Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Johann-Nikolaus Andreae

Das Lenkraddisplay eines Formula Student Rennwagens: Von der Analyse, über die Hardware- und Linuxtreiberentwicklung bis zum Prototypen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Johann-Nikolaus Andreae

Das Lenkraddisplay eines Formula Student Rennwagens: Von der Analyse, über die Hardware- und Linuxtreiberentwicklung bis zum Prototypen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Franz Korf
Zweitgutachter : Prof. Dr. rer. nat. Stephan Pareigis

Abgegeben am 21. August 2008

Johann-Nikolaus Andreae

Thema der Bachelorarbeit

Das Lenkraddisplay eines Formula Student Rennwagens: Von der Analyse, über die Hardware- und Linuxtreiberentwicklung bis zum Prototypen

Stichworte

Lenkrad, Display, Mikrocontroller, ARM9, Hardware, Linux, Treiber, CAN-Bus, Platinenlayout, Formula Student

Kurzzusammenfassung

Das Thema dieser Bachelorarbeit ist die Entwicklung eines Lenkraddisplays für einen Formula Student Rennwagen. Die Arbeit beschäftigt sich mit der Hardware und dem Betriebsystem. Im Hardwareteil wird ein ARM9 basiertes Mikrocontroller-System entwickelt, welches im Softwareteil mit einem Linux-Betriebsystem in Betrieb genommen wird. Der Datenaustausch mit dem Auto findet über den CAN-Bus statt.

Johann-Nikolaus Andreae

Title of the paper

Steering wheel display for a Formula Student racing car: from the analysis over to the hard- and linux driver development to the prototyp

Keywords

steering wheel, display, microcontroller, ARM9, hardware, linux, driver, CAN-Bus, Printed Circuit Board (PCB), Formula Student

Abstract

The topic of this bachelor thesis is the development of a steering wheel display for a Formula Student racing car. It deals with the hardware and the operating system. In the hardware part it describes the development of an ARM9 based microcontroller system. The software part adds a linux based operating system. The communication with the car uses the CAN-Bus.

Inhaltsverzeichnis

1 Einführung

Um einen Rennwagen zu optimieren werden eine Unmenge von Messdaten benötigt. Diese werden zu unterschiedlichen Zeiten ausgewertet. Einige während der Fahrt vom Fahrer und vom Begleitteam, andere hinterher bei der Verbesserung der Konstruktion. Diese Daten zur Verfügung zu stellen, ist die Aufgabe der Telemetrie.

Diese Bachelorarbeit beschäftigt sich mit der Hardware und dem dazugehörigen Betriebssystem für die Anzeige der Daten während der Fahrt.

Abbildung 1.1: Hawk08 beim Endurence in Hockenheim

1.1 Motivation

Im Hawk07 waren zwei Telemetriesysteme verbaut. Das eine in Form eines zugekauften Lenkraddisplays (siehe Abb. 1.2), welches Motordaten, Beschleunigung und Raddrehzahl

anzeigte und aufzeichnete. Das andere, ein von Sebastian Haase (2007) und Simon Schuckert (2007) entwickeltes System (siehe Abb. 1.3), übermittelte die Daten an den Kontrollstand und zeichnete sie auf.

Abbildung 1.2: Lenkraddisplay des Hawk07

Ein Zusammenführen der Daten ist nur schwer möglich, da es keine Bezugszeit zwischen den Systemen gibt. Es besteht nur eine Möglichkeit durch die Daten der Motorsteuerung, die Daten in Relation zueinander zu setzen. Diese stehen beiden Systemen zur Verfügung und digital übermittelt werden. Die Vorderradsensoren stehen ebenfalls beiden Systemen zur Verfügung können aber durch Messungenauigkeiten geringfügig abweichen, so dass sie sich nur schwer vergleichen lassen. Weitere Nachteile des zugekauften Displays sind das proprietäre Datenformat und die vorgegebenen Darstellungsmöglichkeiten auf dem Display. Es lässt sich nur mit der mitgelieferten Software auswerten. Auch ist eine Erweiterung um weitere Sensoren nicht möglich.

Bei der Planung für den neuen Rennwagen Hawk08 wurde beschlossen, ganz auf das neue System zu wechseln. Hierzu mussten folgende Komponenten neu Entwickelt werden: Beschleunigungs- und Drehratensensor, Lenkraddisplay, System zur Datenauswertung. Mit dem Lenkraddisplay beschäftigt sich diese Arbeit.

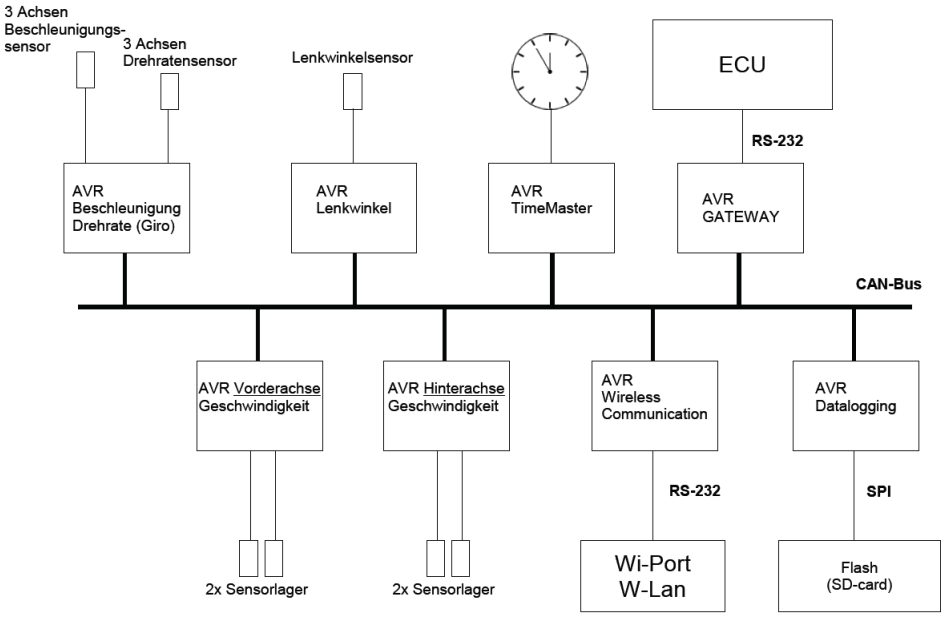


Abbildung 1.3: Entwurf des Telemetriesystems von Schuckert (2007)

1.2 Formula Student

Formula Student ist ein Wettbewerb für Studenten. Bei diesen Wettbewerb geht es darum einen Prototypen für einen Rennwagen für Hobbyrennfahrer zu bauen. Es geht nicht nur darum, das schnellste Auto und den besten Fahrer zu haben, sondern es geht um das beste Gesamtkonzept. Bei der Konstruktion sind strenge Regeln SAE (2007) zu beachten. Diese betreffen hauptsächlich die Sicherheit des Fahrzeuges. Am Anfang des Wettbewerbs steht die technische Abnahme bei der das Auto auf Regelkonformität geprüft wird. Ohne die technische Abnahme darf das Auto nicht auf die Rennstrecke.

Die Bewertung der Teams findet in unterschiedlichen Kategorien statt. Es gibt statische und dynamische Events. Jedes Event wird einzeln Bewertet. Das Team mit der höchsten Gesamtpunktzahl gewinnt.

1.2.1 Statische Events

Für die Statischen Events ist es nicht erforderlich, das das Auto die technische Abnahme bestanden hat.

Engineering Design Beim Design-Report wird die Konstruktion des Autos bewertet. Die Baugruppenleiter erklären den Juges die Technischen Errungenschaften des Autos.

Cost Analysis Beim Kostreport werden die Kosten, die das Auto in Serienfertigung kosten würde berechnet und bewertet. Hierbei wird von einer Absatzzahl von 1000 Stück im Jahr ausgegangen.

Presentation Hier stellt das Team sein Geschäftskozept für eine Vermarktung des Autos vor.

1.2.2 Dynamisch Events

Skid-Pad Auf dem Skidpad wird das Kurvenverhalten des Autos getestet. Es besteht aus zwei Kreisen die eine Acht bilden. Beide Kreise werden jeweils zweimal durchfahren, wobei das zweite Rundenergebnis gewertet wird.

Acceleration Beim Beschleunigungsrennen wird die Zeit über eine Strecke von 75m ermittelt.

Autocross Der Handlingkurs ist eine kurvenreiche Strecke, bei der man viel beschleunigen und abbremsen muss.

Fuel Economy Hier wird der Bezinverbauch während des Endurance gemessen.

Endurance Die Ausdauerstrecke beträgt 22km, wobei bei der Hälfte der Strecke ein Fahrerwechsel mit abgestelltem Motor stattfindet. Für diesen Wechsel sind max. 3min Zeit.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist die Entwicklung eines Displays für das Lenkrad des Wagens, welches die für den Fahrer wichtigen Daten anzeigt. Das Lenkraddisplay besteht aus drei Teilen: Der Hardware, dem Betriebsystem und der Software für die Darstellung. Letzteres wird in der Bachelorarbeit von Lorenz (2008) behandelt. Das Lenkraddisplays soll direkt an den CAN-Bus (siehe Abb. 1.3) angeschlossen werden. Es stehen also für die Anzeige alle Sensoren, die sich im System befinden zur Verfügung. Die Architektur des Lenkraddisplays soll so ausgelegt werden, dass es später um weitere Funktionalitäten erweitert werden kann.

2 Analyse

In diesem Kapitel werden die Anforderungen beschrieben, die an ein Lenkraddisplay (SW-CU) für einen Formula Student Rennwagen gestellt werden. Anschließend werden in der Marktanalyse die passenden Komponenten für das Lenkraddisplay gesucht.

2.1 Anforderungen

Die Anforderungen basieren zum einen auf den technischen Begebenheiten zum anderen aus den Wünschen des Teams. Die Anforderungen, die die Darstellung auf dem Display und die Usebility betreffen, werden von Daniel Lorenz (2008) in seiner Bachelorarbeit behandelt.

2.1.1 Umwelteinflüsse

Der Einsatzort der Hardware ist das Lenkrad eines Renwagens. Dieses Lenkrad ist nicht vor den Wettereinflüssen geschützt. Es muss also mit Regen und starker Sonneneinstrahlung gerechnet werde. Um das Eindringen von Wasser in das Lenkrad zu verhindern, müssen Dichtungen an den Löchern vorgesehen werden. Durch die Sonneneinstrahlung können höhere Temperaturen im Lenkradinneren entstehen und außerdem ist die Ablesbarkeit des Displays beeinträchtigt.

In einem Auto ist die Elektronik Vibrationen durch den Motor und Stößen durch Unebenheiten in der Straße ausgesetzt. Mechanische Verbindungen müssen so ausgelegt werden, dass sie dieser Beanspruchung standhalten.

Das Lenkrad ist beim Hawk08 nicht mehr durch ein Kabel an das Fahrzeug gebunden. Nach dem Abnehmen des Lenkrades, was zum Aus- und Einsteigen nötig ist, kann das Display beim Ablegen des Lenkrades beschädigt werden.

Hardwareanforderungen

- Schutz vor Eindringen von Feuchtigkeit
- Vibrationsfeste Verbindungen
- Schutz des Displays vor Beschädigung beim Ablegen des Lenkrades
- Temperaturunempfindlichkeit der Bauteile
- Keine hohe Eingenwärmeentwicklung

2.1.2 Abmessungen

Die SWCU soll im Lenkrad des Wagens montiert werden. Das Lenkrad wird aus Karbon gefertigt. Die beiden Hälften werden nach dem Laminieren miteinander verklebt und lassen sich somit nicht öffnen. Das Ein- und Ausbauen des Displays muss also entweder durch das Displayloch oder durch ein extra Loch im Lenkrad erfolgen. Die Maße des zur Verfügung stehenden Platzes, sowie die Position der Taster und die Position der Befestigungsschrauben müssen mit der Baugruppe Interieur abgesprochen werden.

