [1]

Welche OBD-Tester gibt es heute?

Betrachtet man die heute auf dem Markt verfügbaren Diagnose- Tester, dann findet man letztlich zwei Kategorien:

Einmal die gefühlt tausend Derivate des Urvaters des Diagnosesteckers, dem ELM 327. Alle diese Geräte sind immer noch Befehlskompatibel zum alten 327, und haben so auch kaum eine Chance, dessen Einschränkungen zu entkommen: Die Adressen der erreichbaren Module sind hart verdrahtet, und der viel zu kleine interne Speicher reicht nicht aus, um moderne Protokolle in voller Telegrammlänge abwickeln zu können.

Einige dieser Geräte versuchen noch, zumindest einen Low-Level CAN Bus Modus zur Verfügung zu stellen, bei dem einzelne Roh-CAN Datenframes gesendet und empfangen werden können. Das hat aber bei Paketen, die nicht mehr in einen einzelnen CAN-Frame passen, die unangenehme Auswirkung, dass die dann notwendige Protokoll-Abwicklung in Echtzeit vom Diagnose-Tester selber durchgeführt werden muss. Das setzt aber eine schnelle, sichere Verbindung zwischen Dongle und Tester und einen schnellen Tester voraus, wodurch aber wackelige Bluetooth-Verbindungen und langsame Handys als Tester ausfallen.

Am oberen Ende der Preis- und Leistungsskala finden sich dann die Geräte der professionellen Hersteller. Diese Geräte können eigentlich fast alles, hängen aber u.a. aus Geschwindigkeitsgründen oftmals am (unerwünschten) Kabel, sind meistens PC- basiert und teuer, nicht zuletzt auch, weil fahrzeugspezifische Internas mit zusätzlichen Lizenzkosten mit im Produkt enthalten sind. Diese Systeme haben dann meistens entweder viel zu viele Stellschräubchen oder reglementieren den Anwender durch fest verdrahtete Frage- und Antwort-Menüs.

Der OOBD-Tester - was macht ihn anders?

Die ganzen oben genannten "Wenn und aber" waren die Eingangsgrößen, als das OOBD-Projekt startete. OOBD versucht nun, die heutige Lücke zwischen gut und billig mit einem neuen Konzept auszufüllen. Unsere selbst gestellten Anforderungen waren dabei:

- Es soll einfach sein
- Es soll an alle Anforderungen anpassbar sein
- Das gesamte System soll in die Hosentasche passen, um es immer dabei haben zu können
- Es soll über Bluetooth funktionieren, weil Handys nun mal keinen Kabelanschluss haben
- Es soll den kompletten Adressbereich und Telegrammlänge des ISO-Protokolls abdecken, um mit jedem Modul im Fahrzeug arbeiten zu können
- Die gesamte Echtzeit-Abwicklung soll im Dongle passieren, um so problemlos mit langsamen Bluetooth-Verbindungen und Handys zu funktionieren
- Es soll herstellerspezifische Besonderheiten bedienen können, ohne diese aber aus lizenzrechtlichen Gründen von vornherein mitzuschleppen.
- Es soll zwei CAN-Busse unterstützen

Die OOBD-Hardware

Zuerst einmal brauchten wir einen Prozessor, der genügend Speicher bietet, CAN unterstützt und einfach beschaffbar und programmierbar ist. Dies fanden wir im STM32-Cortex M3. Der Prozessor wurde dann noch umgeben mit Bustreibern, Spannungsregler, LEDs und Buzzer und in ein passendes Gehäuse gepresst. Damit war der Hosentaschen-Formfaktor schon mal gegeben.

Dabei wurde auch darauf geachtet, dass die Hardware die rauen Bedingungen im Werkstatt- Alltag überstehen kann: Der OBD-II Stecker hat feste vergossene Kontakte, die Platinenaufhängung ist abgefedert, so daß auch ein Fall aus 1 Meter auf Beton auch mehrmals überstanden wird. Die Platine und der OBD-II Stecker sind getrennt, um entweder den OBD-Stecker an eigene Belegungen anpassen zu können oder die Hauptplatine in eigene Systeme einzustöpseln.

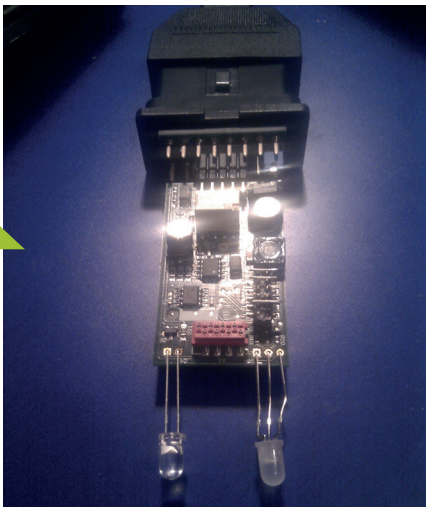


Abb. 1: Die OOBD-Hardware

Die OOBD-Firmware

Die Firmware ist viel komplexer, als man das vielleicht bei einem einfachen Stand-Alone-Controller vermuten würde. Statt sich lange mit proprietärem Code-Gebastel rumzuquälen, wurde dem Prozessor gleich ein komplettes Multitasking-Betriebssystem spendiert, nämlich das bekannte FreeRTOS [2]. Dies ermöglichte eine saubere Trennung der einzelnen Funktionsgruppen.

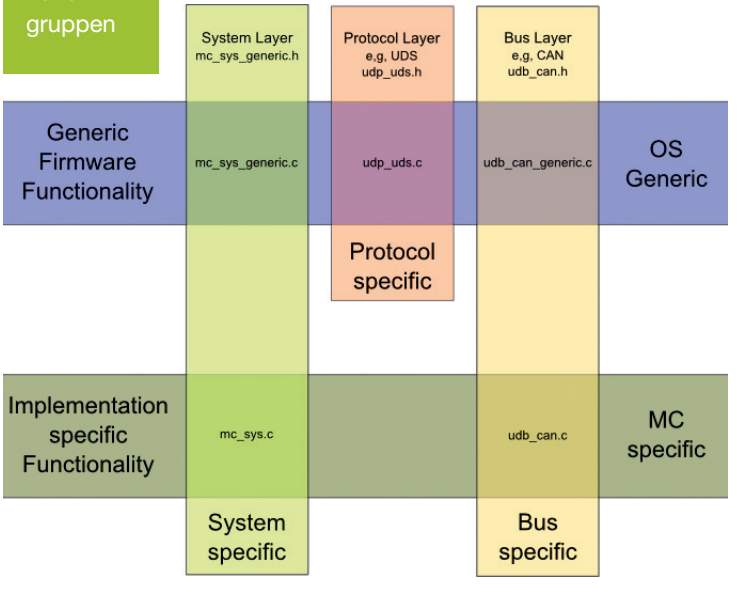


Abb. 2: Saubere Trennung von Funktions-gruppen

Und diese Trennung wiederum bringt weitere Vorteile mit sich:

- Die hardwarespezifische Programmierung befindet sich in eigenen Sourcefiles für's Startup und die I/O-Konfiguration (CAN-Bus, UART für BT-Module, LED-Ansteuerung, L/K-Line, Buzzer)
- FreeRTOS ist für eine Vielzahl von Prozessoren verfügbar. Will man irgendwann mal OOBD auf einen anderen Prozessor portieren, braucht man nur die hardwarespezifischen Sourcen anzupassen, der Rest bleibt unverändert. Dies wurde schon während der Entwicklung ausgiebig genutzt, indem der OOBD-Kernel mit FreeRTOS als Emulator unter Linux läuft und man alle Prozesse direkt am PC debuggen kann, ohne nach jeder Änderung erst einen Controller flashen zu müssen.
- Weiterhin ist das OOBD-OS von vornherein dafür designed, weitere Protokolle und Busse zu unterstützen. Dazu muss nur ein neues Source-file mit der neuen Funktion in den Build-Prozess aufgenommen werden, die notwendigen Befehle zur Protokoll- oder Bus-Umschaltung sind bereits vorhanden und werden auch schon für verschiedene Protokolle benutzt.

Die Befehls-Syntax

Die Befehlssyntax wirkt im Vergleich zu dem, was man vom ELM327 kennt, ausgesprochen spartanisch, aber das täuscht: OOBD kennt nämlich nur zwei Arten von Eingaben, und nur eine davon ist überhaupt ein Befehl.

Alles, was mit "p" anfängt, gilt als Befehl (p steht für Parameter); alles andere wird als hexadezimale Dateneingabe interpretiert und vom jeweils gerade laufenden Protokoll direkt verarbeitet, d.h. es sind nur die Zeichen 0-9, A-F erlaubt. Blanks sind ebenfalls erlaubt und abgeschlossen wird eine Eingabe mit <CR>; alle anderen Zeichen und Eingaben sind ungültig und die Eingabezeile wird ignoriert.

Die P-Befehle dagegen bestehen auch nur aus mehreren Integer Werten, getrennt durch Blanks, die entweder dezimal oder mit $ am Anfang hexadezimal geschrieben werden. Die erste Zahl gibt immer an, für welchen Funktionsblock der Befehl gedacht ist, so gibt zum Beispiel Listing 1 den Versionsstring aus.

```
>p 0 0 0

OOBD D2 651 Lux-Wolf CAN-Invader Thu, 15 Aug
2013 01:15:25 +0200
```

Listing 1: Ausgabe des Versionsstring über einen p-Befehl

So primitiv und für den Menschen unhandlich dieses Format auch ist, so einfach hat es eine Anwendung, mit dem Dongle zu kommunizieren; auch der Inputparser der Firmware braucht längst nicht alles zu verstehen, er reicht Befehle anderer Funktionsgruppen und Daten einfach an die entsprechenden Funktionsgruppen weiter. So sieht dann eine normale Kommunikation mit dem Dongle wie in Listing 2 aus. (die #-Kommentarzeilen dienen lediglich zur Veranschaulichung)

```
# Umschalten des CAN-Anschlusses auf die
Pins des High Speed CAN
>p 8 4 0
.
# CAN Bus initialisieren für 500 kb/s , 11
Bit CAN-ID
>p 8 3 3
.
# CAN Transceiver in den normalen Transmit
Mode schalten (CAN Active)
>p 8 2 3
.
# CAN Empfangs- Filter 1 auf 0x000  setzen
>p 8 10 1 $000
.
# CAN Empfangs- Maske 1 auf 0x000  setzen
(alle IDs 0x000 - 0x7FF) erlaubt)
>p 8 11 1 $000
.
# Aktuelles Protokoll anzeigen lassen
>p 7 0 0
1 - UDS (ISO14229-1)
.
# Daten senden (an Funktionale Modul-Adresse
0x7DF = Broadcast)
>1003
# Und das Modul antwortet
5003003200c8
.
# Done :-)
```

Listing 2: Normale Kommunikation mit einem Dongle

Die Dongle Protokolle und Busse

Im Moment ist der Dongle kompatibel zu dem CAN-Transportprotokoll (ISO-TP) nach ISO 15765-2. Dieses erweitert einen Standard CAN-Frame mit max. 8 Nutz-Datenbytes auf eine maximale Anzahl von 4095 (Nutz-)Datenbytes. Diese sogenannten Multiframes werden v.a. im Umfeld der Fahrzeugdiagnose-Kommunikation oftmals verwendet. Beim KWP2000 on CAN (ISO 15765-4) und UDS (Unified Diagnostic Services, ISO 14229) Protokoll auf den zwei umschaltbaren CAN- Bussen kommt dies ebenfalls zum Einsatz. Darüber hinaus gibt es einen CAN- Raw- Modus, der das Mithören auf dem Bus sowie das Abschicken längerer CAN- Sequenzen ermöglicht. Ein weiterer Modus, das sogenannte RealTimeData (RTD) Protokoll, welches Echtzeitdaten (Raw data) im Dongle zwischenspeichert und auf Anfrage zur Applikation schickt, ist bereits enthalten.

Die OOBD- Applikation

Der Dongle erfüllt bereits wie oben beschrieben die an ihn gestellten Anforderungen, doch wie erfüllt die dazu passende Anwendung die Forderungen nach Einfachheit und Anpassungsfähigkeit, ohne herstellerspezifische Geheimnisse mit sich tragen zu müssen?

Der Trick ist: OOBD besitzt einen eingebauten Script-Interpreter, und zwar für die LUA-Skriptsprache. Diese einfache und überaus mächtige Programmiersprache kann nahtlos um neue Funktionsaufrufe ergänzt werden, mit denen das Lua-Programm mit seinem umgebenden Hauptprogramm kommunizieren kann, und genau das wird in OOBD auch umgesetzt. Es gibt eigene Befehle, um die Fensteransicht aufzubauen, um über Datenbanken Fehlernummern in Klartext umzuwandeln und um über die serielle Schnittstelle mit dem Dongle zu sprechen. D.h. man schreibt sich seine gewünschte Funktionalität in Lua, kompiliert und speichert das File dann auf seinem OOBD-Gerät ab. Das OOBD-Programm zeigt dann die verfügbaren Möglichkeiten im Auswahlfenster an, man wählt das gewünschte Programm aus und ab geht's. Am seriellen Port muss auch nicht unbedingt ein Diagnosetester hängen, sondern das funktioniert mit allem, mit dem man seriell kommunizieren kann.

Das Lua-Listing, um das Hauptmenü darzustellen und von da aus dann z.B. die Fahrgestellnummer (engl. VIN) eines Autos auszulesen, sieht aus wie in Listing 3 dargestellt. Man muss nun aber nicht grundsätzlich alles von Hand programmieren. Für den eingangs erwähnten Automobilhersteller gibt es z.B. mittlerweile (interne) Bibliotheken, mit denen komplette Modulspezifikationen automatisch ins Lua-Format übersetzt werden und so innerhalb von Minuten hunderte neue Messgrößen zur Verfügung stehen.

```
function Start(oldvalue,id)
   identifyOOBDInterface()
   setSendID(„$7E8“) -- set specific ISO 15765 compatible sender
(=answer) address for OOBD firmware
   openPage(„OOBD-ME Main“)
   addElement(„Sensor Data>“, „createCMD01Menu“,“>>>“,0x1, „“)
   addElement(„Snapshot Data>“, „createCMD02Menu“,“>;;>;;>;;“,0x1, „“)
   addElement(„Dynamic Menu3>“, „createCMD03Menu“,“>;;>;;>;;“,0x1, „“)
   addElement(„Trouble Codes“, „showdtcs“,“-“,0x1, „“)
   addElement(„VIN Number“, „vin“,“-“,0x2, „“)
   addElement(„Clear Trouble Codes“, „clearDTC“,“-“,0x0, „“)
   addElement(„System Info>>;;>;;“, „SysInfo _ Menu“,“>;;>;;>;;“,0x1, „“)
   addElement(„Greetings“, „greet“,““,0x1, „“)
   pageDone()
   return oldvalue
end

function vin(oldvalue,id)
   echoWrite(„0902\r\n“)
   udsLen=receive()
   if udsLen>0 then
     if udsBuffer[1]==73 then
            local pos=4
            local res=““
            while pos <= udsLen and pos <36 do
                  if udsBuffer[pos]>31 then
                        res=res..string.char(udsBuffer[pos])
                  end
                  pos = pos + 1
            end
            return res
     else
            return „Error“
     end
   else
     return „NO DATA“
   end
end
```

Listing 3: Kommunikation mit einem Dongle

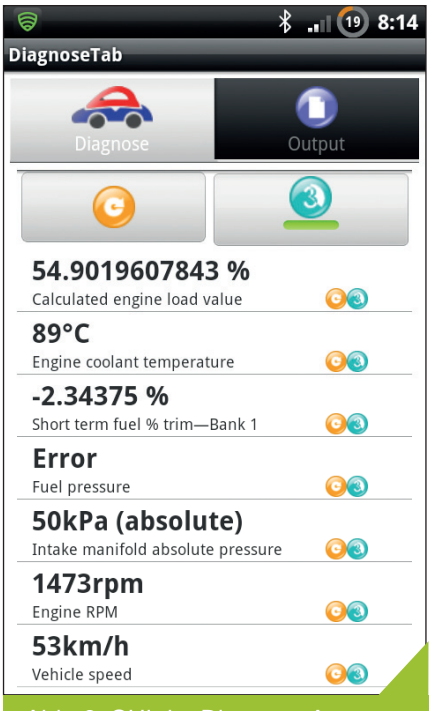


Abb. 3: GUI der Diagnose App

Eine gewünschte Sonderfunktion, die mehr auf die gewerbliche Nutzung abzielt, ist die Möglichkeit, die Lua-Skripte per PGP zu verschlüsseln und so nur den Anwendern zugänglich zu machen, die das passende Keyfile und Passwort haben. Und um das Ganze nun auch möglichst weiträumig nutzen zu können, sind die Apps in Java geschrieben, so das die selben Kern- Sourcen sowohl in Android als auch auf Windows, Linux und Mac funktionieren, lediglich die Oberflächen waren anzupassen.

Und wie geht‘s weiter?

Als wir damals mit dem Projekt anfingen, hätten wir nie gedacht, wie weit es uns tragen würde, denn jede geschaffene Möglichkeit weckte wieder neue Begehrlichkeiten, was man denn noch so dazu bringen könnte. OOBD ist heute bereits bei einem großen Hersteller im Einsatz, der das Projekt auch aktiv unterstützt, aber es gibt immer noch viele neue Ideen und Möglichkeiten, aber eher zuwenig Mitarbeiter; darum wie bei den meisten OpenSource Projekten auch hier die Ansage: Wer Interesse hat, an OOBD mitzuwirken oder es für seine eigenen Anforderungen zu erweitern: Wir freuen uns über jeden neuen Mitstreiter.

Literatur

[1] OpenOnBoardDiagnostics: www.oobd.org

[2] Echtzeitbetriebssystem FreeRTOS: www.FreeRTOS.org

Ein Mini-Mikroprozessor

Hypothetische Akkumulator-Maschine

Kai Dorau <kai.dorau@gmx.net>

Erste Überlegungen

Im Prinzip lassen sich mit klassischen Steuerwerken Mikroprozessoren realisieren, die eine ISA-Architektur (Instruction Set Architecture) implementieren. Eine Sequenz von Maschinenbefehlen kann als Sequenz von Maschinenprogrammen aufgefasst werden. Der prinzipielle Ablauf der Befehlsverarbeitung ist:

- Fetch Cycle (Befehlsholphase)
- Decode Cycle (Befehlsdekodierungsphase)
- Execute Cycle (Befehlsausführungsphase)

Diese drei Phasen sind u.a. Bestandteil einer sog. Von Neumann-Architektur. Im Folgenden wird eine einfache Beispiel-CPU realisiert, die speziell für Schulungszwecke den Einstieg in die Mikroprozessorwelt erleichtert. Entwickelt wurde die HAM vom Institute of Computer Science der Universität Osnabrück.

Die Beispiel-CPU HAM

HAM steht für Hypothetische Akkumulator Maschine und stellt so eine einfache CPU dar. Es sind nur die absolut notwendigen Befehle integriert. Sie kann aber ohne großen Aufwand erweitert werden. Die Maschine hat einige Einschränkungen, die der erfahrene Anwender leicht erkennt:

Die Register sind vorzeichenlos!

Bei arithmetischen Operationen finden keine Überlauferkennung und keine entsprechende Verarbeitung statt.

Das Blockschaltbild der Hypothetischen Akkumulator Maschine ist wie folgt gezeigt. Die HAM besteht aus folgenden Hardware-Komponenten:

- **Data Processing Unit**: Diese Unit stellt das Rechenwerk aus Akkumulator und ALU zur Verfügung. Sämtliche arithmetische und logische Operatoren werden hier durchgeführt.

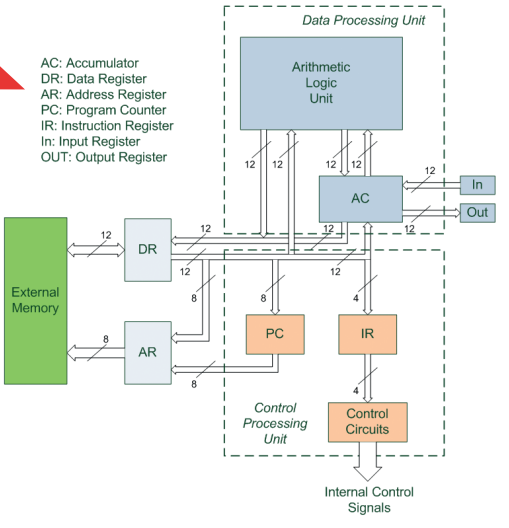


Abb. 1: Blockschaltbild der HAM

- **Control Processing Unit:** Das Steuerwerk besteht aus dem Programm-Zähler und dem Befehls-Register. Interne Kontroll-Signale regeln den Ablauf.
- **Speicher-Schnittstelle**: Der externe Speicher wird über das Adress- und Daten-Register angebunden. Der Speicher enthält das Programm inklusive Daten.
- **I/O-Schnittstelle**: Über ein Input- und Output-Register kann die HAM Kontakt zur Außenwelt herstellen.

Software-Modell der HAM

Um die Maschine zu programmieren, benötigt der Entwickler eine Übersicht über den Befehlssatz der Beispiel-CPU. Mit diesem Minimal-Befehlssatz ist man in der Lage, fehlende arithmetische und logische Operationen wie z.B. die Subtraktion zu programmieren. Die folegende Übersicht enthält Mnemonic, Opcode und die Beschreibung der Befehle (Tabelle 1). Die Befehle sind in dem folgenden Ablaufplan dargestellt (Abbildung 2).

Mnemonic	Opcode	Function	Description
LOAD X	1	AC <- M(X)	Move content of ext. Memory to Accumulator
STORE X	2	M(X) <- AC	Store accumulator at memory location X
ADD X	3	AC <- AC + M(X)	Two's-complement addition
ADD X	4	AC <- AC & M(x)	Logical AND
JUMP	5	PC <- X	Unconditional branch to X
JUMPZ	6	PC <- X, if AC=0	Conditional branch to X
COMP	7	AC <- !AC	One's-complement accumulator
RSHIFT	8	RingShift-AC	Ring-right-shift of the Accumulator
IN	9	AC <- IN	Input to AC
OUT	A	OUT <- AC	AC to Output

Tabelle 1: Befehlssatz der HAM

HAM-Design in RTeasy

RTeasy ist eine Entwicklungsumgebung für die Registertransfersprache. Mit ihr ist es möglich, Registertransferprogramme zu entwerfen und zu simulieren. RTeasy ist nicht standardisiert wie VHDL, Verilog oder System-C, aber sie ist sehr leicht zu erlernen und entsprechend können komplexe Modelle entwickelt und getestet werden.

RTeasy wurde vom Institut für technische Informatik der Universität Lübeck designt. Die IDE und ein ausführliches Tutorial findet sich hier: http://www.iti.uni-luebeck.de/lehre/rteasy.html. Einige syntaktische Gegebenheiten:

- Default Clock-Flanke ansteigend
- Clock-Zyklus von Semikolon zu Semikolon
- Komma gibt parallel-Verarbeitung innerhalb eines Clock-Zyklus an
- „|"-Zeichen kennzeichnet absteigende Clock-Flanke

Das RTeasy-Modell der HAM sieht nun folgendermaßen aus. Kommentare sind im Code zu finden. In der Init-Phase werden das Adress-Register und der Programm-Zähler zurückgesetzt. In der Fetch-Phase werden die Befehle nacheinander aus dem Speicher gelesen.

In der Decode-Phase werden die Befehle dekodiert und angesprungen.In der Execute-Phase werden dann die jeweiligen Befehle abgearbeitet. Mittels des RTeasy-Tutorial ist es nun sehr einfach, die Maschine nachzuvollziehen.

Listing 1: Das RTeasy-Modell der HAM

```
# all required registers shown in block diagram
declare register AC(11:0), DR(11:0), AR(7:0), PC(7:0), IR(3:0)
declare register INREG(11:0), OUTREG(11:0)
declare memory M(AR,DR) # we need memory (12bit X 256Bytes)
# fetch command
INIT: PC<-0, AR<-0;
FETCH:  AR<-PC;
        read M;
        IR<-DR(11:8), PC<-PC+1|
# decode command
        if IR=1 then goto LOAD  else
        if IR=2 then goto STORE else
        if IR=3 then goto ADD   else
        if IR=4 then goto AND   else
        if IR=5 then goto JUMP  else
        if IR=6 then goto JUMPZ else
        if IR=7 then goto COMP  else
        if IR=8 then goto RSHIFT else
        if IR=9 then goto IN else
        if IR=10 then goto OUT
        else goto FETCH  fi fi fi fi fi fi fi fi fi fi;
# execute commands
LOAD:   AR<-DR(7:0);
        read M;
        AC<-DR | goto FETCH;
STORE:  AR<-DR(7:0);
        DR<-AC;
        write M | goto FETCH;
ADD:    AR<-DR(7:0);
        read M;
        AC<-AC+DR | goto FETCH;
AND:    AR<-DR(7:0);
        read M;
        AC<-AC and DR | goto FETCH;
JUMP:   PC<-DR(7:0) | goto FETCH;
JUMPZ:  if AC<>0 then goto FETCH fi; PC<-DR(7:0)| goto FETCH;
COMP:   AC<- not AC | goto FETCH;
RSHIFT: AC(11)<-AC(0), AC(10:0)<-AC(11:1) | goto FETCH;
# IN/OUT Ports
IN:     AC<-INREG, goto FETCH;
OUT:    OUTREG<-AC, goto FETCH;
```

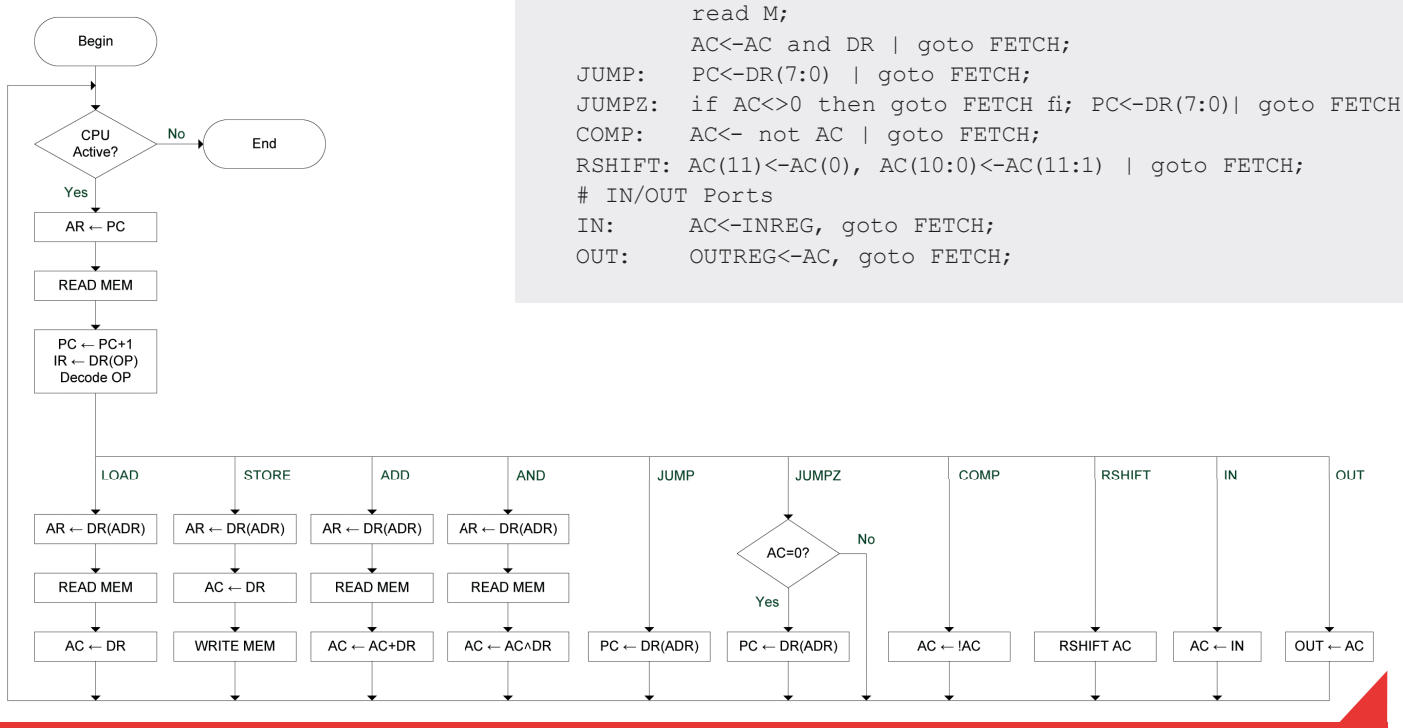


Abb. 2: Befehlsablaufplan der **H**ypothetischen **A**kkumulator **M**aschine

Software-Beispiel einer Subtraktion

Die HAM wird - wie jeder andere Prozessor auch – ohne Software keine sinnvolle Aktion durchführen können. Deshalb soll eine Subtraktion zweier vorzeichenloser Zahlen als Beispiel durchgeführt werden. Eine Tabelle in folgender Form hat sich für die Programmentwicklung in Assembler bewährt. Alle Zahlenwerte in hexadezimaler Form. Die Rosa-Spalten enthalten den Assembler-Code, die blauen Spalten links den Maschinen-Code.