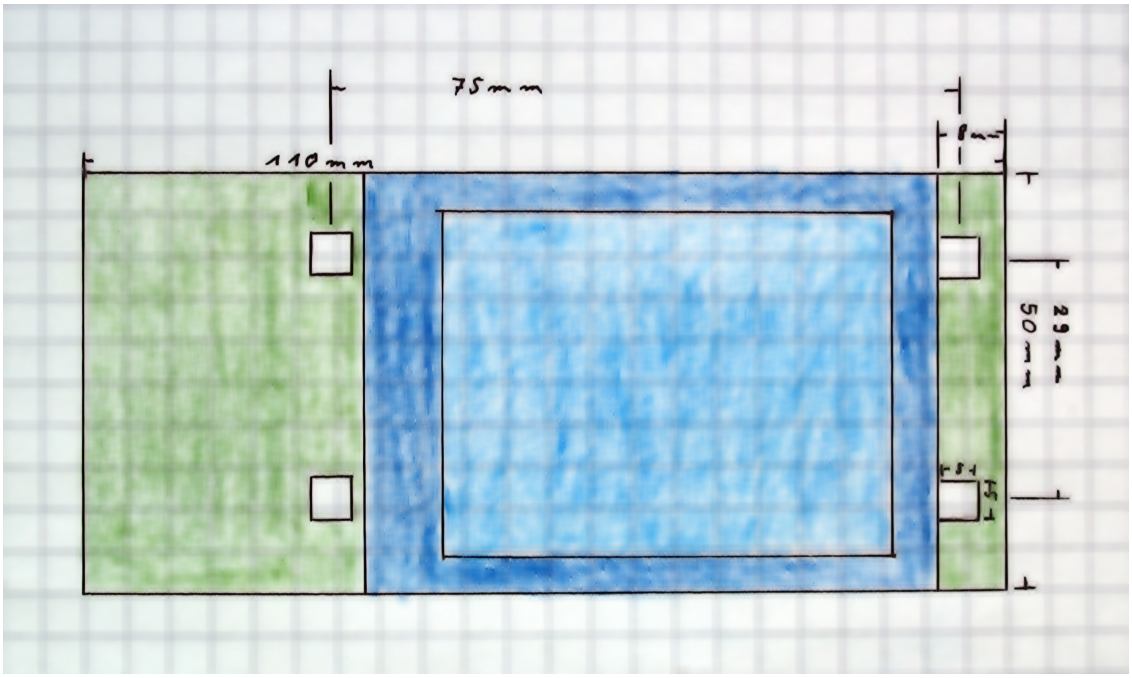


Abbildung 2.1: Skitze mit dem Abmessungen wie sie zu begin des Projektes mit dem Interieur abgesprochen wurden. Die viereckingen Löcher sind für die Befestigungsbolzen (siehe Kap. 3.1.8). Grün: Platine, Blau: Display (Sichtbereich/Rahmen)

Zu Beachten ist auch die Position von Display und Tastern. Diese muss beim Platinenlayout Berücksichtigt werden.

Hardwareanforderungen

- Platinen Maße max. 50x110mm bei Einführung über Displayloch
- Platinen Maße max. 75x120mm bei Einführung über extra Loch an der Ober- oder Unterseite des Lenkradmittelteils

2.1.3 Stromversorgung

Der Strom für die SWCU wird direkt aus dem Bordnetz bezogen. Die Spannung ist Schwankungen unterworfen. Sie sinkt beim Starten des Motors ab und ist bei laufendem Motor höher als bei Batteriebetrieb. Des weiteren gibt es Schwingungen vom Motor in der Stromversorgung.

Die Strom- und Datenversorgung der SWCU wird im H04 über einen Stecker im Quickrelease der Lenkradanbindung realisiert. Zum Ein- und Aussteigen muss das Lenkrad abgenommen werden. Es wird also die Stromversorgung bei jedem Ein- oder Aussteigen unvermittelt getrennt. Hier muss über eine Optimierung der Bootzeit und eventueller Pufferung des Stroms für das Aus- und Einsteigen nachgedacht werden.

Hardwareanforderungen

- Eingangsspannung 8-15V
- Störungsunemfindlichkeit
- Pufferung des Stroms wenn möglich. Für den Endurance wäre eine Überbrückungszeit von 3min sinnvoll (siehe 1.2.2).

Softwareanforderungen

- Kurze Bootzeiten
- Standby Modus

2.1.4 Schnittstellen

2.1.4.1 CAN-Bus

Das Lenkraddisplay soll die Daten aus dem bestehenden Time Triggert CAN-Bus (TTCAN) System empfangen können. Für spätere Erweiterungen soll die Möglichkeit bestehen, time-triggert Daten auf den CAN-Bus zu verschicken.

Beim TTCAN werden die Nachrichten nicht wie beim CAN-Bus üblich priroritätsgesteuert verschickt, sondern jede Nachricht erhält ihren exklusiven Zeitslot. Während das Empfangen von Nachrichten keinen Unterschied macht, stellt das Versenden besondere Anforderungen an die Hardware. Die Nachrichten müssen zu einem bestimmten Zeitpunkt bezüglich der Referenznachricht verschicktwerden. Des weiteren dürfen sie im Falle eines Fehlers nicht wiederholt werden (retransmitt). Weitere Informationen über den Aufbau des TTCAN im Hawks-Rennwagen sind in den Bachelorarbeiten von Simon Schuckert (2007), Sebastian Haase (2007) und Felix Kolbe (2008) zu finden.

Hardwareanforderungen

- CAN-Controller
- TTCAN Unterstüzung für spätere Erweiterungen

Softwareanforderungen

- CAN-Treiber
- Erweiterbar um TTCAN-Unterstüzung

2.1.4.2 TFT-Display

Durch ein Sponsoring seitens der Firma DATA MODUL standen uns zwei TFT 2,7” Displays von Hitachi zur Verfügung. Das Display wurde nach Usebility Gesichtspunkten und dem zur Verfügung stehenden Platz ausgewählt.

Bei der Auswahl des Displays wurde entschieden, dass ein Touchscreen keinen Sinn macht, da er nur schwer mit Handschuhen, die der Fahrer tragen muss, zu bedienen ist.

Abmaße	
Außenmaße	50,54x68,62x2,6mm
Bildfläche	41,04x54,72mm
Bilddaten	
Auflösung	240x320 Pixel
Farben	262.000
Bildwiederholfrequenz	60Hz
Hintergrundbeleuchtung	LED
Leuchtstärke	420 cd
Kontrast	300:1
Schnittstelle	
Schnittstelle	6bit pro Farbe TTL
Stromversorgung	3,3V; 5V; 15V; -15V
Stecker	Hirose FH23-45S-0.3SHAW

Tabelle 2.1: Technische Daten Display

Hardwareanforderungen

- TFT-Display-Controller
- Stromversorgung mit den Spannungen 3,3V, 5V, 15V und -15V
- Stecker Hirose FH23-45S-0.3SHAW an der richtigen Position

Softwareanforderungen

- Display-Treiber

2.1.4.3 LED's

Zum Anzeigen des optimalen Schaltpunktes, sowie für Warnmeldungen werden oben in den Lenkradkranz rechts und links je zwei helle Duo-Color-LED's eingelassen. Siehe Abb. 2.3. Die LED's werden symmetrisch (innen/außen) beschaltet. Rechts und Links soll nicht separat ansteuerbar sein. Es sind also 2x2 LEDs die separat angesteuert werden müssen. Optional soll die Helligkeit der LEDs regelbar sein. Die Farben und Funktionen der LED's werden in der Bachelorarbeit von Daniel Lorenz (2008) festgelegt.

Abbildung 2.2: TFT-Display (im Hintergrund die noch unbestückte Platine)

Hardwareanforderungen

- Anschlüsse für 4 LEDs
- Helligkeitsregelung via PWM (optional)

Softwareanforderungen

- Treiber mit Schnittstelle zum steurn der LEDs

2.1.4.4 Taster

Für die Bedienung des Displays durch den Fahrer werden Taster benötigt. Die Anzahl und Position werden in der Usebility Analyse Lorenz (2008) in Absprache mit der Baugruppe Interieur festgelegt.

Hardwareanforderungen

- Anschlüsse für 3 Taster

Abbildung 2.3: CAD-Bild des Lenkrades

Softwareanforderungen

- Treiber mit event Interface

2.1.5 Betriebssystem

Das Betriebssystem soll für die Displaysoftware eine definierte Umgebung bieten und die Hardwarezugriffe kapseln. Weiterhin soll es möglich sein, das Betriebssystem um Hard-Reatime Funktionalitäten zu erweitern, z.B. zum Übertragen der Signale der Schaltpadels über den CAN-Bus. Alternativ lässt sich diese Funktionalität über einen separaten Mikrocontroller hinzufügen.

Für die reine Anzeigefunktion sind keine Echtzeitanforderungen notwendig. Sollten CAN-Nachrichten verloren gehen, wird dies vom Benutzer nicht bemerkt, da diese spätestens nach $40ms$[1] wiederholt werden. Die Nachrichten treffen max. alle $400\mu s$ ein, wenn alle Zeitslots belegt sind. Kürzere Abstände können durch die feste Sendezeitvorgabe des TT-CAN nicht vorkommen. Dies entspricht einer Auslastung des CAN-Busses von ca. 60%.

[1] 1 Basiszyklus hat $10ms$. Dieser ist in Zeitslots von $400\mu s$ pro Nachricht unterteilt. Die niederfrequenten Nachrichten werden alle 4 Basiszyklen = Matrixzyklus wiederholt. Dies entspricht einer Wiederholfrequenz von $25Hz = 40ms$. Vgl. Zyklenaufbau von Sebastian Haase (2007)

Hardwareanforderungen

- Mikrocontroler

Softwareanforderungen

- Bearbeitungszeit für CAN-Messages $< 400\mu s$
- erweiterbar um harte Echtzeitanforderungen

2.2 Marktanalyse

Um die oben (2.1 Anforderungen) genannten Anforderungen zu erfüllen, wurden die Produkte auf dem Markt untersucht. Für das weitere Vorgehen musste zuerst eine Entscheidung bezüglich der Ansteuerung des Displays getroffen werden.

Die Ansteuerung des Displays ohne Zusatzhardware ist mit den im Auto eingesetzten AVRs nicht sinnvoll möglich. Das Display erwartet die einzelnen Bildpunkte mit einer Geschwindigkeit von 5MHz bis 6,5MHz (siehe Dotclock (DCLK) im Datenblatt Hitachi (2006)). Die Daten mit dieser Geschwindigkeit an die IO-Ports des AVRs (16MHz) anzulegen ist nicht möglich. Es stehen pro Bildpunkt max. $\frac{16MHz}{5MHz} = 3{,}2$ Taktzyklen zur Verfügung. In diesen 3,2 Taktzyklen pro Bildpunkt müsste die Berechnung der Grafik, die Behandlung des CAN-Busses und die IO-Operationen stattfinden. Auch wenn der AVR8 die meisten Operationen in einem Takt abarbeitet ist er zu langsam (Schuckert, 2007, Tabelle 2.1.: Einige wichtige Kenndaten des Atmel AVR AT90CAN128). Es ist also mindestens ein externer Grafikkontroller für die Ansteuerung nötig.

Folgende Lösungsansätze wurden untersucht:

1. Verwendung der bis jetzt im Auto eingesetzten AVR8 Prozessoren
   a) mit einem externen Displaycontroller z.B. Epson S1D13705
   b) mit einem selbstprogrammierten CPLD / FPGA
2. Verwendung eines SoC mit integriertem Displaycontroller.

Wir entschieden uns einen Mikrocontroller mit integriertem Displaycontroller zu suchen. Die höhere Rechenleistung dieser Mikrocontroller schafft mehr Freiheiten in der grafischen Gestaltung des Userinterfaces, da man weniger auf die begrenzten Ressourcen achten muss.

	AVR		*SoC*
	Displaycontroller	*CPLD / FPGA*	*int. Displaycontroller*
Beschaffung	schwierig	einfach	mittel
Programmierung Treiber	mittel/schwierig	mittel/schwierig	einfach
Entwicklung Controller	-	schwierig	-
Möglichkeiten Grafik	begrenzt	begrenzt	groß

Tabelle 2.2: Vergleich der Möglichkeiten

2.2.1 Betriebssysteme

Das Betiebsystem stellt die grundlegenden für den Betrieb des Rechners notwendigen Funktionen zur Verfügung. Das Betriebssystem hat zwei Aufgaben: Zum einen die „Veredelung der Hardware“ zum anderen die „Verwaltung der Ressourcen“ Tanenbaum (2003).

Es gibt verschiedene Arten von Betriebsystemen. Diese Auflistung lehnt sich an die Auflistung (Tanenbaum, 2003, Unterschiedliche Arten von Betreibssystem (S.30)) an. Einige Kathegorien werden aber zusammengefasst z.B. sind die meisten Desktopsysteme milerweile Multiprozessorsysteme. Nicht alle Betriebsysteme lassen sich einem genauen Typen zuordnen.

Server/Mainframe In diese Kategorie fallen Systeme die einen Dienst zur Verfügung stellen. Sie sind dafür ausgelegt, viele Benutzer gleichzeitig zu bedienen oder große Ressourcenmengen zu verwalten.

Embedded Unter embedded Systemen fasse ich alle Systeme zusammen, die nicht von jedem als Computer erkannt werden. Sie sind auf geringe Größe optimiert. Bei den Betriebssystemen für eingebettete Systeme, handelt es sich um für den jeweiligen Aufgabentyp und die eingesetzte Hardware optimierte Systeme. Sie lassen sich nur für diese Aufgabe, meist auch nur auf genau einer Hardware einsetzen.

Realtime Realtimebetriebsysteme sind auf bestimmte Zeitanforderungen optimiert. Sie erfüllen z.B. harte Echtzeitbedingungen.

Desktop Desktop Betriebssysteme sind für die tägliche Arbeit im Büro oder zu Hause optimiert. Sie bieten meist eine grafische Oberfläche

Spielekonsolen Sind ausschließlich für die die Unterhaltung optimiert. Man kann sie auch als eine Kategorie der embedded Systeme betrachten

Linux ist ein System, das sich in fast allen Kathegorien einsetzen lässt (Die Spielekonsolen nutzt es nicht zum Spielen). Es ist mitlerweile sehr gut anpassbar und läuft auf fast jeder Hardware mit MMU. Aber auch für die Systeme ohne MMU gibt es mit μLinux eine Lösung.

Neben Linux sind die propritären Betriebssystem WindowsCE, vxWorks und QNX auf embedded Systemen verbreitet (Systeme die ausschließlich für Mobiltelefone konzipiert sind wie Symbian wurden hier nicht beachtet).

Betriebssystem	*OpenSource*	*Lizenzkosten*	*Supportkosten*	*Auswahl Grafikbib.*
Linux	ja	nein	teilweise	groß
QNX	nein	ja	ja	klein
vxWorks	nein	ja	ja	klein
WindowsCE	nein	ja	ja	klein

Tabelle 2.3: Vergleich Betriebssysteme für embedded Systeme

Wir entschieden uns für den Einsatz von Linux, weil sich bei Linux zum einen der Arbeitsaufwand sehr gut einschätzen lässt. Man hat direkten Zugriff auf die aktuelle Entwicklung und kann genau sehen, welche Treiber schon existieren. Zum anderen ist man unabhängig vom Support eines Herstellers und es fallen keine Lizenzkosten an. Ein weiterer Grund für die Wahl, ist dass für Linux eine große Auswahl an Grafikbibliotheken zur Verfügung stehen. Unter diesen konnte in Lorenz (2008) die passende ausgesucht werden.

Die harte Echtzeitfähigkeit lässt sich bei Linux mit Hilfe von RTAI[2] nachrüsten. RTAI fügt einen Echtzeit-Kernel zwischen Hardware und Linux-Kernel ein. Siehe Abb. 2.4. Dieser kümmert sich um das Ausführen der Echtzeit-Prozesse. Der Linux-Kernel läuft als Prozess im Echzeit-Kernel mit der geringsten Prirorität. Da der Linux-Kernel nicht mehr direkt auf die Hardware zugreifen kann benötigt es einen Speziellen Reatime-Abstraktions-Layer (RTHAL). Die Kommunikation zwischen Echtzeit-Prozessen und den Linux-Prozessen findet über einen Fifo statt.

Somit lassen sich später bei Bedarf Echzeitanforderungen nachrüsten. Weiter Informationen zu RTAI finden sich unter: Schwebel (2002), Abbott (2006), Bruyninckx (2002) und Opdenacker (2007)

2.2.2 CAN-Teiber

Es gibt verschiedene CAN-Treiber für Linux. Die meisten implementieren ein Character-Device für ihre spezielle Hardware oder wie das can4linux Projekt[4] der Port GmbH für verschiedene Hardware. Ein anderer Ansatz ist, den CAN-Bus als Netzwerk zu betrachten und

[2] https://www.rtai.org/

[3] Bild: Niklaus Burren Lizenz: GNU FDL Quelle: Wikipedia http://de.wikipedia.org/wiki/Bild:Architekturrtai.jpg

[4] http://www.port.de/deutsch/canprod/hw_can4linux_ext.html

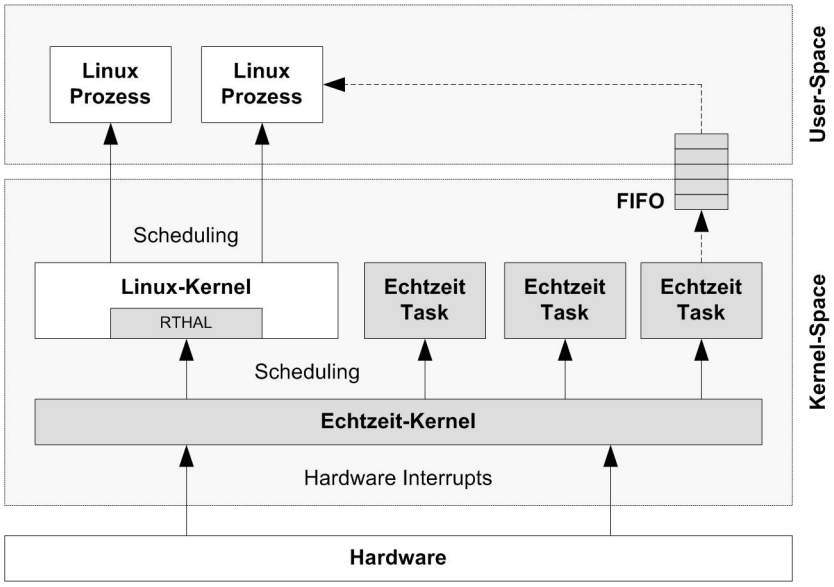


Abbildung 2.4: RTAI ist zwischen Hardware und Linux-Kernel[3]

ihn als Netzwerkprotokoll zu implementieren. Diesen Ansatz wählte das socketcan Projekt[5]. Mit der Kernelversion 2.6.25 erhielt es Einzug in den Linux-Kernel. Linux

2.2.3 Mikrocontroller

Um den Harwareaufbau möglichst einfach zu halten, wurde entschieden, ein SoC System zu verwenden. Im Mikrocontroller sollte möglichst ein CAN- und ein Displaycontroller integriert sein. Es ist zwar möglich, sowohl Displaycontroller als auch CAN-Controller extern an einen Mikrocontroller anzuschließen, dies vergrößert aber zum einen den Platzbedarf auf der Platine zum anderen den Beschaltungsaufwand.

Mikrocontroller mit integriertem CAN-Buskontroller oder Displaycontroller gibt es von verschiedenen Herstellern. Beide zusammen wurden aber nur bei dem AT91SAM9263 von Atmel gefunden. Siehe Tabelle 2.4.

Bei dem AT91SAM9263 von Atmel (2007b) handelt es sich um einen ARM926EJ-S mit bis zu 240MHz. Neben CAN- und Displaycontroller bietet er eine Fülle an Schnittstellen. RAM und Flash Speicher können über zwei externe Speicherschittstellen angebunden werden. Der CAN-Buskontroller vom AT91SAM9263 hat zusätzlich eine hardwareseitige Unterstützung von TTCAN. Durch die TTCAN Hardwareunterstüzung muss sich die Software nicht darum kümmern, dass die CAN-Nachricht zur richtigen Zeit versandt wird. Sie gibt lediglich

[5] http://developer.berlios.de/projects/socketcan

Hersteller	*Prozessorfamiele*	*Displaycontroller*	*CAN*
Atmel	AVR32	ja	nein
	ARM7	nein	ja
	ARM9	ja	ja
NXP	ARM7	nein	ja
	ARM9	nein	ja
ST	ARM7	nein	ja
	STM32	nein	ja
	ARM9	nein	ja
Analog Devices	ARM7	nein	nein
Samsung	ARM7	nein	ja
	ARM 9	ja	nein
	ARM11	ja	nein
Freescale	ARM7	nein	ja

Tabelle 2.4: Mikrocontroller Angebot der Hersteller

vor, in welchem Zeitslot die Nachricht versandt werden soll. Ohne die Hardwarunterstüzung von TTCAN müsste die Software harte Echtzeitanforderungen erfüllen. Es müsste in Software auf Timemaster-Nachricht gewartet werden und dann im richtigen Zeitslot die Nachricht versandt werden.

Für die meisten Schnittstellen des AT91SAM9263 existieren Linux Treiber. Selbst für den CAN-Bus ist eine einfache Socketcan Implenebtierung vorhanden, welche aber noch überarbeitet werden muss.

3 Hardwareaufbau

Kern der Entwicklung ist das TFT-Display (Hitachi TX07D09VM1CBB) und der ARM9 Prozessor (Atmel AT91SAM9263) sowie die Tasten und LEDs des Userinterfaces. Neben diesen beiden Hauptkomponenten werden noch weitere Komponenten für den Betrieb benötigt. Ein Flash-Speicher zum Speichern der Daten, Arbeitsspeicher, ein SD-Karteninterface für die Konfigurationsdaten. Desweiteren wird eine Spannungswandlung für die einzelnen Komponenten benötigt.

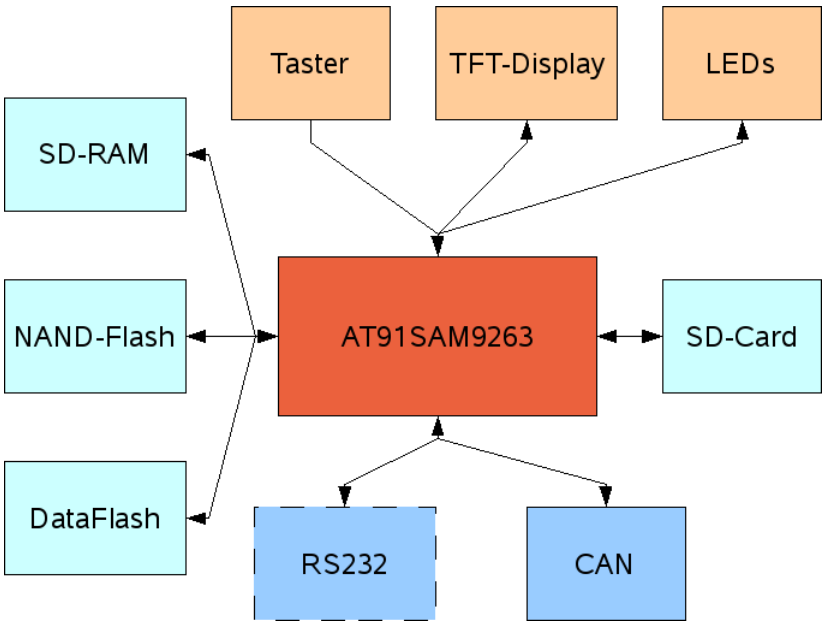


Abbildung 3.1: Hardwareaufbau

Die RS232-Schnittstelle dient lediglich als Debugschnittstelle. Über diese Schnittstelle wird die Konsole zum PC übertragen.

3.1 Komplette Eigenentwicklung

Der Hardwareaufbau orientiert sich an dem Entwicklungsbord AT91SAM9263-EK von Atmel. Von dem Bord stellt Atmel sowohl Schaltplan Atmel (2007a) als auch das Platinenlayout Atmel (2008) zur Verfügung. Um eine möglichst sichere Funktion zu gewährleisten wurden die Modifikationen am Schaltplan so gering wie möglich gehalten. Es wurden nur die nicht benötigten Komponenten entfernt. Alle anderen Bauteile wurden übernommen.

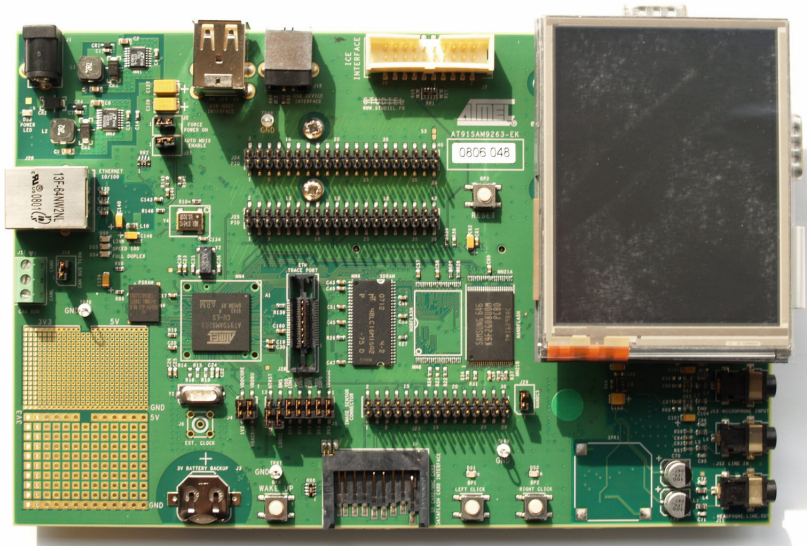


Abbildung 3.2: Entwicklungsbord AT91SAM9263-EK von Atmel

3.1.1 Speicher

Auf dem Entwicklungsbord sind verschiedene Speicher für unterschiedliche Aufgaben vorhanden. Der Speicher ist über zwei **E**xtendet **B**us **I**nterfaces (EBIs) an den Prozessor angebunden.