Mem Adr	Mem Opcode	Mem Operant	Label	Mnem	Operant	Kommentar	Takte
00	1	51	SUB:	LOAD	V2	AC ← V2	6
01	7	00		COMP		AC ← !AC (one's complement)	5
02	3	53		ADD	ONE	AC ← AC+1 (two's complement)	6
03	3	50		ADD	V1	AC ← AC+V1 (V1-V2)	6
04	2	52		STORE	ERG	ERG ← AC	6
05	5	05	END:	JUMP	END	endless loop	4
06	0	00					
..							
50	0	32	V1				
51	0	21	V2				
52	0	00	ERG				
53	0	01	ONE				

Tabelle 2: Subtraktionsprogramm in Assembler und Maschinen-Code

Da nur der ADD-Befehl von der HAM zur Verfügung gestellt wird, muss eine Subtraktion mit dem Zweier-Komplement des Subtrahenden durchgeführt werden. Zuerst wird der Subtrahend (21h) in den Akku geladen. Dann wird nacheinander das Einer- und Zweier-Komplement des Subtrahenden gebildet. Danach wird der Minuend (32h) zum Akku addiert. Zum Schluss muss das Ergebnis im Akku (11h) noch in den Speicher an Stelle 52h geschrieben werden. Eine Endlosschleife einen definierten Befehl ausführt. Die Takte zählt man von Semikolon zu Semikolon. Die Subtraktion braucht demnach 29 Takte.

Subtraktion simuliert in RTeasy

In RTeasy sollte die Subtraktion simuliert werden, um sicherzustellen, das sie auch funktioniert. Zuerst lädt man das HAM-Modell in RTeasy und geht in den Simulationsmode. Dann überträgt man den Maschinen-Code (Spalte Mem Opcode/ Operant) in den Speicher (Content). Die Daten dürfen nicht vergessen werden. Danach arbeitet die Maschine im Step- oder Microstep-Mode das Programm ab.

- Step: Clockweise, von Semikolon zu Semikolon
- Microstep: Micro-Codeweise von Komma zu Komma

Wie die Abbildung 3 zeigt, das Ergebnis wurde gerade abgespeichert und ist auf Speicherstelle 52h verfügbar.

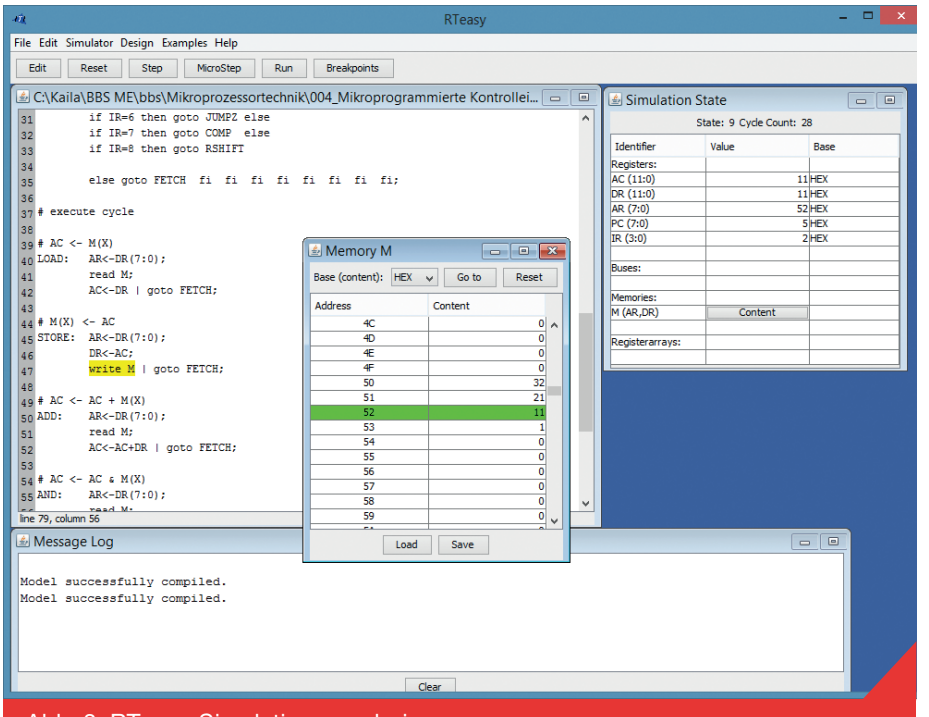


Abb. 3: RTeasy-Simulationsergebnis

Das VHDL-Modell

Da RTeasy für die Lehre entwickelt wurde, hat sie im Prinzip nur einen stark eingeschränkten praktischen Nutzen. Es gibt keine Möglichkeit, RTeasy-Code zu synthetisieren und auf eine CPLD- oder FPGA-Struktur abzubilden. Aus diesem Grunde wurde ein VHDL-Modell aus dem RTeasy-Modell generiert, das nun synthetisierbar ist und auf eine FPGA-Plattform gebracht werden kann. Das VHDL-Modell ist nachfolgend gezeigt (Abbildung 4). Das HAM-Design besteht im Wesentlichen aus der HAM-Entity, in der die Prozessor-Funktionalität abgebildet ist und der CodeMem-Entity, die den Code- und Datenspeicher repräsentiert. Die Toplevel-Entity verbindet beide Module miteinander und kapselt das Clock- und Reset-Signal, sowie die beiden I/O-Register. Dieses Modell wird auf ein Spartan-6 LX9 MicroBoard implementiert, so dass die Xilinx ISE-Toolchain zum Einsatz kommt.

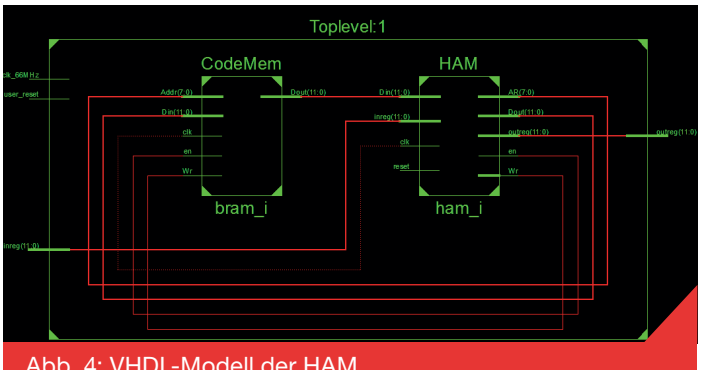


Abb. 4: VHDL-Modell der HAM

Subtraktion simuliert in ISIM

Während der Implementierung gibt es die Möglichkeit, mit unterschiedlichen Simulationsstrategien das Design zu testen. Zuerst ist die Verhaltenssimulation die erste Wahl. Mit ihr ist der Designer in der Lage, alle Signale und Signalvektoren sichtbar ins Verhältnis zu bringen. Ein Ausschnitt einer Verhaltenssimulation mit ISIM (Xilinx Simulator) kann man in Abbildung 5 sehen. Nach dem Reset (High-Active) läuft die HAM los und lädt einen Befehl nach dem anderen aus dem Speicher in das Daten-Register (Din). Im Akku (ac) sieht man die Komplementbildung des Subtrahenden und schlussendlich das Ergebnis 11h, das zusätzlich noch auf das Output-Register gelegt wird. Damit können auf dem LX9 MicroBoard LEDs angesteuert werden, um das Ergebnis auch „Live“ nachvollziehen zu können. Ein Blick in den simulierten Speicher verrät, dass das Ergebnis tatsächlich auf Speicherstelle 52h abgelegt wurde.

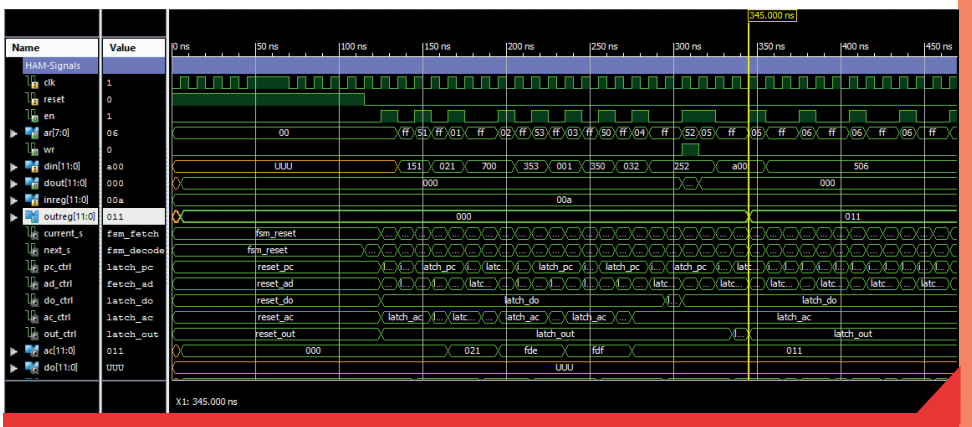


Abb. 5: Funktionale Simulation der Subtraktion mit ISIM

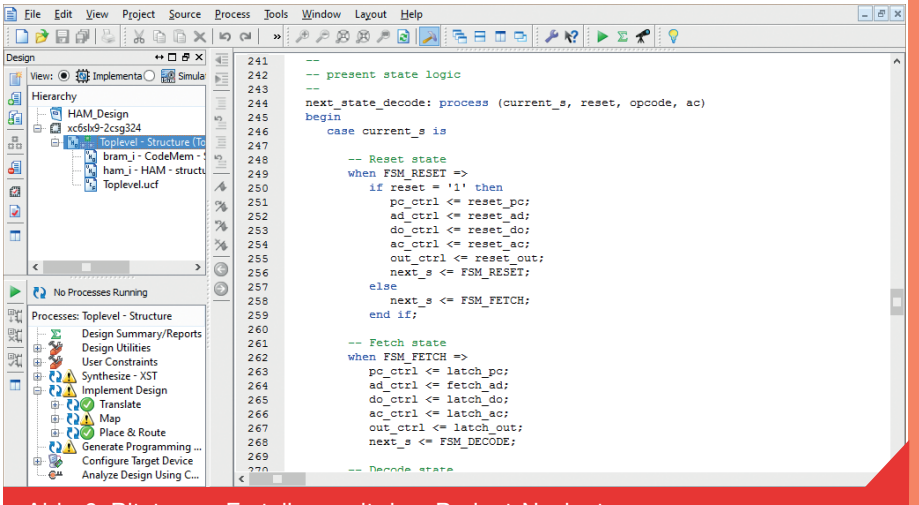


Abb. 6: Bitstream-Erstellung mit dem Project-Navigator

Das Spartan-6 FPGA LX9 MicroBoard

Das Spartan-6 LX9 MicroBoard ist eine Low-Cost-FPGA-Plattform, mit der ein komplettes „Embedded-System“ inklusive MicroBlaze und Linux realisiert werden kann. Die technischen Daten sprechen für sich:

- Xilinx Spartan-6 XC6SLX9 FPGA, Package: 2CSG324C
- 64 MB LPDDR SDRAM
- 128 MB Multi-I/O SPI Flash
- 10/100 Ethernet Phy
- USB-to-UART port
- On-board USB JTAG circuitry
- Programmable clock chip
- Four LEDs on board
- 4-bit DIP switch
- Reset and PROG push-buttons

Für das sehr interessante Board finden sich unter http://www.em.avnet.com/s6microboard diverse Tutorials, Dokumentationen, Schaltpläne und sonstige Informationen. R&S vertreibt es für netto 77 Euro.

Das HAM-Design auf dem LX9 MicroBoard

Das LX9 MicroBoard wird über die USB-Typ-A-Schnittstelle mit dem PC ver-bunden. Es wird darüber mit Strom versorgt und programmiert. Eine detaillierte Beschreibung der Inbetriebnahme findet sich im User Guide.

Das Toplevel-Design wurde für die Simulation im ISE integriert und kann nach erfolgreicher Simulation synthetisiert werden. Es sind im Prinzip 4 Implementie-rungsschritte notwendig.

1. Synthese: Dieser Schritt erzeugt aus dem VHDL eine xilinx-spezifische Netzliste.

2. Implementierung:

a.) Translate: Die Netzliste wird auf logischer Ebene gegen Xilinx-Primitives ersetzt.

b.) Map: Das Design in Form von Primitives wird auf physikalischer Ebene auf CLBs und IOBs transformiert.

c.) Place&Route: Zu guter Letzt werden die CLBs und IOBs mit-einander verbunden.

Ist die Synthese und die Implementierung ohne Fehler durchlaufen, wird der Bitstream für die Konfiguration des FPGAs erstellt.

Nun muss nur noch das Block-RAM mit den Programm-Daten gefüllt werden.

Softwareänderungen im Bitstream

Die Block-Rams im FPGA können nach der Bitstream-Erstellung mit Daten gefüllt werden. Das hat den Vorteil, dass Softwareentwickler unabhängig vom Hardware-Flow die Software auf einem Embedded-System implementieren können. Das nachträgliche Einbringen des Softwareprogramms wird mit Xilinx's „data2mem" bewerkstelligt. Das Dienstprogramm benötigt neben dem Bitstream und der Software im Memory-Format noch eine Block-Ram-Beschreibungsdatei „ham.bmm" (Listing 2).

```
// BMM file for my HAM running on LX9 MicroBoard
// manually edited by K.Dorau, 04.05.2014
// memory depth: 256, memory width: 12
//
ADDRESS_SPACE Mram_memory RAMB8 [0x00000000:0x000003FF]
    BUS_BLOCK
            bram_i/Mram_memory [15:0] PLACED = X1Y25;
    END_BUS_BLOCK;
END_ADDRESS_SPACE;
```

Listing 2: Inhalt der Block-Ram-Beschreibungs-Datei ham.bmm

An dieser Stelle wird nocheinmal auf die Tabelle 2 verwießen. In Zeile 5 wird noch zusätzlich der Akku – also das Ergebnis der Subtraktion – in das Output-Register übernommen.

Die Subtraktion als Softwareprogramm „ham.mem" im Memory-Format entspricht folgendem Aufbau (Listing 4).

Man kann das Subtraktionsprogramm sehr leicht ab Adresse @00 erkennen. Die Daten befinden sich ab Adresse @50. Die Nullen können, müssen aber nicht vorhanden sein. Sie bieten sich allerdings an, um den ganzen Inhalt zurück zusetzen.