3.1.1.1 Bootspeicher

Das Booten geschieht beim AT91SAM9263 auf zwei Wegen, entweder aus dem NOR-Flash (muss beim Evaluation-Bord nachbestückt werden) oder über das im ROM befindliche Bootprogramm. Welche Alternative gewählt wird, wird hardwaremäßig über den BMS-Switch festgelegt. Das Bootprogramm versucht den ausführbaren Code auf der SD-Karte, im Nand-Flash oder im DataFlash zu finden. Ist kein ausführbarer Code vorhanden, wird das Debug-

tool SAM-BA gestartet. Es ist über USB/RS232 als Konsole verfügbar. (Atmel, 2007b, Kapitel 13. AT91SAM9263 Boot Program)

Standardmäßig verwendet Atmel auf seinem Entwicklungsbord zum Booten die hauseingene DataFlash-Karte. Hierbei handelt es sich um einen Flash mit SPI-Schnittstelle. Diese ist sowohl als Karte also auch als Chip erhältlich. Die Karte hat die gleiche Bauform wie eine MMC Karte.

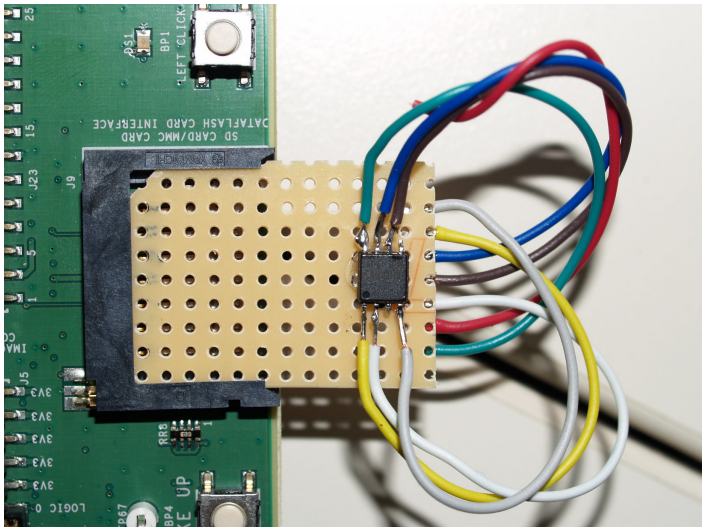


Abbildung 3.3: Eine DataFlash-Karte war kurzfristig nicht Lieferbar. Als Ersatz wurde der DataFlash auf eine Streifenrasterplatine gelötet, welche richtig zugesägt genau in einen SD-Kartenslot passt.

Um Platz zu sparen und weil wir keinen wechselbaren Boot-Speicher benötigen, wurde die Chipvariante im Lenkrad vorgesehen.

3.1.1.2 Datenspeicher

Als Datenspeicher kommt wie auf dem Evaluation-Bord ein 8-Bit NAND-Flash mit 256MB zum Einsatz. Im Datenspeicher soll das Dateisystem für das Betriebsystem liegen. Siehe Kap. 4.4.2.

3.1.1.3 Arbeitsspeicher

Die 64MB Arbeitsspeicher wurden vom Evaluation-Bord übernommen. Der Speicherbedarf der Displaysoftware lässt sich im voraus nicht einschätzen. 64MB ist für ein embedded Sys-

tem großzügig dimensioniert. Sollte der Speicher zu knapp werden muss die Displaysoftware optimiert werden.

3.1.1.4 Framebuffer

Das Evaluation-Bord verfügt über 4MB PSRAM. Diese können als Framebuffer genutzt werden. Da dieser Speicher als einziges am EBI1 hängt ist der Zugriff auf diesen Speicher besonders schnell. Bei kleineren Displays reicht es aus den Framebuffer im Arbeitsspeicher zu halten. Da das Zeitverhalten noch nicht ausgetestet werden konnte wurde der Speicher beibehalten.

3.1.2 CAN-Bus

Neben dem im AT91SAM9263 integrierten CAN-Controller ist noch ein CAN-Transiver nötig. Der CAN-Transiver ist für die Wandlung des Differenzsignals des CAN-Buses in einen TTL-Pegel und zurück zuständig. Der auf dem Evaluation-Bord verwendete SN65HVD234 von Texas Instruments wurde beibehalten. Dieser Transiver verfügt über eine Enable/Disable-Funktion sowie über einen Stromsparmodus. Texas Instruments (2008) Diese Funktionen werden per GPIO gesteuert.

3.1.3 Display

Als Display kommt wie erwähnt (siehe 2.1.4.2 Anforderungen für das TFT-Display) ein 2,7” TFT-Display (TX07D09VM1CBB Hitachi (2006)) zum Einsatz. Dieses Display wurde Aufgrund seiner Helligkeit aus dem Sortiment des Sponsor DATA MODUL ausgesucht. Auf dem Entwicklungsbord von Atmel kommt ein 3,5” TFT-Display (TX09D70VM1CCA Hitachi (2005)) zu Einsatz. Diese Display unterscheidet sich von dem bei uns eingesetzten Display in den in Tabelle 3.1 aufgeführten Punkten.

Für die Schaltung sind vor allem die Unterschiede in der Schnittstelle interesant. Der Hauptunterschied besteht in der Stromversorgung. Das TX09D70VM1CCA besitzt auf der Rückseite eine Platine die aus den 3,3V Eingangspannung die entsprechenden Spannungen für das Display erzeugt. Ohne diese Platine wäre das Display nur 2mm dicker. Ein Display zu genaueren Untersuchung der Platine stand zum Zeitpunkt der Schaltplanentwicklung nicht zur Verfügung.

	TX07D09VM1CBB	TX09D70VM1CCA
Abmessungen		
Außenmaße	50,54x68,62x2,6mm	64,0x86,0x8,05mm
Bildfläche	41,04x54,72mm	53,64x71,52mm
Bilddaten		
Auflösung	240x320 Pixel	
Farben	262.000	
Bildwiederholfrequenz	60Hz	
Hindergrundsbeleuchtung	LED	
Leuchtstärke	420 cd	320 cd
Kontrast	300:1	
Schnittstelle		
Schnittstelle	6bit pro Farbe TTL	
Stromversorgung	3,3V; 5V; 15V; -15V	3,3V
Touchscreen	nein	ja

Tabelle 3.1: Unterschiede zwischen dem Display des Entwicklungsbord von Atmel und dem im Lenkrad eingesetzten

3.1.4 Taster

Die Taster werden an GPIO-Pins des AT91SAM9263 angeschlossen. Damit immer ein definierter Pegel an den Pins anliegt ist ein Pullup-Widerstand nötig. Im AT91SAM9263 ist ein Pullup-Widerstand ($\sim 100k\Omega$ (Atmel, 2007b, Kap. 46. Electrical Characteristics)) integriert und kann Softwareseitig eingeschaltet werden. Beim Drücken des Taster wird der Pegel auf Masse gezogen. Wenn der Taster zu stark prellt, kann das Signal mit Hilfe eines Kondensators geglättet werden. Dieser Kondensator wird bei Bedarf direkt am Taster integriert.

3.1.5 LEDs

Die GPIO-Pins des AT91SAM9263 dürfen nur mit einem geringen Strom (8mA (Atmel, 2007b, Kap. 46. Electrical Characteristics) belastet werden. Die LEDs benötigen deutlich mehr Strom (20mA). Deswegen werden sie über einen MOSFET angesteuert. Das Strom/-Spannungsverhältnis wird über einen Vorwiderstand sichergestellt. Dieser ist für die Farben Rot und Blau unterschiedlich.

	Rot	*Blau*
Durchmesser	3mm	
Helligkeit	3cd	
Spannung	1,8 - 2,0V	3,2 - 3,4V
Strom	20mA (50mA max.)	

Tabelle 3.2: Technische Daten LEDs

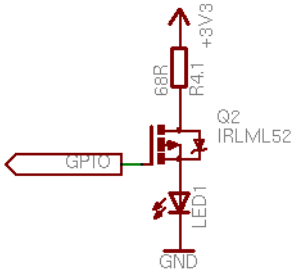


Abbildung 3.4: Ansteuerung der LEDs (P-Kanal MOSFET)

3.1.6 Stromversorgung

Die Stromversorgung für die Spannungen 1,2V, 3,3V, 5V wurde von dem Entwicklungsbord von Atmel übernommen. Für die +15V und -15V wurden die Step-Up / -Down Wandler MAX762 (Maxim (1993)) und MAX766 (Maxim (1994)) von der Firma Maxim gewählt. Für die Spannungswandlung kommen so genannte Schaltregler zum Einsatz. Diese haben eine besonders hohe Effektivität. Diese Wandler setzten hohe Frequenzen zum Wandel der Spannung ein. Um ein saubere Ausgangsspannung zu bekommen sind deshalb Spulen, Kondensatoren sowie eine Diode nötig (siehe Horowitz und Hill (1996)). Die genaue Beschaltung und die Werte der Bauteile wurden dem Referenzschaltbild aus dem Datenblättern der Bauteile übernommen.

In Tabelle 3.3 ist der Strombedarf der einzelnen Komponenten aufgelistet. Dieser liegt deutlich unter den von den Spannungsregler zur Verfügung gestellten Werten siehe Tabelle 3.4.

Für den Standbybetrieb des Lenkrades, wenn es vom Bordnetz getrennt ist (Siehe 2.1.3 Hardwareanforderungen Stromversorgung) sind folgende Dinge notwendig:

- Alle Stromverbraucher die nicht unbedingt benötigt werden müssen abschaltbar sein.
- Absenken des Systemtaktes
- Der Strom muss zwischengespeichert werden. Für den Fahrerwechsel im Endurance wäre eine Zeit von min 3min sinnvoll.

Bauteil	*Spannung*	*max. Strom*
AT91SAM9263	1,2V	73,3mA
	3,3V	250mA
Arbeitsspeicher	3,3V	285mA
NAND-Flash	3,3V	30mA
PSRAM	3,3V	30mA
DataFlash	3,3V	17mA
SD-Karte	3,3V	
	3,3V	0,6mA
Display	5V	2,5mA
	15V	0,06mA
	-15V	0,22mA

Tabelle 3.3: Spannungen und Ströme die von den einzelnen Bauteilen benötigt werden

Spannung	*max. Strom*
15V	150mA
5V	4A
3,3V	4A
1,2V	250mA
-15V	105mA

Tabelle 3.4: Maximale Ausgangsströme die die Spannungswander liefern

Zur Speicherung des Stroms kommen folgende Ansätze in Frage

Ansatz	*Vorteile*	*Nachteile*
Speicherung mit Akku	lange Laufzeit	benötigt Ladeelektronik
		kurze Lebenzeit des Akkus
Speicherkondensatoren	einfach in der Anwendung	kurze Laufzeit

Tabelle 3.5: Lösungsansätze Strompufferung

Bei den eingesetzten Spannungswandlern ist keine sinnvolle Abschaltung möglich. Die Spannungswandler MAX762 und MAX766 besitzen zwar einen Shutdown Pin, dieser hat aber zur Folge das die Eingangsspannung durchgeschleift wird. Eine Lösungsmöglichkeit wäre der Einsatz des LTC3450 von Linear Technology. Dieser für kleine TFT-Displays optimierte Spannungswandler stellt die Spannungen 5V, 15V und -15V bereit. Sie lassen sich über einen Shutdown-Pin zusammen abschalten.

Auf eine Pufferung des Stromes wurde in der ersten Version vor allem aus Platzgründen verzichtet.

3.1.7 Platine

Für die Herstellung der Platine müssen zu erst die Bauteile plaziert werden. Anschließend müssen die Leiterbahnen verlegt werden.

3.1.7.1 Platzverteilung

Vom Interieur wurden für die Platine die Maße 50x110mm vorgegeben. Damit ist die Platine kaum größer als das Display. Die Breite resultiert aus der Entscheidung das Display über das Displayloch im Lenkrad zu montieren.

Alle Komponenten müssen also auf engstem Raum untergebracht werden. Der Abstand zwischen Platine und Display beträgt max. 5mm, sodass die Oberseite der Platine nur eingeschränkt bestückbar ist. Alle Bauteile mit größeren Bauhöhen müssen auf der Unterseite untergebracht werden. Auch von der Wärmeabfuhr ist es günstiger die Bauteile auf der Unterseite zu bestücken. Auf der Oberseite liegt das Display fast auf dem Bauteil auf, so dass kaum eine Möglichkeit besteht die zwar meist geringe Wärme abzuführen.

Der Bereich unterhalb des Displays ist für die Taster vorgesehen. Die Bauform der Taster stand zum Zeitpunkt des Routens noch nicht fest, so dass der Bereich komplett freigehalten werden musste. In Abb. 3.5 ist hier nur die Batterie für die Erhaltung der Zeit in der RTC platziert. Bei Platzproblemen mit den Tastern kann die Batterie weggelassen werden, eine gültige Zeit für die RTC ist für die Funktion des Displays nicht nötig.

Fast die Hälfte des Platzes unter dem Display nimmt die Stromversorgung in Anspruch. Hier fallen vor allem die großen Pufferkondensatoren ins Gewicht. Beim Platzieren der Bauteile für die Spannungversorung ist zu beachten, dass sie möglichst nahe am IC liegen. Bei den meisten Bauteilen gibt es im Datenblatt einen Hinweis wie die Bauteile anzuordnen sind. Des weiteren bekommt jede Spannungsquelle ihre eigene Massefläche um ein Übertragen der Schwingungen aus den Schaltreglern (siehe Kap. 3.1.6) zu vermeiden.

3.1.7.2 Technische Randbedingungen

Beim Routen muss man zum einen beachten, was technisch umsetzbar ist. Je kleiner die Leiterbahnabstände, die Größen der Bohrlöcher usw. sind, um so teurer ist die Fertigung. Zum anderen muss man auf die elektrischen Anforderungen achten.

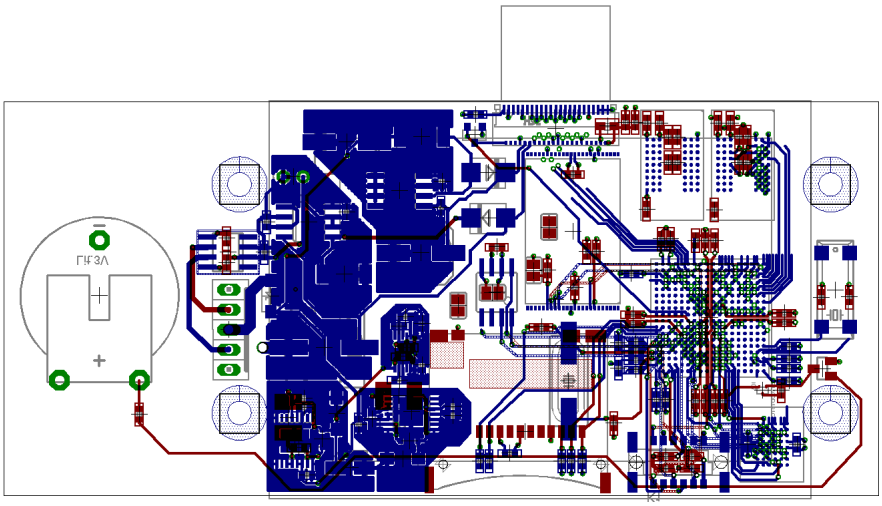


Abbildung 3.5: Teilweise geroutete Platine, noch ohne Anschlüsse für Taster und LEDs (rote Leiterbahnen sind auf der Oberseite, die blauen auf der Unterseite)

Hier ist vorallem auf die Leiterbahnlänge und der Abstand zu benachbarten Leiterbahnen wichtig. Die maximale Leiterbahnlänge und der maximale Leiterbahnlängenunterschied hängt von der Frequenz des zu übertragenden Signals ab. Gemäß Jones (2004) breitet sich ein Signal um $6Zoll \approx 1,5cm$ pro Nanosekunde aus. Hieraus ergibt sich, dass für die bei der Speicheranbindung verwendeten 100MHz Signale der maximale Abstand 7cm beträgt. Zu beachten ist auch, dass die Leiterbahnen zwischen zwei Komponenten die gleiche Länge haben, ansonsten kann es passieren, dass das Clocksignal beim Chip ankommt, bevor alle Daten anliegen.

Bei mehrlagigen Platinen wird die Platine aus mehren Doppelseitigen Platinen zusammengeklebt. Die Platinenkerne sind in Abb. 3.6 grün dargestellt. Diese Kerne werden mit Prepreg (grau) zusammengeklebt. Die einzelnen Kerne lassen sich einzeln mit Durchkontaktierungen versehen. Auch eine Gruppe zusammengeklebter Kerne lässt sich mit Durchkontaktierungen versehen. Zusätzlich gibt es noch so genannte Blind-Vias. Diese werden von außen bis zu einer Bestimmten Länge in die Platine gebohrt. Die maximale Bohrtiefe hängt vom Bohrerdurchmesser ab. Diese Sacklöcher haben den Vorteil, dass sie einen besonders kleinen Bohrdurchmesser haben.

Die Fertigung von mehrlagigen Platinen sollte man vorher mit der Fertigungsfirma absprechen. Vorallem wenn man Plant Burried- (versteckte) und Blind-Vias einzusetzten möchte.

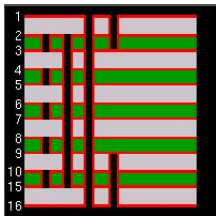


Abbildung 3.6: Leiterplattenquerschnit mit 12 Layern. Neben den Durchkontaktierungen durch alle Layer ist es auch möglich nur einzelne Layer zu verbinden. Die Roten Linien sind die Leiterbahnen, grün und grau sind die Isolierschichten zwischen den Layern

3.1.7.3 Raster

In Jones (2004) wird empfohlen ein thou (1/1000 Zoll) basiertes Raster zu verwenden. Dieses Raster sollte so grob wie möglich und so fein wie nötig sein. Mittlerweile sind viele Bauteile in metrischen Maßen. Zum Routen dieser Bauteile ist ein metrisches Raster emfehlenswert. Bei BGA's ist es sinnvoll, den halben Pinabstand als Raster zu nehmen. So kommt man immer mittig zwischen den Pads durch. Leider haben die BGA's kein einheitliches Raster, so dass man häufig das Raster umschalten muss. Dies macht man am Besten in einiger Entfernung von Bauteil wo man ein bisschen Platz hat. Direkt am Bauteil ist der Versatz, der beim Rasterwechsel entsteht sehr störend, da dort meist nicht viel Platz zwischen den Leiterbahnen ist.

3.1.7.4 Routing

Für das richtige Routing gibt es kein Rezept. Leiterplattendesigner betrachten ihre Arbeit als Kunst Jones (2004). Der Autorouter hilft nur bei einfachen Platinen weiter. Sobald man besondere Vorgaben umsetzen möchte muss man dies von Hand tun.

Manchmal ist es hilfreich, sich anzusehen wie das Problem auf andere Platinen gelöst wurde. Vom Atmel (2008) wurde die Lösung des Herausführens der Leiterbahnen unter dem BGA übernommen. Die Vias werden sternförmig auf der Außenseite des Pads platziert. Die äußeren beiden Rehen können ohne Durchkontaktierung herausgeführt werden. Siehe Abb. 3.7

3.1.8 Halterung im Lenkrad

Im Lenkrad soll die Platine das Display gegen das mit Dichtungsband beklebte Alublech drücken. Dies soll verhindern, das Wasser zwischen Alublech und Display in das Innere

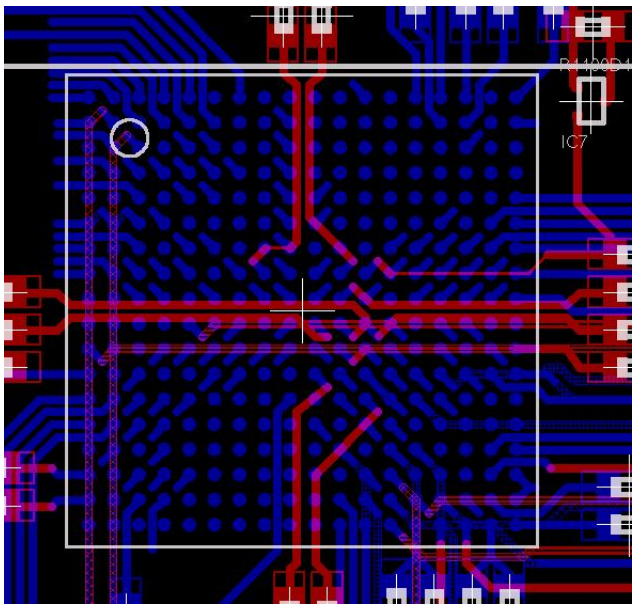


Abbildung 3.7: Die Leiterbahnen werden sternförmig unter dem BGA herrausgeführt. Vias und Luftlienien sind zur besseren Übersicht ausgeblendet.

des Lenkrades gelangt. Da es nicht möglich ist, die Platine von hinten zu verschrauben, es müssten hierfür extra Löcher gebohrt werden, wurden viereckige Bolzen (siehe Abb. 3.8) entworfen. Diese Bolzen werden von den viereckigen Löchern in der Platine gehalten und können sich beim Festziehen der Schrauben von vorne nicht mitdrehen. Die viereckige Form wurde gewählt, da bei ihr die Kraft besser auf die Fläche verteilt wird als beim Sechseck. Bei Sechseck besteht die Gefahr das sich die Ecken in der Platine rundschleifen und sich der Bolzen mitdreht.

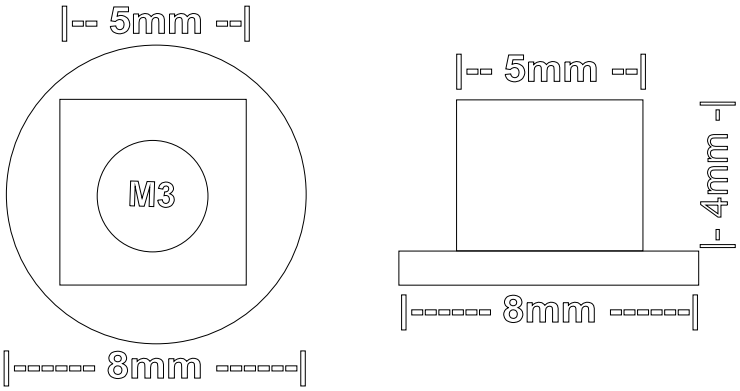


Abbildung 3.8: Bolzen zum Halten der Platine

Das Display wird durch die Bolzen oberhalb und unterhalt des Displays gehalten. Siehe Abb.

3.5. Als Abstandshalter zwischen Platine und Display kommt ein Plastikrahmen zum Einsatz.

3.2 Mit System on Modul

Beim Routen der Platine zeichnete sich ab, dass diese Arbeit wohl mehr Zeit beanspruchen würde als erwartet. Zu diesem Zeitpunkt brachte die Firma Congent das System on Modul CSB737 auf den Markt welches den AT91SAM9263 einsetzt. Als System on Modul bezeichnet man ein Mikroprozessor System die als Modul in das eigene System integriert werden können. Beim CSB737 handelt es sich um die Bauform eine 200-Pin So-Dimm (2,5V) Arbeitsspeicherriegels.

Abbildung 3.9: SoM-Modul CSB737

Auf dem CSB737 Modul befinden sich neben dem Mikrocontroler AT91SAM9263 folgenden Komponenten:

- 64MB NOR-Flash
- 512MB NAND-Flash
- 64MB SDRAM
- 8MB PSRAM
- Ethernet Transiver LAN8700
- Externe Real Time Clock DS1339
- Spannungswandler mit einem Eingansspannungsberiech von 6-35V mit externen 3,3V belastbar mit 2A

3.2.1 Unterschiede zur Eigenentwicklung

Das CSB737 ist mit den 68mm x 50.8mm x 8mm kaum größer als das Display. Durch die benötigte Halterung muss die Trägerplatine eine Mindestbreite von 65mm haben, so dass ein Einbau durch den Displayschacht nicht mehr möglich ist. Es ergeben sich daraus, die in Tabelle 3.6 aufgeführten Änderungen im Design.

	Eigenentwicklung	*SoM*
Platinenmaße	50x110mm	75,5x116,5mm
Displayanschluss	18bit	16bit
Bootspeicher	DataFlash	NOR-Flash
Bootloader	U-Boot	Micromonitor

Tabelle 3.6: Unterschiede Komplette eingenentwicklung / Platine mit SoM

Beim Bootloader handelt es sich um den Bootloader, der schon an das System angepasst ist. Mit dem entsprechenden Portierungsaufwand ließen sie sich durch einen anderen Austauschen.

3.2.1.1 CAN-Bus

Der CAN-Transiver wird beibehalten. Um GPIOs zu Sparen wird der CAN-Transiver hardwaremäßig enabled.

3.2.1.2 GPIO

Beim CSB737 sind die Funktionen der einzelnen Pins genau definiert. Es sind 8 universell einsetzbare Pins vorgesehen. Es können aber auch viele andere Pins als GPIO eingesetzt werden. Man muss aber darauf achten, dass diese direkt nach draußen geführt werden und auch auf dem Entwicklungsbord CSB703 keine Hardware angeschlossen ist. Ob die Pins direkt nach draußen geführt werden, kann nur anhand ihrer Funktion erraten werden, da der Schaltplan des CSB737 nicht frei zugänglich ist.

3.2.1.3 Display

Beim CSB737 ist die Hintergrundbeleuchtung nicht an dem von Atmel vorgesehen LCD-Backlight-Pin angeschlossen, sondern es wird PWM3 dafür verwendet. Der LCD-Backlight-Pin ist aber beim CSB737 herausgeführt. Er ist für das Synchronous Serial Interface (SSI)

vorgesehen. Da wir diese Funktionalität nicht benötigen und wir den PWM3 für die LEDs verwenden wollen, wurde auf der SWCU auch die Verwendung des LCD-Backlight-Pin vorgesehen.