Um die Software im Block-Ram des FPGA zu ändern, muss der „data2mem"-Aufruf folgendermaßen lauten:

```
data2mem -bm mem.bmm -bt
toplevel.bit -bd mem.mem
-o b  download.bit
```

Der Bitstream „download.bit" enthält nun das Subtraktionsprogramm.

Die Subtraktion auf dem MicroBoard

Um zu testen, ob das VHDL-Modell der HAM korrekt auf dem Spartan-6 FPGA läuft, wird die Subtraktions-Applikation visualisiert. In Zeile 5 der Assembler-Tabelle wird der Inhalt des Akkus in das Output-Register übernommen. Die unteren 4 Bits des Output-Registers, die auch die unteren 4 Bits des Ergebnisses im Akku repräsentieren, werden auf die 4 LEDs des LX9 Micro-Boards abgebildet. Dieses Port-Mapping findet in der entsprechenden UCF-Datei statt (Listing 3).

Welche LEDs müssen nun leuchten? Das Subtraktionsprogramm berechnet die Differenz von 32h – 21h = 11h = 10001b. Also muss die LED leuchten, die am Output-Register 0 angeschlossen ist…

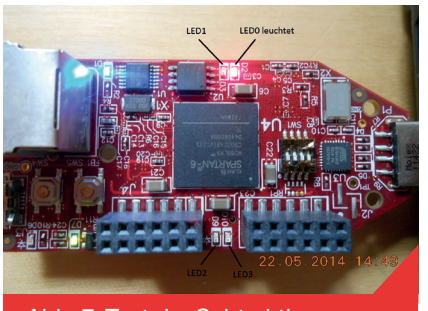


Abb. 7: Test der Subtraktion

```
NET outreg[0] LOC = „P4"  | IOSTANDARD = „LVCMOS18";
NET outreg[1] LOC = „L6"  | IOSTANDARD = „LVCMOS18";
NET outreg[2] LOC = „F5"  | IOSTANDARD = „LVCMOS18";
NET outreg[3] LOC = „C2"  | IOSTANDARD = „LVCMOS18";
```

Listing 3: Inhalt der UCF-Datei

Der Cross-Assembler

Wie man an dem Software-Beispiel in Abbildung 4 bemerkt, ist die Entwicklung von HAM-Programmen relativ schwierig und zeitaufwendig. Um etwas leichter Software zu entwickeln, ist ein Cross-Assembler geschrieben worden. Damit ist es nun möglich, mit einem Editor das Assembler-Programm zu entwickeln und dann entweder für RTeasy oder für das LX9 MicrBoard zu übersetzen. Das Statusdiagramm des Cross-Assemblers kann wie in Abbildung 8 dargestellt werden.

Im Prinzip besteht das Assembler-Programm für die HAM aus einem Code-Segment, einem Daten-Segment, Variablen, Label und dem Befehlssatz, die eine entsprechende Ausgabe in eine Memory-Datei generieren. Der Cross-Assembler ist in Java geschrieben, um eine möglichst große Kompatibilität unterschiedlicher Betriebssysteme zu gewährleisten. Das Subtraktionsprogramm kann man in Abbildung 9 sehen.

Das übersetzte Programm wird von der Memory-Datei (a.mem) repräsentiert. Anders wie in der obigen Memory-Datei sind keine Nullen nötig, die nicht zum Programm gehören.

```
@00
0151 0700 0353 0350 0252 0a00 0506
@A0
0032 0023 0000 0001
```

Das Subtraktionsprogramm entspricht exakt dem aus Tabelle 2, außer der Startadresse des Daten-Segments. Das VHDL-Modell liest aus dem Block-Ram und schreibt in das Block-Ram immer 16-Bit-weise. „Data2mem“ benötigt allerdings Byte-Adressen, also 8-Bit. Deshalb muss die Startadresse mit 2 multipliziert werden, in unserem Beispiel $A0.

Auch hier sorgt „data2mem“ für den Software-Update im Bitstream:

```
data2mem -bm mem.bmm -bt toplevel.bit -bd mem.mem -o b download.bit
```

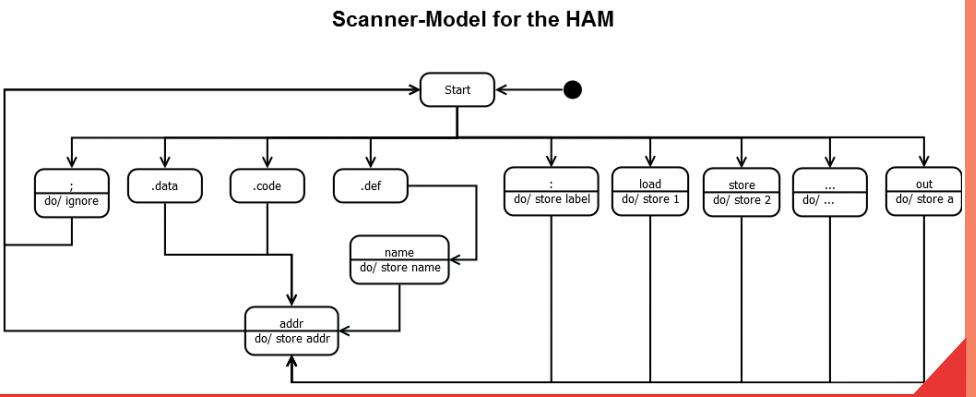


Abb. 8: Status-Diagramm des Cross-Assemblers

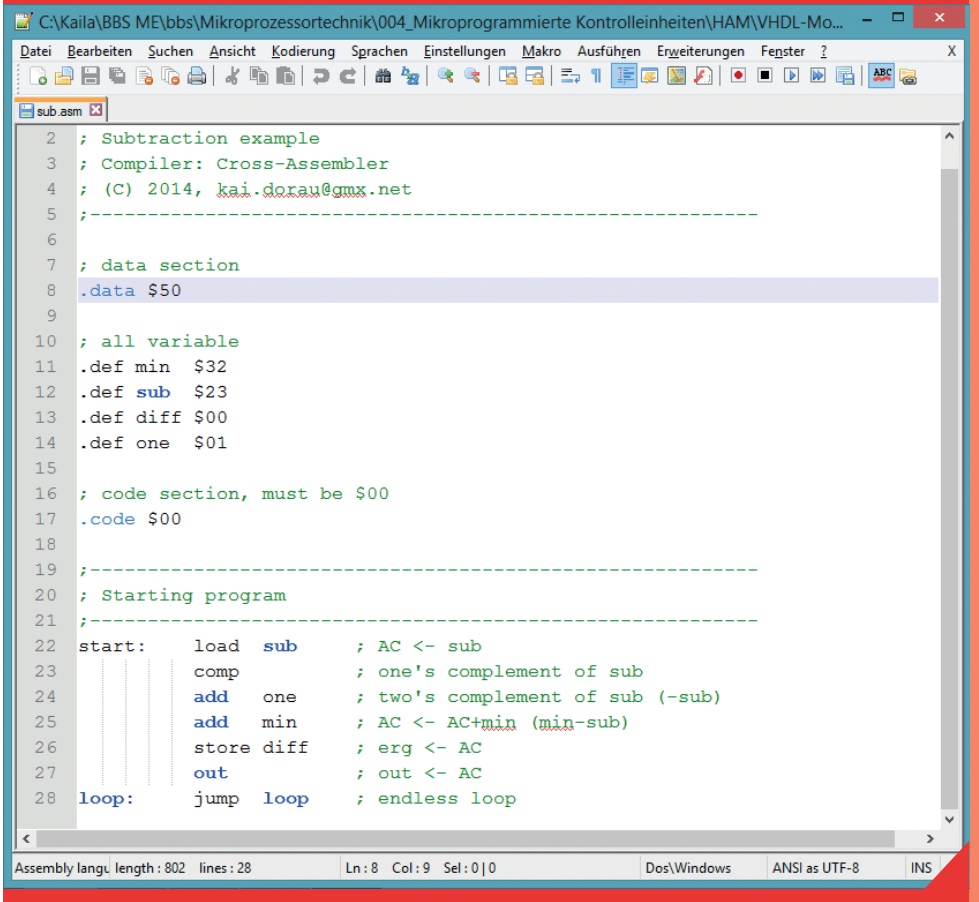


Abb. 9: Subtraktionsprogramm in Assembler

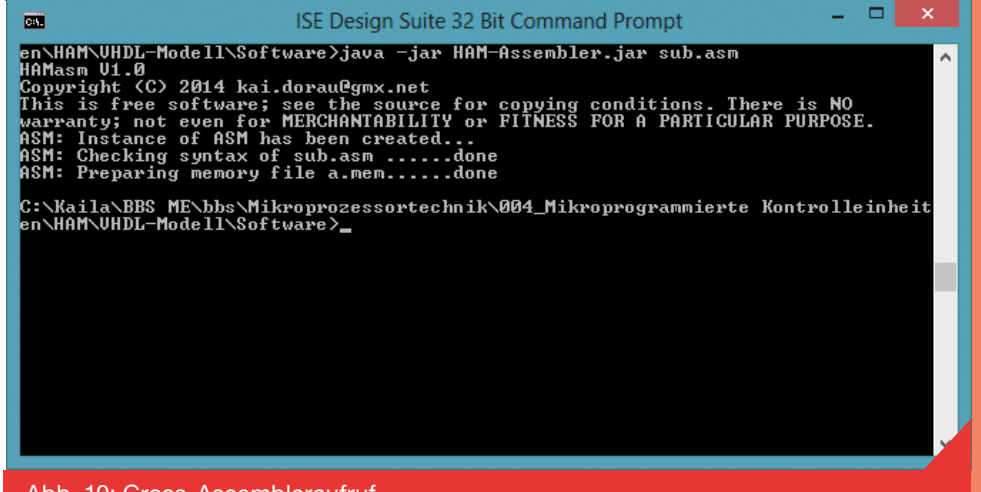


Abb. 10: Cross-Assembleraufruf

Ebook-Reader ohne Fußfesseln

Heiko Stübner <heiko@sntech.de>

Eine kurze Geschichte der Zeit

Im Oktober 2010 stellte die Buchhandelskette Thalia einen eigenen eBook-Reader vor – den Oyo (siehe Abbildung 1). Ein Jahr später im Oktober 2011 erschein der Oyo 2 – ein aufgefrischtes Modell mit leichten technischen Verbesserungen. Beiden aber war der große Durchbruch leider nicht vergönnt.

Eine technische Besonderheit beider Geräte waren die verwendeten ePaper-Display. Diese kamen nicht von PVI-eInk wie in der großen Masse der anderen eReader, sondern von Sipix, das zuvor vom großen chinesischen Displayhersteller AUO aufgekauft wurde.

Ein großes Problem der Sipix-Displays, insbesondere der ersten Generation, war ihr deutlich schlechterer Kontrast. Besonders deutlich war dieser Unterschied im Vergleich mit den, auch damals schon erhältlichen, eInk-Pearl Displays zu sehen. Besserung stellte sich erst mit den Displays der zweiten Generation ein, wie sie im Oyo 2 verbaut wurden. Diese hatten ein deutlich höheres Kontrastverhältnis und waren den Perl-Displays nahezu ebenbürtig.

Eine weitere interessante Komponente ist der kapazitive Touchscreen. Dieser verwendet AUOs sogenannte In-Cell-Technologie bei welcher der Touchscreen in das Display selbst integriert ist und nicht als zusätzliche Schicht auf dieses aufgebracht wird. Dadurch werden Kontrastverluste durch den Touchscreen im Vergleich zu Anderen verringert.

Als Prozessor verwenden Oyo 1 und 2 einen SoC von Samsung – S3C2416/400MHz im Oyo 1 und S3C2450/533MHz im Oyo 2. Bei erscheinen des Oyo 1 durfte sich dieser noch zur (unteren) Mittelklasse zählen, wurde aber im Folgenden von anderen SoC-Neuerscheinungen zunehmend abgehängt.

Eine interessante Entdeckung war, dass der Oyo keine Einzelentwicklung war, sondern Mitglied einer ganzen Familie von eBook-Reader ist. Über die ganze Welt verteilt, gibt es mindestens 20 verschiedene Modelle, die sich die gleiche oder mindestens ähnliche Hardware teilen. Ein relativ bekanntes weiteres Mitglied der Familie ist zum Beispiel der Asus eeeReader mit einem 9-Zoll-Display. Entwickelt und hergestellt wurden die Geräte von Qisda – der inzwischen eigenständigen ehemaligen Produktionssparte von BenQ.

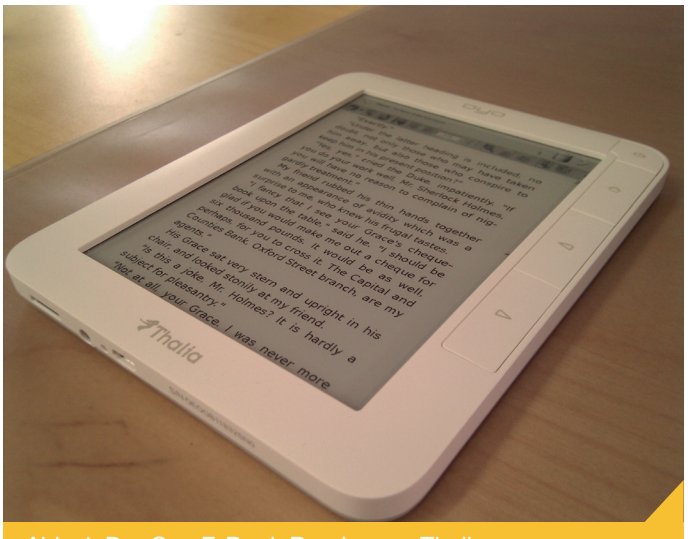


Abb. 1: Der Oyo E-Book Reader von Thalia

Problemär

Während seiner gesamten Verkaufszeit wurde der Oyo von Software- und Geschwindigkeitsproblemen geplagt. Der Hauptverursacher für diese, war mit sehr hoher Warscheinlichkeit der verwendete Linux-Kernel.

Dieser basierte auf einem – auch 2010 schon sehr alten – 2.6.21 aus dem Jahr 2007. Darauf wurde dann der Code für den Samsung Prozessor und darauf wiederum von Qisda der Code für die eigentliche eReader-Hardware gesetzt. Besonders der letzte Schritt erfolge in sehr schlechter Qualität, was vermutlich der Hauptgrund für die späteren Fehler und Probleme gewesen sein wird.

Ein häufiges Muster waren Code-Teile der unabhängig voneinander und ohne jedes locking auf die selbe Hardware-Komponente zugriffen – was natürlich zwangsläufig zu Problemen führt.