3.2.1.4 PWM

Auf dem CSB737 sind nur 2 Pinns als PWM vorgesehen. Einen zusätzlichen PWM haben wir durch das Verwenden des extra LCD-Backlight-PWMs gewonnen. Auf dem SSI-Controller liegen noch zwei weitere PWMs, die wir für die LEDs nutzen können. Für die genau Pin/-Funktionsbelegung siehe Angang B.

3.2.1.5 Debugschnittstelle

Da der Ein- und Ausbau aus dem Lenkrad nicht ganz einfach ist, wurde eine zusätzliche Debugschnittstelle hinzugefügt. Diese sollte alle zum Aufspüren von Fehlern notigen Schnittstellen zusammen fassen. Die in Tabelle 3.7 aufgelisteten Schnittstellen sollten nach Möglichkeit integriert werden.

Schnittstelle	*Anzahl der benötigten Pins*
JTAG	5
RS232	2 (3)
Ethernet	4
USB Device	3 (4)
Ext. Stromversorgung	2

Tabelle 3.7: Schnittstellen der Debugbuchse und die für sie benötigte Pinanzahl (in Klammer mit Masse)

Die Buchse sollte am oberen Rand der Platine sitzen, da mit sie von oben durch das Loch erreich bar ist. An dieser Stelle ist nicht viel Platz. Zwischen SoM-Modul und Platinenrand sind max. 1cm Platz. Diese stehen aber nur in der Mitte der Platine zur Verfügung. Wir benötigten also eine Buchse mit mindestens 16 Pins, (die Massepins können zusammengefasst werden) die an den oberen Platinenrad passt. Wir wählten die HDMI-Buchse aus. Sie verfügt über 19 Pins und benötigt nur 7x15mm an Platz. Des weiteren ist nicht zu erwarten, dass jemand zufällig ein HDMI-Kabel+Gerät in der Tasche hat und auf die Idee kommt es am Display anzuschließen. Diese würde zu Beschädigungen auf beiden Seiten führen.

Für die Ethernet-Schnittstelle werde externe Spulen zur galvanischen Trennung benötigt. Diese sind meist in einer RJ45-Buchse integriert.

Funktion	*Pin*	*Funktion*	*Pin*
USB		**RS232**	
USB VCC	1	TXD	10
D+	2	RXD	11
D-	3	**Stromversorgung**	
GND		GND	12
JTAG		+12V	13
TRST	5	**Ethernet**	
TDI	6	TD+	14
TMS	7	TD-	15
TCK	8	RD+	16
RST	9	RD-	17
		GND	18

Tabelle 3.8: Pinbelegung der Debugbuchse

3.2.2 Trägerplatine

Trotz der daraus resultierenden Vergrößerung der Bauform, entschieden wir uns für den Einsatz des CSB737, da es das Gelingen des Projektes wahrscheinlicher machte. Bei Platinen dieser Größenordnung und unserer Erfahrung mit dem Routen von Platinen muss mit Fehlern in der ersten Platinenversion gerechnet werden. Da wir uns zeitlich die Produktion einer Zweitversion nicht mehr leisten konnten, wurde besonders viel Zeit in die Abschlusskontroller der Platine investiert. Die Fertigungszeit beträgt ca. 10 Arbeitstage. Ein Eilservice würde die Fertigungskosten vervierfachen, was für uns in keinem Verhältnis zum Zeitgewinn steht.

3.2.3 Montage im Lenkrad

Wie bereits oben erwähnt, kann die Platine durch die größeren Abmessungen nun nicht wie ursprünglich geplant durch das Loch für das Display ins Lenkrad eingeführt werden. Stattdessen wurde ein Extraloch an der Oberseite des Lenkradmittelteils vorgesehen. Bei der abschließenden Überprüfung der Maße mit der Baugruppe Interieur stellte sich heraus, dass Positionen der Löcher für die Befestigungsbolzen verändert werden mussten. Um die Befestigungsbolzen seitlich des Displays platzieren zu können, musste die Platine auf ihr maximal mögliche Breite vergrößert werden.

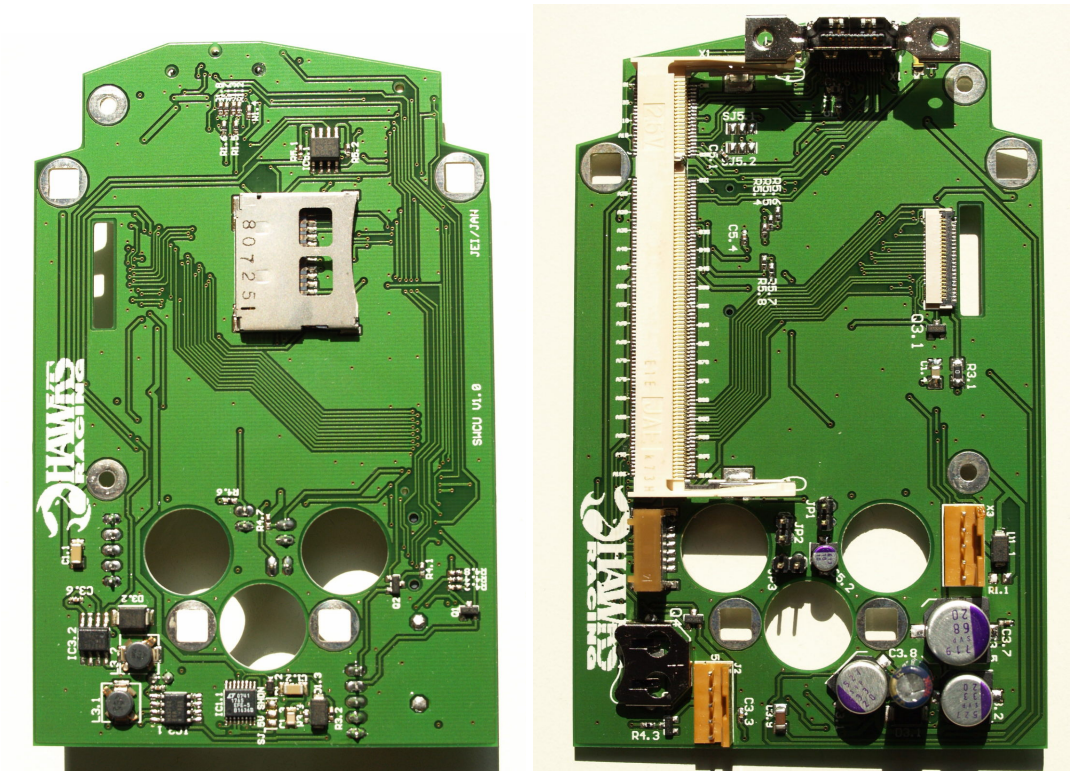


Abbildung 3.10: SWCU-CSB737

3.2.4 Probleme und deren Behebung

In diesem Kapitel werden die Probleme behandelt, die in der ersten Version der SWCU-CSB737-Platine festgestellt wurden. Des weiteren werden Möglichkeiten zum Beheben dieser Probleme aufgezeigt. Um Ideen für die Lösung der Probleme zu bekommen, wurde die Platine auf der Rückseite des Displays TX09D70VM1CCA vom Enwicklungsbord AT91SAM9263-EK (siehe Kap. 3.1.3) genauer betrachtet. Auf den ICs der Platine ist Aufgrund der kleinen Bauform nur eine verkürzte Bezeichnung aufgedruckt. Diese Bezeichnung ist nicht eindeutig, was die Identifiezierung erschwert.

Hintergrundbeleuchtung Durch ein Missverständniss des Datenblattes wurde die Hintergrundbeleuchtung nur mit einer Spannung von 3,3V versorgt. Das Datenblatt meinte aber eine Spannung von 3,2V pro LED. Dies ging aber nicht deutlich aus dem Datenblatt hervor. Lediglich ein vermerk am Rand wies darauf hin. Die Information das 5 LEDs in der Hintergrundbeleuchtung verbaut sind geht nur aus dem Blockdiagramm weiter hinten im Datenblatt hervor. Die Spannung müsste $5 * 3,2V = 16V$ betragen. Zum kurzfristigen Beheben dieses Problems, wurde das Display mit Hilfe von Widerständen an +15V und -15V angeschlossen.

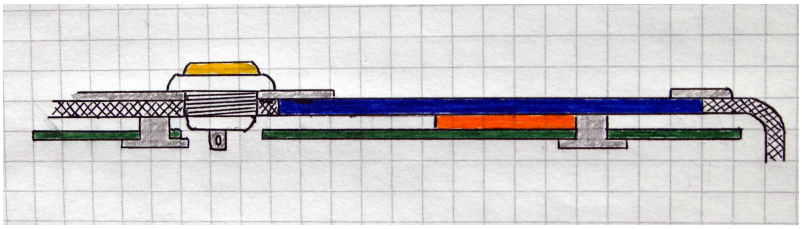


Abbildung 3.11: Skitze der Platine im Lenkrad von der Seite. (Grün: Platine, Blau: Display, Gelb: Taster, Grau: Halterungsbolzen und Alublende, Schwarz schrafiert Carbon)

Es stand also eine Spannung von 30V zur Verfügung. Gemäß dem omischen Gesetzt benötigen wir als Vorwiederstand $\frac{30V-16V}{20mA} = 700\Omega$. Der nächstgelegene Normwert ist 680Ω.

Der in Abb. 3.13 unten links zu erkennende IC8 ist ein LED Treiber (LT3465) von Linear Technology (b). Dieser sorgt dafür, dass aus den 3,3V Eingangsspannung die richtige Ausgangsspannung für die in Reihe geschalteten LED's der Hintergrundbeleuchtung zur Verfügung steht. Dieser LED-Treiber bietet sich auch für die nächste Platinenversion an. Zur Helligkeitsregelung lässt er sich direkt mit dem PWM-Signal ansteuern.

5V Spannungsversorgung Bei der 5V Spannungsversorgung wurde die Spule vergessen, die zwischen dem Spannungsausgang (SW) und der Feedbackleitung (FB) liegt. Somit war nur die Feedbackleitung an das +5V Netz angeschlossen. Das Problem wurde durch das Nachbestücken einer bedrahteten Spule gelöst.

Flimmerdes Display Nach der Korrektur der Hintergrundbeleuchtung stellte sich heraus, das dass Display flimmert. Die erste Vermutung war, dass es an einer Schwingung in der Stromversorgung der Hindergrundbeleuchtung liegt. Schließlich gelten TFT-Displays als flimmerfrei. Bei einer Überprüfung der Spannungen +15V und -15V mit dem Oszilloskop konnte keine Schwingung auf den Leitungen festgestellt werden.

Um das Problem zu verstehen, muss man sich etwas genauer mit der Funktion von TFT-Displays beschäftigen. Eine Einführung findet man in Rink u. a. (1998). Damit die Flüssigkristalle nicht beschädigt werden, wird bei jedem Schaltvorgang die Polarität der Videosignalspannung (VDH) geändert. Dies geschieht mit Hilfe der Backplane-Referenzspannung (VCOM) Nachbaur (2008). Wir haben die im Datenblatt Hitachi (2006) als typisch angegebenen 2,2V mit Hilfe eines Spannungsteilers aus Festwiederständen eingestellt. Das im Datenblatt im Blockdiagramm eingezeichnete Potentiometer hielten wir für nicht nötig. Trifft man nicht die richtige Referenzspannung kommt es zum Flimmern des Displays. Als Verschärfung kommt noch hinzu das beim Spannungsteiler mit festen Normwiderständen nur

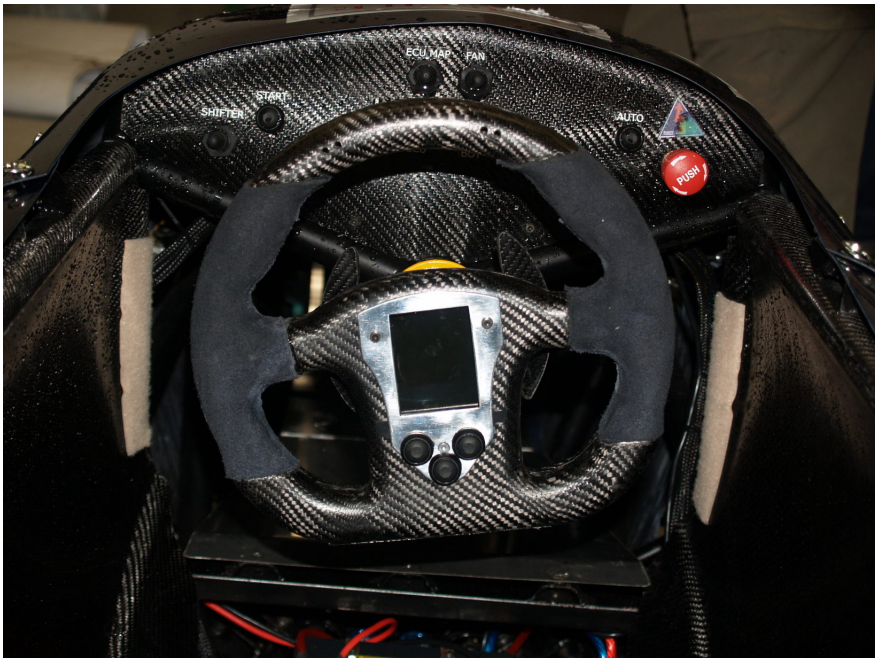


Abbildung 3.12: Display im Lenkrad des Hawk08 montiert

eine Annäherung erzielt werden kann. Der Fehler bei unserem Spannungsteiler liegt bei $|\frac{5V*4,3k\Omega}{5,1k\Omega*4,3k\Omega} - 2,2V| = 0,09V$ zzgl. Winderstandstoleranz. Das Potentiometer ist also zum nachregulieren der Referenzspannung unbedingt nötig. Alternativ bieten viele Chiphersteller spezielle Spannungswandler für VCOM an, bei denen man die Spannung z.B. via I2C einstellen kann.

Beim TX09D70VM1CCA wird ein Potentiometer verwendet wie in Abb. 3.13 oben in der Mitte zu erkennen ist.

Befestigung der Platine im Lenkrad Die Montage im Lenkrad ist nicht optimal gelöst. Dies lag vor allem daran, dass von der Baugruppe Interieur nicht alle Teile rechtzeitig gefertigt werden konnten. Eines der größten Probleme war die wackelige Befestigung des Displays. Dies wurde nur durch das Flachbandkabel mit der Platine verbunden. Ein weiteres Problem ist, dass bei dem Ausmessen der Innenmaße des Lenkrad im CAD die Rundung der seitlichen Kante nicht berücksichtigt wurde. Um die Platine dennoch im Lenkrad montieren zu können musste der Abstand zwischen Platine und Display vergrößert werden. Diese Konstruktion führte zu einer instabielen Verbindung zwischen Diplay und Platine und machte die Montage im Lenkrad zum Geduldsspiel. Diese sehr wackelige Verbindung führte letztendlich zum Totalausfall der Hardware.

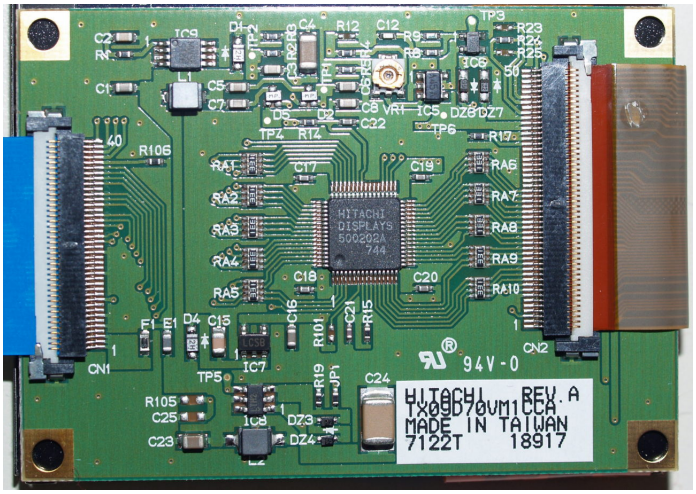


Abbildung 3.13: Platine auf der Rückseite des TX09D70VM1CCA

4 Software

Im zweiten Teil dieser Arbeit geht es um die Software, die auf der Hardware laufen soll. Dabei liegt der Schwerpunkt auf dem Linux-Kernel. Die anderen zum Betrieb des Systems benötigten Komponenten wie Bootloader, Crosscompieler und der für den Betrieb benötigten Userspace-Software wird nur am Rande behandelt.

4.1 Build envirement

Bei embedded Systemen ist die Hardware nicht so einheitlich wie bei PCs. Aus diesem Grunde können Binerdistributionen nur für bestimmte Geräte angeboten werden. Um das Übersetzen der Software für die embedded Geräte zu vereinfachen wurden verschiedene Scriptsammlungen entwickelt, die den Vorgang automatisieren. Diese Scriptsammlungen nennt man Build envirement. Zu den Aufgaben der Scripte gehört es, als erstes den Crosscompieler für das System zu übersetzen. Anschließend kann mit ihm die Software für das System übersetzt werden. Im besten Fall wird vom Build envirement ein fertiges Image für das Zielgerät produziert, welches nur noch in den Chip geflasht werden muss.

4.1.1 Buildroot

Buildroot[1] ist ein Projekt welches aus dem μLinux Projekt hervorgegangen ist. Ziel von μLinux ist es Linux auf Prozessoren ohne MMU laufen zu lassen. Da diese Prozessoren auch meist wenig Speicher besitzen setzt Buildroot auf Platzeffizienz. Statt der umfangreichen Glib verwendet es die optimierte μClib.

Zur Konfiguration steht eine ncurses (Bibliothek für grafische Oberflächen auf der Konsole) Oberfläche (siehe Abb. 4.1), wie sie auch von der Linux-Kernel Konfiguration bekannt ist, zur Verfügung. In dieser Oberfläche lassen sich die verwendete Hardware (sofern unterstützt) und die benötigten Softwarepakete auswählen. Durch Aufrufen des Programmes „make“ startet man den Buidvorgang. Am Ende erhält man fertige Images für Bootloader, Linux-Kernle und Filesystem.

[1] http://buildroot.uclibc.org/

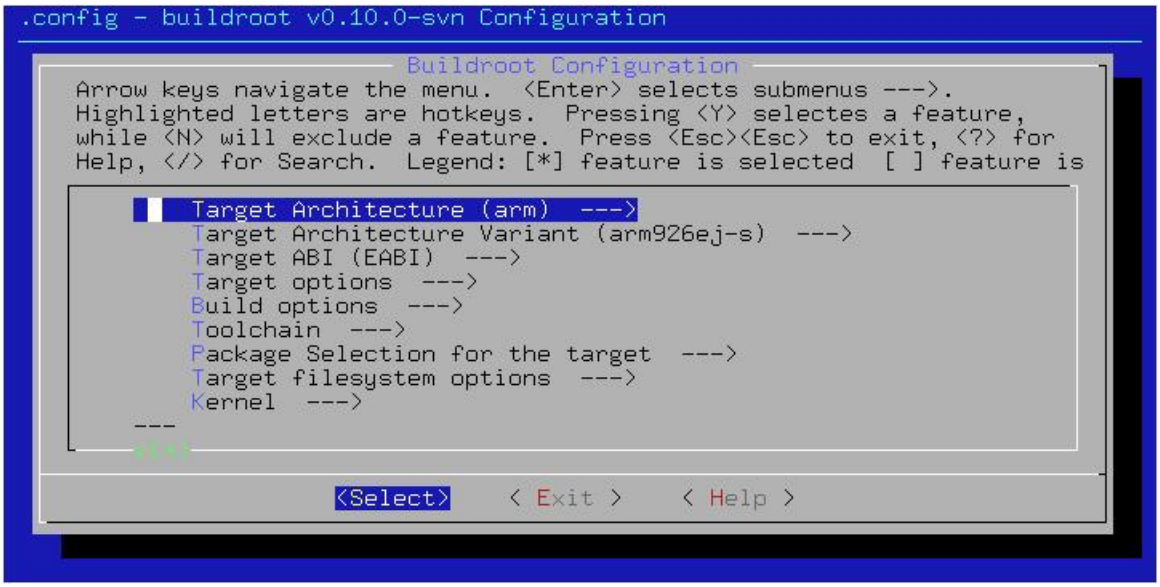


Abbildung 4.1: Konfiguration von Buildroot

Leider erzeugt Buidroot zur Zeit (SVN Rev. 22334) keinen gültigen Byte-Code für den AT91SAM9. Somit lassen sich die mit buildroot erstellen Programme nicht ausführen. Einzig der Linux-Kernel wird korrekt gebaut. Aus diesem Grund mussten wir auf OpenEmbedded wechseln. Es war keine Zeit den Grund dieses Problems genauer zu untersuchen.

4.1.2 OpenEmbedded

OpenEmbedded[2] bietet eine große Auswahl an Software. Es ist die Grundlage vieler embedded Distributionen z.B OpenMoko ein opensurce Handy. Die Konfiguration ist ziemlich Komplex. Sie findet über Variablen in Konfigurationsdateien statt welche nicht vollständig dokumentiert sind. OpenEnbedded wurde nur zum Bauen einzelner Paket verwendet eine Distribution mit automatischer Dateisystemerstellung wurde aus Zeitgründen nicht konfiguriert.

Zum Bauen eines Paketes ruft man das Programm „bitbank“, welches zu OpenEnbedded gehört, mit dem Paketnamen als Parameter auf. Bitbank kümmert sich dann um das Auflösen der Abhängigkeiten und baut alle nötigen Pakete.

Für eine minimale Distribution wurden folgende Pakete übersetzt:

- Busybox

[2] http://www.openembedded.org

- qtopia-core
- mtd-utils
- sysvinit

So wie alle Pakete die von obigen abhängig sind.

Beim Übersetzen von qtopia musste beachtet werden, dass der Mauszeiger deaktiviert ist. Unser System verfügt über keine Maussteuerung, sodas der Mauszeiger auf dem Bildschirm störend ist.

4.2 Bootvorgang

Beim Booten eines embedded Systems wird zu erst der Bootloader geladen. Dieser Bootloader entspricht dem BIOS eines PCs. Er führt die grundlegenden Konfigurationen des Systems durch.

Anschließend lädt der Bootloader den Betriebssystem-Kernel. Die aufgabe des Kernels ist es die Treiber für die Hardware zu laden und diese zu initialisieren. Nach Beendigung dieser Arbeit lädt er ein Init-Programm. Dieses Programm befindet sich bei Linux standardmäßig in /sbin/init kann aber mit Hilfe der Kernel-Argumenten geändert werden.

Das Init Programm ist das erste Userspace-Programm welche gestartet wird. Es hat die Aufgabe alle weiteren Programme zu starten. Hierzu gehören Dienste und die Shell.

Software	*Zeit*
U-Boot	3s
Micromonitor	3s
Linux	5s
sysv-init	10s
DisplayMenu	15s

Tabelle 4.1: Zeit die die einzelnen Softwareteile beim Booten benötigten

Für die Zukunft ist eine weitere Optimierung der Bootzeit wünschenswert. Vor allem die lange Startzeit des sysv-init fällt auf. Diese lässt sich durch den Einsatz durch eine Variante für embedded Systeme verkürzen. Siehe Rebe (2008a).

4.3 Bootloader

4.3.1 AT91-Bootstrap

AT91-Bootstrap stammt vom Atmel und führt die grundlegende Konfiguration durch.

1. Setzen des Systemtaktes
2. Konfiguration des Arbeitsspeichers
3. Konfiguration der Debug-Schnittstelle
4. Konfiguration der Flashspeicher
5. Laden des Programms aus dem Flash in den Arbeitsspeicher
6. Ausführen des Programms

Leider kann AT91-Bootstrap noch nicht direkt Linux starten. Deswegen lädt es U-Boot.

4.3.2 U-Boot

U-Boot steht für Universal Bootloader. U-Boot ist beim AT91SAM9 zur Zeit (git[3] 2008-07-31) ein 2nd Stage Bootloader. Es ist zwingend erforderlich das AT91-Bootstrap den Mikrocontroller schon initialiesiert hat.

U-Boot verfügt über eine Netwerkanbindung, so dass man Daten wie z.B. den Linux-Kernel über tftp laden und ausführen kann. Dies ist vor allem in der Entwicklung sehr praktisch, da man nicht nach jeder Änderung den Kernel neu ins Flash schreiben muss. Um dem Kernel Parameter beim Start zu übergeben, kann man eine Systemvariable mit den Parametern als wert setzen.

4.3.3 Micromonitor

Micromonitor[4] ist ein Bootloader der Firma Microcross, Inc.. Er steht unter einer Open Source Lizenz, die lediglich fordert dass die Änderungen an den Autor zurückgemeldet werden (siehe Kommentar im Quellcode vom uMon). Micromonitor wird als Dienstleistung auf verschieden Hardware portiert. So auch auf das CSB737 von Congent.

Micromonitor ist mehr als nur ein einfacher Bootloader. Er stellt ein komplettes kleines mini Betriebssystem zur Verfügung.

[3] http://git.denx.de/cgi-bin/gitweb.cgi?p=u-boot.git;a=summary
[4] http://microcross.com/html/micromonitor.html

- Eigenes Flash-Dateisystem
- tftp Client und Server
- gdb-schnittstelle
- scripting Funktionen

Das Ausführen von Programmen und Skripten beim Systemstart wird über einen Flag im Dateisystem geregelt. Hier ist auch gleich eine Wartezeit um das automatische Ausführen abzubrechen mit integriert.