Die Unterstützung für den S3C2416/S3C2450 im Mainline-Kernel war zu diesem Zeitpunkt jedoch auch nicht besser – genauer war sie extrem unvollständig. Der im Oktober 2010 aktuelle

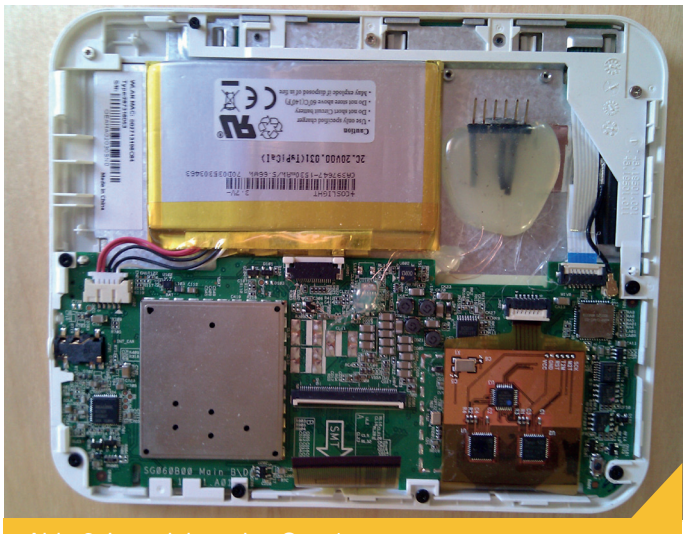


Abb. 2: Innenleben des Oyo 1

Kernel 2.6.36 konnte zwar auf dem Oyo booten, mehr aber auch nicht. Es fehlte bereits an elementarer Unterstützung für die SDHC-Kanäle, den USB-Gadget-Port und auch alle sonstigen Komponenten des SoC. Das heißt es gab weder Massenspeicher noch die Möglichkeit eine USB-Verbindung zu nutzen.

Hack-a-ton

Wie bei den meisten neu erscheinenden mobilen Geräten fanden sich auch beim Oyo einige Enthusiasten zusammen, die herausfinden wollten, was sich mit dem Gerät noch alles anstellen lässt.

Das größte Problem bei solchen Anti-Vendor-Ports ist aber natürlich immer der Mangel an Informationen. Einerseits fehlt natürlich jegliche Dokumentation zum Aufbau der Hardware selbst – d.h. welche Pins des SoC sind mit welcher Komponente verbunden. Andererseits sind auch die Hersteller der wichtigsten Komponenten nicht gerade auskunftsfreudig – Datenblätter sind also meißt schwer zu bekommen.

Zuerst wurde natürlich eine serielle Schnittstelle gesucht. Interessanterweise besitzen die Geräte einen richtigen Debug-Port und nicht nur Lötstellen auf der Platine dafür. Dieser Debug-Port stellt nicht nur drei der vier seriellen Schnittstellen des S3C2416 zur Verfügung, sondern für Abenteuerlustige auch dessen JTAG-Pins. Leider hat der Anschluss ein sehr eigenartiges Format – bezeichnet als 2KK2173018.

Eine besondere Erkenntnis der ersten Tage war, dass der Reader einen ungeschützten Dual-Boot-Modus besitzt. Es ist damit möglich beliebige Kernel-Abbilder von einer SD-Karte zu starten ohne dass originale System zu gefährden.

Eine andere Überraschung war, dass Patches für die weiter oben bereits erwähnten SDHC-Kanäle und den USB-Gadget-Port bereits einige Monate zuvor auf Kernel-Mailinglisten gepostet, dann aber vergessen wurden. Hier half also bereits ein kleiner Schubser um diese in den Mainline-Kernel zu bekommen.

Hardwaredetails

Der S3C2416 ist ein interessanter Chip, da er genau zwischen den Samsung-SoC-Epochen zu sitzen scheint. Er besitzt zwar nur einen ARMv5-Kern (genauer einen ARM926-EJS), teilt sich aber die meisten seiner Peripherie-Komponenten mit den neueren SoCs bis hin zum aktuellen Exynos. Obwohl die Zahl der S3C2416-interessierten Entwickler sehr überschaubar ist, wird es dadurch möglich, mit relativ geringem Aufwand von den meisten Entwicklungen der neueren Samsung-SoCs zu profitieren.

Die Energieverwaltung der Reader ist sehr einfach aufgebaut. Ein TPS650240 von Texas Instruments produziert die meisten benötigten Spannungen und ein BQ24075 vom selben Hersteller kümmert sich um das Laden des Akkus. Wo bei anderen Geräten der PMIC über ein I2C-Interface umfangreiche Einstellungen und feine Dosierung der Spannungen ermöglicht, kommen sowohl TPS650240 als auch BQ24075 mit ein paar GPIOs aus. Sie werden deshalb vom dafür neu entwickelten gpio-regulator-Treiber verwaltet, der beliebige per GPIOs angebundene Spannungs- und Stromregler verwalten kann. Es existieren noch ein paar weitere ganz einfache Spannungsregler, die nur ein- und ausgeschaltet werden können und deswegen vom fixed-regulator-Treiber verwaltet werden.

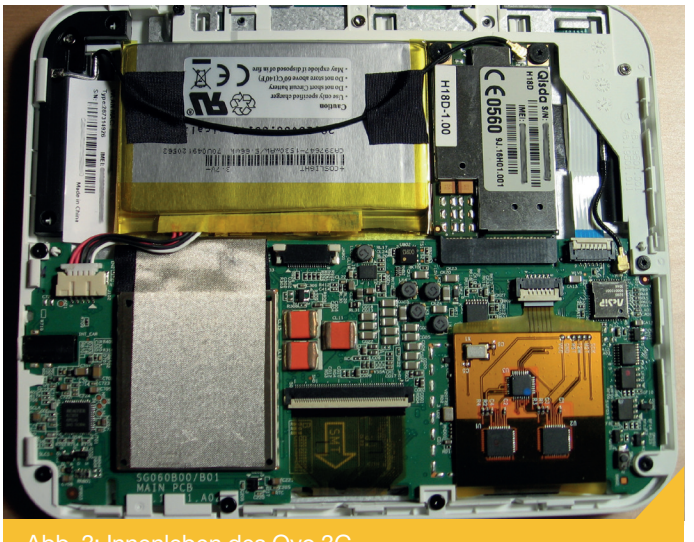


Abb. 3: Innenleben des Oyo 3G

Die Hardwaretasten des Oyos werden ganz simpel über GPIOs angebunden. Dafür existiert ein Konnektor auf dem Board, der diese GPIOs über ein Flachbandkabel zu den Tasten führt. Als Treiber für diese wird natürlich gpio-keys genutzt. Durch diese Konstruktion mit dem Flachbandkabel liessen sich die GPIOs aber auch umnutzen, da die Verbindung natürlich auch zur anderen Komponenten hergestellt werden kann.

Ein weiteres interessantes Detail ist das Qisda H18D Modem der 3G-Variante des Readers (siehe Abbilung 3). Es nutzt die USB- und SIM-Signale der Mini-PCIe-Verbindung, welchen dann wiederum zum USB-Host-Controller des S3C2416 geführt sind. Da jedoch nur der Gadget-Controller USB 2.0 untersützt und der Host-Controller auf USB 1.1 limitiert ist, sind die erreichbaren Geschwindigkeiten gering.

Der ePaper-Controller

Das wohl wichtigste Bauteil ein einem eBook-Reader ist aber vermutlich der ePaper-Controller. Dieser stammt – wie das Display – von AUO. Der Controller im Oyo 1 nennt sich K1900 und die Variante im Oyo 2 K1901.

Ein grosser Unterschied zwischen gewöhnlichen Displays und elektronischem Papier ist die Updatezeit. Während normale Displays unzählige Updates pro Sekunde bewerkstelligen, schaffen ePaper-Displays im Allgemeinen höchstens ein bis zwei. Das heißt ein vollständiges Update des gesamten Displays dauert meist zwischen einer halben und einer Sekunde.

Im Gegensatz zu gewöhnlichen Displays wird elektronisches Papier auch nicht kontinuierlich aktualisiert. Stattdessen müssen dem Controller einzelne Updatebereiche und die dort darzustellenden Daten explizit mitgeteilt werden.

Ein Update des Displays erfolgt dabei nach folgendem Schema:

- start Update (x, y, width, height)
- Übermittlung der Grauwerte für jedes Pixel dieser Box
- stop Update und ausführen der Änderung des Displays

Dieses Vorgehen ist natürlich nicht kompatibel mit den Erwartungen von generischen Programmen. Diese erwarten einen Framebuffer in den sie einfach ihre Pixel schreiben können, die dann während des nächsten sync-Vorgangs von selbst auf dem Display erscheinen.

Um also ein ePaper-Display anzusteuern, müssen entweder die Anwendungen angepasst werden, damit sie bei Display-Änderungen die Update-Kommandos an den Controller senden oder man bedient sich eines Tricks der den Anwendungen einen Standard-Framebuffer vorgaukelt und sich selbstständig um das Auslösen der Updates kümmert.

Der verwendete Trick nennt sich deferred-io.

Dabei wird ein Framebuffer angelegt, der normalen Arbeitsspeicher verwendet. Alle Speicherseiten dieses Bereiches werden als nur-lesen markiert. Bei den Schreibvorgängen auf den Framebuffer-Speicher die durch Display-Änderungen entstehen kommt es dadurch zu Page-Faults. Für diese wird eine spezielle Behandlungsroutine angelegt, die die betreffende Speicherseite in eine Liste einträgt und danach den Schreibzugriff freigibt.

Nach Ablauf eines konfigurierbaren Zeitintervals, wird eine Funktion im Treiber aufgerufen. Diese erhält die Liste der veränderten Speicherseiten und löst das hardwarespezifische Update dieser Bildschirmbereiche aus. Wenn dies geschehen ist, wird die Liste der Speicherseiten geleert, allen Seiten das Schreibrecht entzogen und der Zyklus von neuem begonnen.

Dieser Mechanismus fasst also Änderungen am Displayinhalt zusammen und ermöglicht eine nach oben begrenzte Anzahl von Updates pro Sekunde. Solange eine Speicherseite in der Liste steht bleibt die Seite auch beschreibbar, sodass bei weiteren Schreibzugriffen auf dieselbe Seite kein weiterer Page-Fault und damit kein Overhead entsteht.

Hier tritt auch der gravierenste Unterschied zwischen dem K1900 und dem deutlich leistungsfähigeren K1901 Controllern zu tage. Der K1900 kann immer nur genau ein Update zu jeder Zeit verarbeiten. Da Updates der Sipix-Displays im Mittel eine Sekunde dauern, limitiert dies das deferred-io-Intervall auf 1fps. Im Gegensatz dazu kann der K1901 mehrere Updates gleichzeitig verarbeiten und dabei entstehende Kollisionen von Updates der selben Region selbstständig auflösen. Obwohl sich die Geschwindigkeit des Displays selbst dabei nicht erhöht, lassen sich damit 4 frames pro Sekunde erreichen.

Aktueller Zustand und Ausblick

Das ursprüngliche Ziel, einen vollständigen eBook-Reader ohne die titelgebenden Fussfesseln auf die Beine zu stellen, ist jedoch leider nicht mehr zu erreichen. Zum einen ist dies als Projekt für eine Einzelperson viel zu umfangreich und andererseits ist die Hardwareplatform mittlerweise so alt, dass sie vermutlich niemand mehr freiwillig zum lesen einsetzen würde.

Nicht desto trotz ist ein System auf basis der Kernelquellen funktionstüchtig und kann zum lesen eingesetzt werden – z.B. mittels fbreader auf einem Debian-System.

Interessanterweise sind die Reader aber auch zu weit mehr fähig, als es die originale Firmware und User-Meinungen vermuten ließen. So reichen die 4 Frames pro Sekunde des K1901 im Oyo 2 dazu, Spiele auf Basis von ScummVM auf dem Gerät zu spielen, wie Abbildung 4 zeigt.v

Allgemein ist die Zahl der Entwickler, die sich mit dem S3C2416 im Mainline-Kernel beschäftigen überschaubar, weswegen die Reader-Platform für mich mehr zu einem Experimentierfeld geworden ist. Dadurch ist es möglich ganz ungestört neue Technologien auszuprobieren – wie zum Beispiel gerade laufende DeviceTree-Konvertierung der S3C2416 SoCs.

Abb. 4: ScummVM (Indiana Jones and the Fate of Atlantis)

Typical behavior of the I2C Bus

Simulation with COMSOL Multiphysics and LTSpice

Julian Sarcher <julian@sarcher.de>

Der I2C Bus wird bei vielen elektronischen Schaltung angewendet, damit Bausteine miteinander kommunizieren können. Typischerweise werden Daten mit einer Frequenz von 100kHz bis 400kHz gesendet. Anwendung findet der Bus meist für die Kommunikation auf einer Platine oder zwischen Platinen, welche in ein Gerät verbaut werden. Bei höheren Frequenzen und Leitungslängen treten hier bekannte Probleme wie z.B. Cross-Channel-Talk auf. Diese Probleme gehen vor allem auf parasitäre Kapazitäten zwischen den Leitungen zurück. Diese Kapazitäten werden über ein geometrisches Modell in COMSOL Multiphysics errechnet. Das daraus resultierende Bus-Verhalten wird anschließend in LTSpice simuliert.

Grundlagen

Der I2C Bus ist ein Low-Aktiver, synchroner Datenbus mit einer Datenleitung SDA und einer Taktleitung SCL. Er ist darauf ausgelegt, dass ein Master mit unterschiedlichen Slaves abwechselnd kommunizieren kann. Beginnt ein Teilnehmer zu senden, zieht dieser die Datenleitung auf Low. Daraufhin ist der Bus für alle anderen Teilnehmer gesperrt. Wenn kein Teilnehmer etwas sendet, wird der Bus über Pull-Up Widerstände an SDA und SCL auf HIGH-Pegel gezogen. Der Bus ist wieder freigegeben. Das typische Einsatzgebiet des I2C Bus ist die Kommunikation von Bausteinen auf einer Leiterplatte. Um dort Platz einzusparen, werden insbesondere die Datenleitungen, auf welchen nur ein kleiner Strom fließt, meist an der geometrischen Technologiegrenze designed. Dies hat jedoch zur Folge, dass sich gewisse parasitäre Kapazitäten ausbilden. Zunächst sollen die Leitungskapazitäten, welche zwischen den Datenleitungen und dem Massepotential und die Cross-Channel-Kapazitäten, welche sich zwischen den Datenleitungen ausbilden, betrachet werden.

Cross-Channel-Talk

Der Cross-Channel-Talk ist auf die Kapazität, welche sich zwischen den Datenleitungen ausbildet, zurückzuführen. Diese Interferenzen zwischen SDA und SCL treten immer dann auf, wenn eine der Datenleitungen ihren logischen Pegel wechselt, während die andere ihn behält. Daraufhin findet ein Ladungsträgerausgleich von einer Leitungskapazität zu der Cross-Channel-Kapazität statt, was im ersten Schritt zu einer Ausgleichspannung führt. Im zweiten Schritt müssen beide Kapazitäten über den 4,7kΩ Widerstand wieder aufgeladen werden. Mit typischen Technologiewerten von 35µm Kupfer-Dicke und einem Abstand von 0,2mm zwischen den Leiterbahnen kann mit der einfachen Formel

$$C = \varepsilon_0 \cdot \frac{A}{d}$$

die Kapazität bei einer 20cm langen Datenleitung umschlagsmäßig errechnet werden:

$$C_{cc} = 8,85 \cdot 10^{-12} \frac{As}{Vm} \cdot \frac{0,2m * 35\mu m}{0,2mm} = 0,31pF$$

In der Simulation später wird sich jedoch zeigen, dass es sich bei diesem Wert eher um einen Schätzwert, als um ein errechnetes Ergebnis handelt. Aus der Simulation erhalten wir einen Wert für Cross-Channel-Kapazität C_{cc} = 4,86pF. Dieser Wert wird vor allem problematisch, wenn er im Vergleich zu den Leitungskapazitäten hoch ist. Dem kann entgegengewirkt werden, indem man beim Leiterplattendesign größere Abstände zwischen SCL und SDA einhält oder die Länge der Datenleitung verkürzt.

Signalverzerrung durch Leitungskapazitäten

Die SCL Leitung wird bei einer Kommunikation mit f=400kHz alle T=2,5µs auf Low gezogen. Nach T/2 wird SCL vom Master wieder auf High Impedance gelegt und der 4.7kΩ Pull Up Widerstand zieht die Taktleitung wieder nach oben. Dies sollte spätestens nach T/2 geschehen sein, da der Slave in dieser Zeit den HIGH-Pegel erkannt haben muss. Die Zeitkonstante τ lässt sich wie folgt berechnen: τ=RC. Dabei sollte stets gelten:

$$5\tau < \frac{T}{2}$$

Daraus folgt:

$$\tau_{max} = 0,25\mu s$$

Um dies einzuhalten gibt es mehrere Möglichkeiten. Benötigt man ohnehin keine hohe Datenrate kann man im einfachsten Fall auch die Taktfrequenz verringern. Ansonsten besteht die Möglichkeit die Pull Up Widerstände zu verkleinern. Dabei ist unbedingt zu bedenken, dass dies zu einem erhöhten Stromverbrauch der Schaltung führt.

Simulation mit COMSOL Multiphysics

Die Simulation teilt sich in zwei Abschnitte ein. Zunächst werden in COMSOL Multiphysics mittels der Finite-Elemente-Methode alle Kapazitäten zwischen den Leiterbahnen ermittelt. Anschließend wird in LTSpice, einem SPICE Simulator für elektronische Schaltkreise, mit den simulierten Werten aus COMSOL Multiphysics ein Schaltung entworfen, welche den I2C Bus simuliert. Abschließend werden die generierten Kurven ausgewertet und bewertet.

Um in COMSOL Multiphysics Kapazitäten berechnen zu können, benötigt man zunächst das AC/DC Modul Electrostatics. Mit diesem Modul lassen sich jegliche elektrostatische Problemstellungen berechnen. Gelöst wird hier nach dem elektrischen Potential.

Das geometrische Model ist in meinem Fall simpel gehalten, was jedoch völlig ausreichend ist. Es wurde ein 2D-Schnitt durch die Leiterplatte erstellt. Auf der Unterseite ist die komplette Kupferfläche auf GND gelegt. Auf der Oberseite sind 3 Leiterbahnen zu erkennen. Diese stellen SDA, SCL und eine VCC Leitung dar. Die geometrischen Maße sind einer echten Leiterplatte mit 35µm Kupferdicke nachempfunden. Die Abstände zwischen den Leiterbahnen und die Breite der Leiterbahnen sind aus dem Default DRU-File der CAD-Software Eagle übernommen worden. In Abbildung 1 kann man die Leiterplatte im 2D-Schnitt sehen.

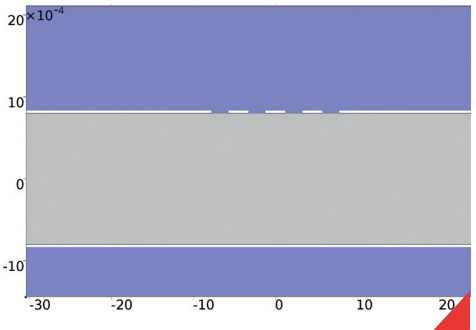


Abb. 1: Leiterplatte im 2D-Schnitt

Die Dicke der Leiterplatte ohne Kupfer beträgt 1,6mm. Als Basismaterial wurde FR4 gewählt. Um die Umgebung der Leiterbahnen zu simulieren wurde Luft als Umgebungsmaterial gewählt. Die Leiterbahnen wurden hierbei einfach aus der Luft herausgeschnitten. Das Kupfer wurde nicht einfügt, da es ohnehin das Simulationsergebnis nicht beeinflusst hätte. Als Länge der Leiterplatte, also Länge der Leiterbahnen in z-Richtung wurde 20cm gewählt.

Im nächsten Schritt müssen den einzelnen Oberflächen Potentiale zugewiesen werden. Zunächst müssen alle Masseflächen auf GND gelegt werden. Anschließend kann man allen Leitungen das Potential 5V zuweisen.

Bevor man das erste Mal das Ergebnis simulieren kann, muss man noch Netzeinstellungen vornehmen. Da relativ große Unterschiede in den Dimensionen vorhanden sind, reicht das voreingestellte Netz nicht mehr aus. Also wird das Netz auf Very Fine gestellt. Da das geometrische Model sehr einfach gehalten ist, halten sich auch die benötigten Berechnungen in Grenzen. In Abbildung 2 kann man den simulierten Potentialverlauf erkennen, wenn eine der Leiterbahnen auf VCC gelegt wird.

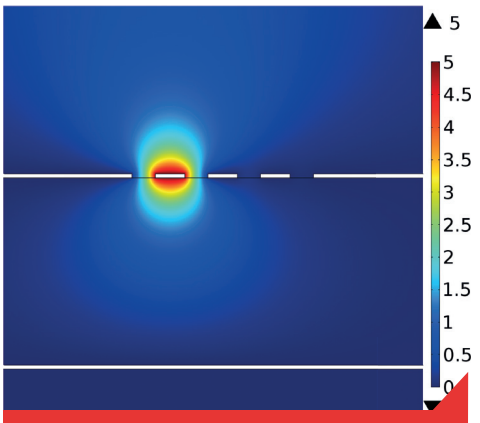


Abb. 2: Potentialverlauf bei VCC = 5V

COMSOL Multiphysics kann nun Lumped Parameter berechnen. Die konzentrierten Parameter, im Modul Electrostatics sind unter anderem die Kapazitäten. Um an alle Kapazitäten auf einmal heranzukommen benötigt man einen Parametric Sweep. Gesweept werden hier die Potentiale der Leiterbahnen. Jeweils eine wird auf 5V und alle anderen auf Masse gelegt. Anschließend kann COMSOL Multiphysics alle errechneten Kapazitäten in einer Matrix ausgeben. Folgende Matrix wird generiert:

$$\begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{pmatrix} \cdot \begin{pmatrix} V_1 \\ V_2 \\ V_3 \end{pmatrix}$$

Diese Matrix enthält sämtliche Kapazitäten, welche durch dieses Leiterplattendesign entstehen. Auf der Hauptdiagonalen der Matrix werden die Leitungskapazitäten auftauchen, also die Kapazitäten zwischen den Leiterbahnen und dem Massepotential. C_{12} und C_{21}, C_{13} und C_{31} sowie C_{23} und C_{32} stellen jeweils die selbe Kapazität dar. C_{12} bezieht sich hierbei auf die Kapazität zwischen dem ersten gesweeptem Potential und dem zweiten, C_{23} zwischen dem zweiten und dem dritten, usw. In dieser Simulation hat die Reihenfolge SDA, SCL, VCC folgende Matrix ergeben:

$$\begin{array}{c} \\ SDA \\ SCL \\ VCC \end{array} \begin{array}{c} \begin{array}{ccc} SDA & SCL & VCC \end{array} \\ \begin{pmatrix} 14.4pF & 4.86pF & 0.726pF \\ 4.86pF & 14.4pF & 4.86pF \\ 0.726pF & 4.86pF & 14.4pF \end{pmatrix} \end{array}$$

Simulation mit LTSpice

Im Weiteren wird das Verhalten am Bus nun mit dem Tool LTSpice simuliert. LTSpice bietet eine grafische Oberfläche zum Simulieren von elektronischen Schaltkreisen.

Zunächst gilt es, aus den extrahierten Daten einen möglichst realen Schaltplan zu entwerfen. Dazu wurden alle errechneten Kapazitäten integriert sowie die Leitungswiderstände mit einbezogen:

$$R_L = \rho_{cu} \cdot \frac{l}{A} = 0,39\Omega$$

Zudem wurden Pull Up Widerstände mit 4,7kΩ simuliert. Der daraus entwickelte Schaltplan ist in Abbildung 3 zu sehen.

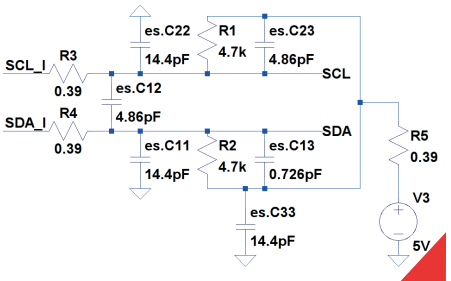


Abb. 3: Schaltplan in LTSpice

Auf diesem Schaltbild kann man noch nicht erkennen, wie die Signale, des Low-Aktiven Bus in die Schaltung eingebracht werden. Wichtig ist, dass SCL immer T/2 auf Low gezogen und die zweite Hälfte der Periode auf High Impedance geschal-

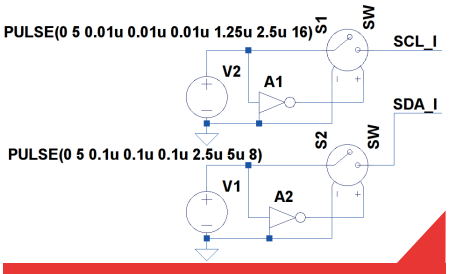


Abb. 4: Treiberschaltung in LTSpice

ten wird. Das wurde wie in Abbildung 4 zu sehen ist, über einen Voltage Controlled Switch und einem Inverter realisiert.

Da die Kapazitäten C_{23}, C_{13} sowie C_{33} nach dem ersten Aufladevorgang keine besondere Wirkung auf den Bus zeigen, kann man diese bei der anschließenden Simulation auch vernachlässigen.

Es wurde eine transiente Simulation gestartet und an den beiden Busleitungen SDA und SCL die Spannung abgegriffen. Dabei fielen einige markante Punkte auf. Diese wurden in Abbildung 5 markiert. Alle rot markierten Stellen gehen auf die Cross-Channel-Kapazität, die gelb markierte Stelle auf die Leitungskapazität zurück. Diese werden im nächsten Kapitel näher erörtert.

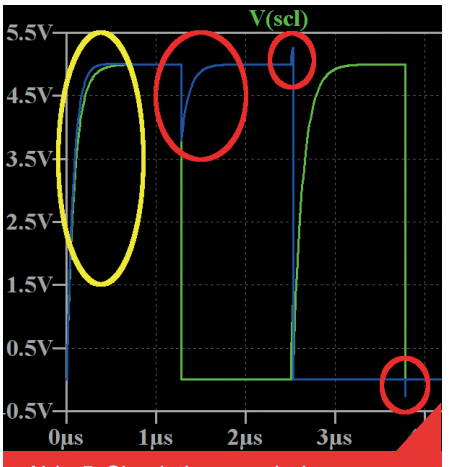


Abb. 5: Simulationsergebnis

Auswertung der Ergebnisse

Die Signalverzerrung (gelb), geht auf die Leitungskapazitäten und den Pull Up Widerstand R_P zurück. $\tau=R_PC_{11}$. Nach 5 τ sollten 99,3% der Maximalspannung anliegen, also 4,965V. 5τ=5*(4,7kΩ*14,4pF)=0,34µs. Diese Zeit kann man in Abbildung 5 noch relativ gut erkennen.

Maximale Leiterlänge: In COMSOL Multiphysics wurden anschließend noch unterschiedliche Leiterbahnenlängen simuliert. Das Ergebnis ließ einen linearen Anstieg der Kapazität proportional zur Leiterbahnenlänge l vermuten.

$$C(l) = 71,8\frac{pF}{m} \cdot l$$

Die Zeitkonstante errechnet sich nun durch: $\tau = R_P*C(l)$. Für die Zeitkonstante τ lässt sich sagen, dass sie auf keinen Fall den Wert $\tau_{max} = 0,25\mu s$ überschreiten darf. Setzt man nun C(l) und τ in die Gleichung ein und löst nach R_P auf, kommt man auf folgende Gleichung:

$$R_{Pmax}(l) = 3,48k\Omega m \cdot \frac{1}{l}$$

Diese Gleichung beschreibt, welchen Pull Up Widerstand man höchstens verwenden darf, bei vorgegebener Leiterbahnlänge l. Mit zunehmender Länge ist zudem der Zuleitungswiderstand nicht mehr zu vernachlässigen. Für diesen gilt:

$$R_L = 1,97\frac{\Omega}{m} \cdot l$$

Dieser Zuleitungswiderstand bildet nun mit dem Pull Up Widerstand einen Spannungsteiler. Beim Entladevorgang sollte die Spannung am Bus nicht über 10% von VCC bleiben, sonst können Busteilnehmer unter Umständen das Signal nicht mehr als ein logisches Low interpretieren. Daher gilt für R_{Pmin}:

$$R_{Pmin} = 9 \cdot R_L = 17,73\frac{\Omega}{m} \cdot l$$

Wird $R_{Pmax} < R_{Pmin}$ so ist das System nicht mehr realisierbar. Setzt man nun $R_{Pmax}=R_{Pmin}$ und löst nach der Länge l auf, so erhält man die maximal mögliche Leiterbahnenlänge l=14m. Daraus resultiert ein Zuleitungswiderstand von R_L=27,6Ω ein Pull Up Widerstand von R_P=248Ω sowie eine Leitungskapazität von C_L=1nF. In Abbildung 6 kann man das Simulationsergebnis für die 14m langen Leiterbahnen sehen.

Interferenzen zwischen SDA und SCL: Alle in Abbildung 5 rot markierten Stellen gehen auf die Cross-Channel-Kapazität zurück. Diese Interferenzen treten immer dann auf, wenn eine der Datenleitungen ihren logischen Wert wechselt, während die andere ihn behält. Gehen wir nun näher auf den Fall ein, welcher bei 1,25µs auftritt. SCL wechselt nun auf Low-Pegel. Zuvor war die Kapazität C_{12} nicht geladen, da auf beiden Seiten das gleiche Potential herrschte. Nun entsteht ein Potentialunterschied zwischen den beiden Leitungen. Dies hat zur Folge, dass ein Ausgleichsvorgang zwischen der vollgeladenen Kapazität C_{11} und der ungeladenen Kapazität C_{12} stattfindet. Auf dem Kondensator C_{11} befindet sich zunächst die Ladung Q:

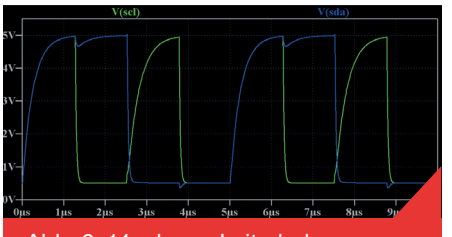


Abb. 6: 14m lange Leiterbahnen

$$Q = C_{11} \cdot U = 14,4pF \cdot 5V = 72pC$$

Auf der anderen Kapazität befindet sich

keine Ladung. Die Gesamtkapazität errechnet sich additiv zu C_{ges}=19,26pF. Somit stellt sich eine Ausgleichsspannung ein:

$$U_{Ausgleich} = \frac{72pC}{19,26pF} = 3,74V$$

Dies kann in Abbildung 5, bei t=1,25µs gesehen werden. Anschließend müssen beide Kapazitäten wieder über den Pull Up Widerstand aufgeladen werden. Dies geschieht mit der Zeitkonstanten $\tau = R_p * C_{ges}$ = 90ns. In Abbildung 7 kann man dies nachvollziehen. Gemessen wurde die Zeit zwischen 4τ und 5τ, also zwischen 5V*98,2%=4,91V und 5V*99,3%=4,965V.

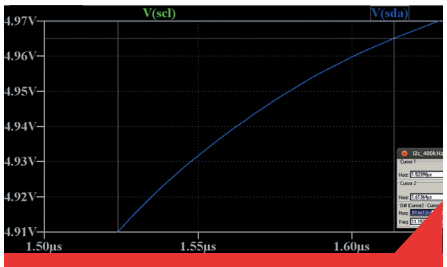


Abb. 7. Wiederaufladen von C_{ges}

Dieser Cross-Channel-Talk sollte zu keinen größeren Problemen führen, solange $C_{12} < C_{11}/3$ gegeben ist. Da sich in dieser Konstellation die Ausgleichsvorgänge in Grenzen halten. Um dies zu erreichen, sollte man gegebenenfalls einen höheren Abstand zwischen SDA und SCL beim Leiterplattendesign einplanen. Um ansonsten das Wiederaufladen der Kapazitäten zu beschleunigen, bleibt die Möglichkeit die Pull Up Widerstände zu verringern.

Fazit

Die Kombination der beiden Tools COMSOL Multiphysics und LTSpice ermöglicht es, viele parasitäre Aspekte, bereits vor dem Leiterplattendesign zu erkennen und so frühzeitig zu vermeiden. So kann man mit COMSOL Multiphysics auch andere konzentrierte Parameter errechnen, wie zum Beispiel Widerstände oder Induktivitäten.

Literatur & Links

[1] COMSOL Multiphysics: http://www.comsol.de/

[2] LTSPICE IV: http://www.linear.com/designtools/software/

Hands-on Workshop BGA-Bestückung

Uwe Dörr <uwe.doerr@eurocircuits.de>

SMD Prototypen-Bestückung von BGA-Bausteinen

Eurocircuits und embedded projects haben ihren ersten Hands on Workshop zur BGA Bestückung am Beispiel eines GNUBLIN Board absolviert. Eingeladen waren alle diejenigen die Interesse am BGA Löten haben. Aber auch allgemeine SMD Bestückung sowie die Inbetriebnahme des GNUBLIN Board unter Linux war ein Teil des Workshops. Somit war das Angebot recht umfassend und ein intensiver Tag konnte beginnen.

Zunächst gab Benedikt Sauter seine Erfahrungen zur Optimierung des GNUBLIN Layouts hinsichtlich der Bestückung bekannt. Anschließend wurden Aspekte des Reflow Löten mit dem Eurocircuits Equipment von Uwe Dörr erörtert. Es wurden wichtige Aspekte zum Lötpastendruck und Reflowlöten gemeinsam mit Claudia Sauter angesprochen und anhand von Anschauungsmaterial besprochen und diskutiert.

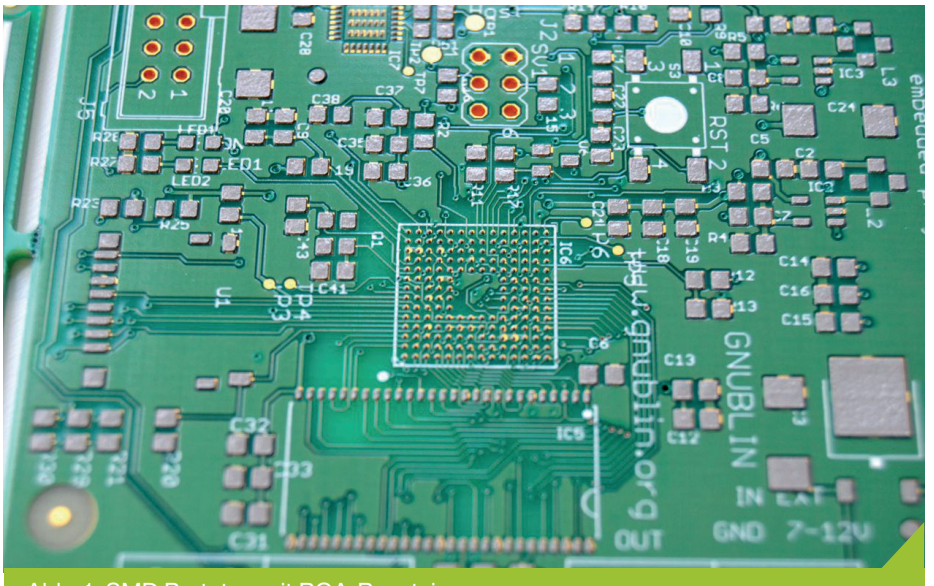


Abb. 1: SMD Prototyp mit BGA-Baustein

Hands-on Workshop bei embedded projects

Nach einer kurzen theoretischen Phase wurde dann der Temperatursensor aufgebaut, weder der Lötpastendruck noch die Bestückung bereitete den anwesenden Teilnehmern hier ein Problem.

Danach ging es nahtlos über zum Aufbau des GNUBLIN Boards. Zunächst erklärte Claudia Sauter alle Tricks und Erfahrungen die sie im Laufe der Jahre bezüglich des BGA Bestücken gesammelt hat. Anschließend mussten ALLE Teilnehmer ihren BGA unter der Anleitung und Kontrolle von Claudia Sauter selbst manuell platzieren. Alle weiteren Bauteile wie Standard Wiederstände und Kondensatoren durfte jeder Teilnehmer von einem Bestückungsautomat durchführen lassen.

Nun kamen wir zu den spannendsten Momenten, denn jetzt ging es darum alle Baugruppen unter der Anleitung von Benedikt in Betrieb zu nehmen. Das Ergebnis war, dass alle 10 Baugruppen sofort funktionierten und somit auch alle BGA und QFN korrekt verlötet waren.

An der Stelle nochmals Danke und großes Lob an Claudia und Benedikt mit welcher Ruhe und Erfahrung dieses Ergebnis zu Stande kam. Dafür, dass keiner der anwesenden Teilnehmer bisher eine BGA selbst gelötet hat ein für mich unfassbares Ergebnis.

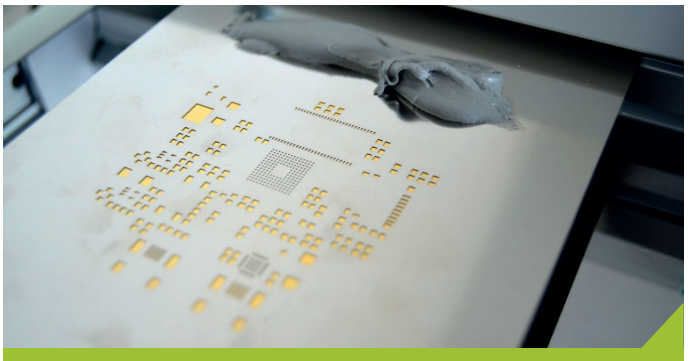


Abb. 2: Lötpastenschablone auf GNUBLIN-SMD-Prototyp

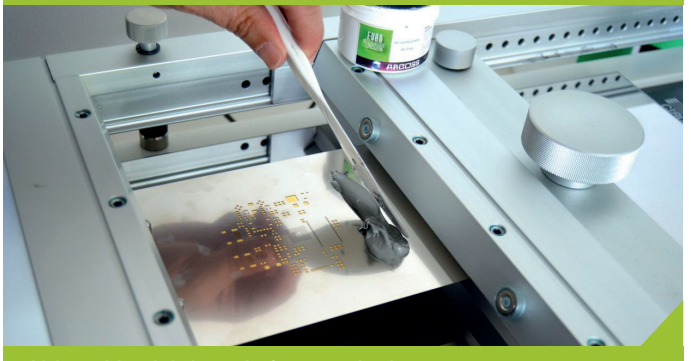


Abb. 3: Vorsichtiges Auftragen der Lötpaste

Einblicke in die Linux Welt

Am Schluss gab Benedikt Sauter noch Einblicke in die Linux Welt. Die Plattform GNUBLIN (embedded Linux) ist als Ersatz für den klassischen Mikrokontroller entwickelt worden. Einfach können Steuerungen und Regelungen implementiert und schnell mit USB oder Webserver ergänzt werden. Der günstige und stromsparende Prozessor von Gnublin kann leicht mit 8-Bit Prozessoren mithalten.

Alle Schaltungen und Layouts können frei von der Gnublin Homepage verwendet und auch in eigene Produkte integriert werden.

Abb. 4: Gespanntes Warten auf die Lötergebnisse

Abb. 5: Reflow-Löten im eC-reflow-mate

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

elektor GOLD membership

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

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

UMWELTSCHONEND – GÜNSTIG – GREEN

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

elektor GREEN membership

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

BTM-222 - Kommunikation mit Bluetooth

Artikelwettbewerb - Mikrocontroller.net [1]

Mit einem Bluetooth-Modul kann eine Schaltung einfach um virtuellen seriellen Port erweitert werden, über den dann drahtlos per Bluetooth Daten mit der Schaltung ausgetauscht werden können. Da heutzutage alle Computer, Laptops, Smartphones und Tablet-PCs etc. über eine integrierte oder nachrüstbare Bluetoothfunktion verfügen, kann so von diesen Geräten aus ein jedes Programm, welches über eine (virtuelle) serielle Schnittstelle kommunizieren kann, ohne weitere Modifikationen transparent auch mit der jeweiligen Schaltung Daten austauschen. Vor allem bei fehlender Unterstützung von USB oder Treiberproblemen bietet Bluetooth eine relativ günstige und flexible Alternative. Zudem entfällt eine kabelgebundene Verbindung und es sind Reichweiten bis zu 100 m möglich.

Bluetooth-Modul

Es gibt am Markt eine Reihe verschiedener Module, welche verschiedene Kriterien erfüllen, die je nach Anwendung relevant sein können, wie zum Beispiel:

- Klasse: Diese gibt die Leistung und Empfindlichkeit des Moduls an. Klasse 1 ermöglicht unter idealen Bedingungen eine Reichweite von bis zu 100 m.
- Datenrate: Wie schnell können Daten empfangen und gesendet werden.
- Profil: Daten zwischen Bluetooth Geräten werden durch so genannte Profile ausgetauscht, die für bestimmte Anwendungsbereiche festgelegt sind. Für Mikrocontrolleranwendungen dürfte vor allem SPP (Serial Port Profile: Serielle Datenübertragung) wichtig sein.
- Antenne: Ist diese integriert oder ist eine externe notwendig.
- Versorgungsspannung: Diese liegt bei allen gängigen Modellen bei ca. 3 V.
- Verfügbarkeit und Preis: Ist das Modul frei verfügbar und zu welchen Kosten?

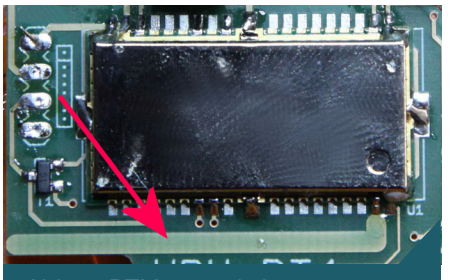


Abb. 1: BTM-222 mit Antenne

Funktionsüberblick des BTM-222

Aus der Vielzahl an Angeboten sticht vor allem das Modul BTM-220/BTM-222 der Firma Rayson hervor, da es bei kleinen Abmessungen leistungsfähig ist, relativ einfach in eigene Schaltungen integriert werden kann und zu einem relativ günstigen Preis frei verfügbar ist. Die wichtigsten Parameter können der Tabelle 1 entnommen werden.

Das Modul befindet sich auf einer Platine mit den Abmessungen 28,2 x15,0 mm. Die gesamte Schaltung ist durch eine Metallhaube geschützt und abgeschirmt. Die Lötkontakte befinden sich an den beiden Längsseiten in Form von durchkontaktierten Halbkreisen. Mit einer feinen Lötspitze läßt sich das Modul auf SMD Pads auch von Hand auflöten. Eine Library des Moduls inkl. Device, Package und Modul ist für Eagle verfügbar [2].

Soll das Modul lediglich zur seriellen Datenübertragung zwischen einem Mikrocontroller und einem Empfänger genutzt werden, ist im Grunde keine externe Beschaltung notwendig und auch keine Konfiguration des Moduls erforderlich. Das Modul sendet transparent die Daten vom seriellen Eingang per Bluetooth weiter und leitet empfangene Daten an den seriellen Ausgang. Die serielle Schnittstelle ist mit den Standardeinstellungen 8N1 vorkonfiguriert: 19.200 bps Baudrate, 8 Datenbits, Keine Parität, 1 Stopbit.

Über Steuerbefehle, den sogenannten AT-Befehlen, kann das Verhalten des BTM-222 aber angepaßt werden. Zusätzlich bietet der BTM-222 auch neben der seriellen UART Schnittstelle noch SPI, USB und PCM. Es sollte vermieden werden, Daten per UART an den BTM-222 zu senden, wenn dieser keine Bluetoothverbindung besitzt, da es in diesem Fall zu Abstürzen des Moduls kommen kann.

Da das Handbuch einige Lücken aufweist und ungenügend ist, gibt es in Tabelle 2 einige Ergänzungen dazu. Das Modul verfügt über frei programmierbare Ein- und Ausgänge. Wie diese programmiert werden können, ist nicht bekannt. Einige Ports sind bereits werksseitig vorbelegt. Die tatsächliche Belegung weicht aber teilweise vom Datenblatt ab.

Name	Hersteller	Klasse	Max Reichweite	Datenrate	Profile	Antenne	Vss	Datasheet	Preis
BTM-222	Rayson	1	100 m	921 kbps	SPP	extern	3-3,6 V	Link [2]	ca. 10...11 €

Tabelle 1: Übersicht der wichtigsten Daten des BTM-222 Bluetooth Moduls

Elektronische Beschaltung

Der BTM-222 arbeitet in einem Spannungsbereich von 3,0 bis 3,6 V und eignet sich daher gut für den Einsatz in Schaltungen, die in diesem Spannungsbereich funktionieren. Bei 3,6V ist eine Schottky-Diode vor PVCC zu empfehlen. Soll er an einen mit 5 V betriebenen Mikrocontroller angeschlossen werden, ist eine Separate Spannungsversorgung und eine Anpassung der seriellen Datensignale mit einem Pegelwandler notwendig.

Der BTM-222 verfügt über 10 frei programmierbare I/O-Pins, denen verschiedene Funktionen zugewiesen werden können. Die Werkskonfiguration sieht für PIO5 den Anschluß einer LED vor, die signalisiert, wenn Daten über das Modul ausgetauscht

werden. Eine LED an PIO7 signalisiert durch blinken die Bereitschaft, eine Bluetoothverbindung aufzubauen und durch konstantes leuchten, daß das Modul mit einem anderen Bluetothmodul gekoppelt (verbunden) ist. Da das Modul über keine integrierte Antenne verfügt, ist eine externe vorzusehen. Die Länge der Antenne kann einfach berechnet werden: $\lambda = c / f$.

Wobei c die Ausbreitungsgeschwindigkeit (in diesem Fall die Lichtgeschwindigkeit im Vakuum) und f die Frequenz der Welle ist (bei Bluetooth zwischen 2,402 und 2,480 GHz). Setzt man für c = 29979245800 cm/s und für f = 2,441GHz ein, erhält man für λ = 12,282 cm. Diese Antenne kann aber kürzer ausfallen, so dass eine Antenne mit einem Viertel der Länge ausreicht: $l = \lambda/4 = 3{,}07$ cm.

Die Antenne kann dabei sehr einfach in Form einer Leiterbahn auf einer gedruckten Schaltung ausgeführt sein. Bei der üblichen Kupferstärke von 35 µm hat sich eine Leiterbahn von 31 mm Länge und 1,78 mm Breite bestens bewährt, wobei auch schmalere Breiten bis hin zu ca. 0,4 mm umsetzbar sind.

GPIO	Pin	Funktion	
#5	11	(LED gegen Masse) signalisiert Austausch von Daten per Bluetooth	
#6	12	(LED gegen Masse) Bluetoothkopplung (Low = verbunden, High = getrennt)	
#7	13	(LED gegen Masse) signalisiert Status der Bluetoothkopplung	
		Signalform	Bedeutung
		Aus	Bluetoothverbindung getrennt
		permanent High	Bluetoothverbindung besteht (Pairing)
		schnell blinkend (0,1s)	wartet auf Verbindung, Slave Mode
		schnell blinkend (0,3s)	Erkennbar, wartet auf Verbindung, Slave Mode
		langsam blinkend (0,9s)	Anfrage, Master Mode
		sehr langsam blinkend (1,2s)	Verbinde, Master Mode

Tabelle 2: Ergänzende Informationen zum Datenblatt über die GPIO Pins

Treiber

Die Treiber für den Bluetoothadapter sind auf dem Gerät gemäß der Anleitung vom Hersteller des Adapters einzurichten. Je nach Betriebssystem läuft dann die Einrichtung und der Verbindungsaufbau ein wenig abweichend ab. Die Bluetoothadapter richten aber stets einen Dienst oder ein Programm ein, mit dem nach anderen Bluetoothgeräten in der Nähe gesucht werden kann. Befindet sich eine Schaltung mit BTM-222 Modul in Reichweite, wird das Gerät als Serieller Adapter (Serial Adaptor) erkannt und eine Verbindung kann aufgebaut werden. Damit Bluetoothgeräte vor dem Zugriff von Unbefugten geschützt werden können, muß auf dem Endgerät ein Paßwort/Schlüssel angegeben werden, der dem BTM-222 signalisiert, daß es sich um einen autorisierten Zugriff handelt. Werksseitig lautet der Schlüssel 1234. Über die AT-Befehle des BTM-222 kann der Schlüssel aber auch geändert werden.

Mit einem einfachen Terminalprogramm können Daten an das BTM-222 Modul gesendet werden und empfangene Daten werden angezeigt. Bei einem virtuellen COM-Port hat die Baudrateneinstellung keine Bedeutung, so daß eine Baudrate von beispielsweise 19.200 bps und 8N1 als Übertragungsparameter gewählt werden können.

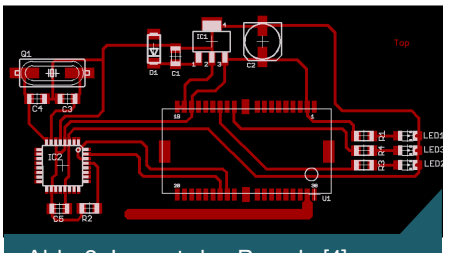


Abb. 2: Layout des Boards [4]

Beispielapplikation

Die gezeigte Schaltung beschränkt sich auf die minimal notwendige Auswahl an Bauelementen, um einen ATmega328 Mikrocontroller bei 3,3 V zu betreiben und dessen serielle Schnittstelle mit einem BTM-222 zu verbinden, so daß über die Schnittstelle Daten gesendet und empfangen werden können. Da die Schaltung mit 3,3 V arbeitet (welche aus einem Linearregler erzeugt wird), ist keine Anpassung der Signale zwischen Mikrocontroller und BTM-222 erforderlich. LED1 signalisiert die Betriebsbereitschaft der Schaltung, LED2 signalisiert den Status der Bluetoothverbindung und LED3 blinkt, wenn Daten in eine beliebige Richtung übertragen werden.

Da der BTM-222 bereits ab Werk so eingestellt ist, daß Daten an seinem UART Eingang per Bluetooth gesendet und per Bluetooth empfangene Daten am UART Ausgang ausgegeben werden, ist keinerlei Konfiguration seitens der Firmware auf dem ATmega notwendig. Die Software im Mikrocontroller braucht lediglich eine UART Schnittstelle zu öffnen und darüber Daten mit den oben genannten Parametern zu senden und zu empfangen. Der BTM-222 kümmert sich autark um die Bluetoothübertragung. Wie die UART Kommunikation im Mikrocontroller umgesetzt werden kann, ist u. a. im AVR-Tutorial und im AVR-GCC-Tutorial beschrieben.

Die exemplarisch gestaltete Platine ist durchgängig in SMD Bauweise bestückt. Sie ist nicht hinsichtlich des Platzes und der Bauteilanordnung optimiert, sondern mehr unter dem Gesichtspunkt der funktionellen Gruppierung der Bauteile zum besseren Verständnis. Besonderes Merkmal ist die als Leiterbahn ausgeführte Antenne. Weiterhin wurden nicht alle am BTM-222 vorhandenen Masseanschlüsse beschaltet, sondern lediglich die beiden an Pin 18 und 19. Da an Pin 19 auch das Metallgehäuse des Moduls angelötet ist, ist somit eine ausreichende Masseverbindung für das Modul gewährleistet. Da auf der Unterseite des BTM-222 blanke Leiterbahnen existieren, ist darauf zu achten, daß Leiterbahnen, die unterhalb des Moduls verlegt sind, gegen das Modul hin isoliert sind (z. B. Lötstoplack).

Literatur & Links

[1] http://www.mikrocontroller.net/articles/BTM-222

[2] Device Package für den BTM-222: http://www.mikrocontroller.net/topic/94558#930456

[3] Datenblatt des BTM-222: http://www.mikrocontroller.net/wikifiles/f/fc/BTM222_DataSheet.pdf

[4] Schaltplan und Layout in Eagle: http://www.mikrocontroller.net/wikifiles/4/43/Btm222.zip

Marktplatz / Neuigkeiten

Die Ecke für Neuigkeiten und verschiedene Produkte

Eagle Schulung für Einsteiger

Heutzutage ist es selbstverständlich, elektronische Schaltungen mit Hilfe einer CAD-Software zu erstellen. Eine Software hat sich bereits in vielen Einrichtungen als sinnvoll erwiesen und etabliert: Die CAD-Software Eagle.

Am 10. Dezember 2012 haben wir damals zum ersten Mal gemeinsam einen EAGLE Workshop mit der Firma CadSoft und Eurocircuits in den Räumlichkeiten von embedded projects in Augsburg veranstaltet. Trotz plötzlichen Wintereinbruchs haben sich 14 interessierte Teilnehmer eingefunden. Da die Nachfrage sehr hoch ist, haben wir uns dazu entschlossen, eine weiter Schulung in diesem Format anzubieten.

Programm:

- Einführung in das Schaltungs- und Boarddesign mit Eagle
- Einführung in die Bauteilbibliotheken und mehrlagige Layouts
- Mittagspause
- Erstellung von eigenen Bauteilen
- Einführung in PCB Visualizer und PCB Checker von Eurocircuits
- Wer möchte kann am darauffolgenden Tag auch zur Bestückung des Gnublin Boards dabei sein. (siehe unten)

Veranstaltungsort/-termin:

Augsburg 16.09.2014, 09.00 bis ca. 17.00 Uhr bei embedded projects GmbH

Teilnahmegebühr (inkl. MwSt.): 349 €

Anmeldung: info@embedded-projects.net

SMD Prototypen-Bestückung von BGA-Bausteinen am Beispiel des Embedded Linux Boards GNUBLIN

Immer häufiger werden in Schaltungen insbesondere Linux-Boards SMD Bausteine oder BGAs verwendet. Moderne Prozessoren, AD-Wandler oder neue Speicher werden oft nur in sehr kleinen Bauformen angeboten. Dort, wo für Prototypen früher der Lötkolben und Lötzinn gereicht haben, benötigt man jetzt SMD-Lötpaste, eine passende Schablone und etwas Erfahrung bzw. die notwendigen Tricks um solch ein Projekt erfolgreich durchzuführen.

Oft hört man von Entwicklern, die noch zu großen Respekt vor diesen Bausteinen haben und sich deshalb an solche Bauteile gar nicht erst herantrauen.

In diesem Workshop werden wir gemeinsam anhand praktischer Beispiele demonstrieren, wie man solche Hürden überwinden und Prototypen mit vermeintlich schwierigen Bausteinen per Hand bestücken kann.

Jeder Teilnehmer wird ein eigenes BGA Board aufbauen und anschliessend in Betrieb nehmen.

In der verwendeten Schaltung kommt ein BGA, QFN und TSOP54 Gehäuse zum Einsatz.

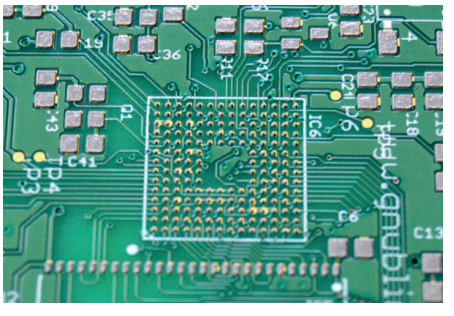
Ziel ist es, einen BGA ohne großen Aufwand löten zu können.

Jeder Teilnehmer wird mit Hilfe von unseren Experten seine eigene Platine bzw. BGA-Bausteine bestücken,das Board im Ofen löten und anschließend in Betrieb nehmen.

Programm:

- Einführung SMD Reflow löten / Vorstellung Equipment / Tips fürs Layout
- Vorführung diverser Techniken (SMD Schablonen-Druck, Löten, Platzieren)
- Praktische Einheit 1: Aufbau des Temperatursensors (Handbestückung mit LM75)
- Mittagspause
- Praktische Einheit 2: Aufbau des Embedded Linux Boards Gnublin
- Optional Rest-Bestückung mit SMD-Automat
- Optische Kontrolle / Reflow Lötgang
- Inbetriebnahme des Linux Boards
- Fragen / Diskussion / Erfahrungsaustausch

Max. 10 Teilnehmer



Veranstaltungsort/-termin:

Augsburg 17. und 18.09.2014, 09.00 bis ca. 17.00 Uhr bei embedded projects GmbH

Teilnahmegebühr (inkl. MwSt.): 349 €

Im Preis sind jeweils Material, Seminarunterlagen, Mittagessen sowie Dokumentation und Aushändigung eines Teilnahmezertifikats inbegriffen.

Jeder Teilnehmer erhält:

- Platine + Material für 1 x GNUBLIN und 1 x Temperatursensor
- Stencil Fix zum einfachen Bedrucken von SMD / BGA Platinen
- Handout

Interesse an einer Anzeige?

info@embedded-projects.net

embedded-projects
JOURNAL
OPEN SOURCE SOFT-AND HARDWARE PROJECTS

Werdet aktiv!

Das Motto: Von der Community für die Community !

Das Magazin ist ein Open Source Projekt.

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

journal@embedded-projects.net

Regelmäßig

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

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

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

Sponsoren gesucht!

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

Bitte frei machen

embedded projects GmbH
Holzbachstraße 4
D - 86152 Augsburg

Name / Firma

Straße / Hausnummer

PLZ / Ort

Email / Telefon / Fax

◯ Ich möchte jede zukünftige Ausgabe erhalten

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

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

Impressum

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

Verteilt durch:

embedded projects GmbH
HARDWARE FOR PROJECTS

Veröffentlichung: 4x / Jahr
Ausgabenformat: PDF / Print

Auflagen Print: 2500 Stk.
Einzelverkaufspreis: 1 €

Layout / Satz: EP
Druck: flyeralarm GmbH
Titelbild: elektor

MIT FLEXIBILITÄT MEHR BEWEGEN.

FLEXIBLE LEITERPLATTEN ONLINE BESTELLEN.

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

LEITON
RECHNEN SIE MIT BESTEM SERVICE

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

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