Um Linux auf einem ARM zu booten sind einige Dinge zu beachten (siehe (King, 2004, Booting ARM Linux)). Dies fällt einem beim Verwenden von U-Boot gar nicht auf, da U-Boot sich von selbst darum kümmert. Versucht man aber den Linux-Kernel einfach auszuführen quittiert er es mit einer Fehlermeldung. Der Linux-Kernel erwartet in Register0 eine 0 und in Register1 die MACH_TYPE-Nummer (siehe Abschnitt 4.4.1). In Register2 erwartet der Kernel die Adresse einer Konfigurationsstruktur ATAG. Diese kann man für den ersten Test aber weglassen. Man erhält bei Fehlen der Struktur vom Kernel lediglich eine Warnung.

4.4 Betriebssystem

Als Betriebssystem kommt Linux zum Einsatz. Soll der Kernel auf einer anderen Architektur als i386 laufen, muss diese beim Starten des Konfigurationstool als Parameter angeben. Im unseren Fall ist die Architektur ARM deswegen müssen wir die Konfiguration wie folgt starten:

```
make ARCH=arm {config|menuconfig|xconfig}
```

4.4.1 Treiber

Congent hat Bill Gatliff mit der Portierung von Linux auf das CSB737 beauftragt. Zum Zeitpunkt der Implementierung war der von Gatliff veröffentlichte Patch (Mail vom 2008-05-12 auf der linux-arm-kernel mailingliste[5]) noch nicht ganz vollständig. Es fehlten auch die Konfiguration die für den Betrieb des CSB737 mit dem Entwicklungsbord CSB703 nötig sind.

Die Konfiguration der Treiber findet über Strukturen statt, in denen die Konfigurationsparameter stehen. So lassen sich die Treiber möglichst systemunabhängig programmieren.

[5] http://thread.gmane.org/gmane.linux.ports.arm.kernel/41451

Für jedes Bord gibt es eine C-Datei. Die einzelnen Bords werden über einen Mach-Type-Nummer unterschieden. Diese Nummer wird in einer Datenbank[6] verwaltet. Jeder kann für sein System eine Nummer beantragen. Für die SWCU wurde keine neue Nummer beantragt da es sich bei der Platine um ein Einzelstück handelt.

```
1  MACHINE_START(CSB737, "Cogent CSB737 SOM (AT91SAM9263)")
2      /* Maintainer: Bill Gatliff <bgat@billgatliff.com> */
3      .phys_io        = AT91_BASE_SYS,
4      .io_pg_offst    = (AT91_VA_BASE_SYS >> 18) & 0xfffc,
5      .boot_params    = AT91_SDRAM_BASE + 0x100,
6      .timer          = &at91sam926x_timer,
7      .map_io         = csb_map_io,
8      .init_irq       = csb_init_irq,
9      .init_machine   = csb_board_init,
10 MACHINE_END
```

Listing 4.1: Initiale Strucktur für das CSB737

Die im Mikrocontroller integrierte Hardware ist über einen Prozessorbuss an den Prozessor angebunden. Diese Geräte werden im Kernel als „Plattform Devices“ eingebunden. Hier finden sich alle Geräte, die sich nicht einem der Bussysteme die im Kernel definiert sind zuordnen lassen (Quade und Kunst, 2006, Kap. 7.2.2 Geräte anmelden).

4.4.1.1 LEDs

Die LEDs sind über eine GPIO angeschlossen. Für diese Devices ist der Treiber “leds-gpio” bzw. für die Helligkeitsregelung über PWM der Treiber “leds-atmle-pwm” zuständig.

```
1  static struct gpio_led csb_leds[] = {
2      {
3          .name           = "rot1",
4          .gpio           = AT91_PIN_PB27,
5          .active_low     = 0,
6          .default_trigger = "heartbeat",
7      },
8  };
```

Listing 4.2: Konfiguration der Taster im Linux-Kernel

Über trigger können die LEDs kernelseitig gesteuert werden. Der eingestellt default Trigger lässt sich später jeder Zeit ändern oder deaktivieren.

[6] http://www.arm.linux.org.uk/developer/machines/

4.4.1.2 Taster

Zum entprellen der Taster ist es wichtig, dass der Pullup im AT91SAM9263 aktiviert ist. Ansonsten ist bei geöffnetem Taster ein undefiniertes Signal am Eingang, welches durch Störungen zwischen low und high zufällig wechselt. Durch den hochohmigen Pullup Widerstand wird das Signal auf high gezogen und ist nur bei gedrückten Taster low. Deswegen ist in der Konfigurationsstruktur des Treibers (siehe Listing 4.3) “active_low” auf 1 zu setzen. Um das Prellen der Tasten währen des Drückes zu verhindern, verfügt der AT91SAM9263 über einen Glitchfilter (Atmel, 2007b, Kap. 31.4.9 Input Glitch Filtering) der ein Signal nur akzeptiert, wenn es länger als 1/2 Tackt stabil ist.

```
1  static struct gpio_keys_button swcu_buttons[] = {
2      {
3          .code        = KEY_1,
4          .gpio        = AT91_PIN_PB18,
5          .active_low  = 1,
6          .desc        = "left",
7          .type        = EV_KEY,
8      },
9  };
10
11 static void __init swcu_add_device_buttons(void)
12 {
13     at91_set_GPIO_periph(AT91_PIN_PA15, 0); /* left button */
14     at91_set_deglitch(AT91_PIN_PA15, 1);
15
16     platform_device_register(&swcu_button_device);
17 }
```

Listing 4.3: Konfiguration der Taster im Linux-Kernel

Über den “code” kann festgelegt werden, welchem Event der Tastendruck entspricht. Das Eventsystem von Linux verwaltet verschiedene Ereignisse, die im System auftreten können. Dazu gehören auch die Tastartur und Mausevents. Die zu Verfügung stehen Events und Event-Typen sind in der linux/input.h definiert. In unserem Fall werden die Keycodes der Tasten 1, 2 und 3 (einer normalen Tastertur) für die Bedienung des Displays benötigt.

4.4.1.3 Display

Alle drei Platinen (AT91SAM9263-EK, CSB703 und SWCU) verwenden unterschiedliche Displays. Die grundlegende Konfiguration wurde vom AT91SAM9263-EK übernommen.

Die Konfiguration gliedert sich in drei Strukturen „fb_videomode“, „fb_monspecs“ und „atmel_lcdfb_info“ (siehe Abb 4.2). In den ersten beiden Strukturen wird der Frambuffer mit den Displaytimings konfiguriert. Die Angaben hierzu wurden den Datenblättern entnommen und sind in Tabelle 4.4.1.3 zusammengefasst.

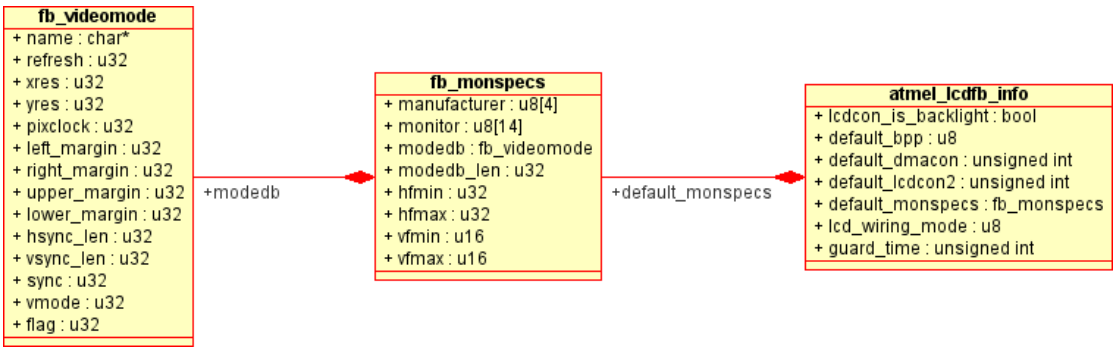


Abbildung 4.2: UML-Diagramm Strukturen der Displaykonfiguration

	TX09D70VM1CCA	*TX07D09VM1CBB*	*OSD043TN13*
Bildwiederholfrequenz	60Hz	60Hz	60Hz
Pixelclock	4,965MHz	5,6MHz	9MHz
Horizontal min	10,92kHz	18,57kHz	17,14kHz
Horizontal max	22,12kHz	22,73kHz	
Vertikal min	52Hz	54Hz	59,94Hz
Vertikal max	68Hz	68Hz	
links	1	11	3
rechts	33	17	3
oben	1	7	2
unten	0	19	2
hsync	5	1	42
vsync	1	1	11
Auflösung	240x320	240x320	480x272

Tabelle 4.2: Display Timings

In der Struktur „atmel_lcdfb_info“ wird der Displaycontroller des AT91SAM9263 konfiguriert. Im folgenden werden nur die geänderten Parameter beschrieben.

lcdcon_is_backlight Hier wird angegeben ob der LCD-Backlight-Ausgang des Displaykontrollers genutzt werden soll. Für das Entwicklungsbord CSB703 muss er deaktiviert werden, da hier der PWM3 für die Hintergrundbeleuchtung genutz wird.

lcd_wiring_mode Gibt die Pinnbelegung des Pixelbusses an. Möglche Werte sind AT-

MEL_LCDC_WIRING_BGR und ATMEL_LCDC_WIRING_RGB. Alle drei Platinen verwenden die Belegung BGR (Blau, Grün, Rot).

4.4.2 Dateisystem

Als Dateisystem kommt das für Flashspeicher optimiert JFFS2 zum Einsatz. Es handelt sich um eines der verbreitetsten Dateisysteme für embedded Systeme und wird auch in der Beispielkonfiguration von Atmel eingesetzt.

Seit dem Kernel 2.6.26 kann man zwischen der softwareseitigen Fehlerkorrektur und der hardwareseitigen wählen. Vorher war nur die Softwareseitige implementiert. Wir nutzen die Hardwarefehlerkorrektur um möglichst viel Arbeit in die Hardware auszulagern. Vor dem Einhängen des Flashs in das Dateisystem muss der Flash gelöscht werden. Dies geschieht mit dem mtd-tool „flash_erraseall“. Anschließend kann der Flash als Dateispeicher genutzt werden. Da ein automatisches erzeugen des Images nicht möglich war, wurden die Daten per Hand von dem NFS-Dateisystem kopiert.

4.5 CAN-Treiber

Der CAN-Treiber für den AT91SAM9263 stellt die Schnittstelle zwischen Hardware und CAN-Socket dar. Das CAN-Protokoll ist komplett in die Netzwerkarchitektur von Linux integriert. Somit handelt es sich beim CAN-Treiber um einen Netwerkkarten-Treiber für das CAN-Netzwerk. Aus Anwendungssicht macht es fast keinen Unterschied, ob die Daten über das Ethernet oder über den CAN-Bus übertragen werden. Weitere Informationen zur Benutzung des CAN-Sockets in Anwendugen finden sie in der Bachelorabeit Lorenz (2008). Der Unterschied zwischen dem CAN-Bus und anderen Netzwerkprotokollen ist, dass der CAN-Bus alle Nachrichten als Broadcast versendet. Es gibt bei den CAN-Nachrichten nur eine Absender aber keine Empfänger ID. Mit der CAN-ID wird auch der Inhalt der Nachricht festgelegt. Der Empfänger filtert sich anhand der ID die Nachrichten die ihn interessieren heraus.

Informationen zu der Implementierung eines Netzwerkartentreibers siehe (Quade und Kunst, 2006, Kap. 8.3 Netzwerk-Subsystem).

Der CAN-Treiber setzt auf die aktuelle Version (SVN Rev. 751) des Socketcan[7]. Gegenüber der Version des Socketcan, welche mit der Linux Kernel Version 2.6.25 Einzug erhalten hat, wurde sie um einige Funktionen erweitert, die die CAN-Treiber Implementierung vereinfachen. Zu den Funktionen gehört auch eine auf (Hartwich und Bassemir, 1999, The Configuration of the CAN Bit Timing) basierende Berechnung des Bittimings.

[7] http://developer.berlios.de/projects/socketcan

4.5.1 Initialiesirung des Treibers

Die Initialiesirung des Treibers findet in der Probe-Funktion statt. Der Ablauf ist in Abb. 4.3 dargestellt. Damit der Treiber auf verschiedenen Systemen mit dem gleichen CAN-Controler aber unterschiedlicher Adresstruktur laufen kann, sollte er möglichst portabel gehalten werden. Deswegen holt sich der CAN-Treiber die benötigten Resourcen mit Hilfsfunktionen aus der Hardwarekonfiguration (siehe Kap. 4.4.1). Beim CAN-Treiber bestehen die Resourcen aus der Adresse des memory mapped CAN-Controlers und des IRQs.

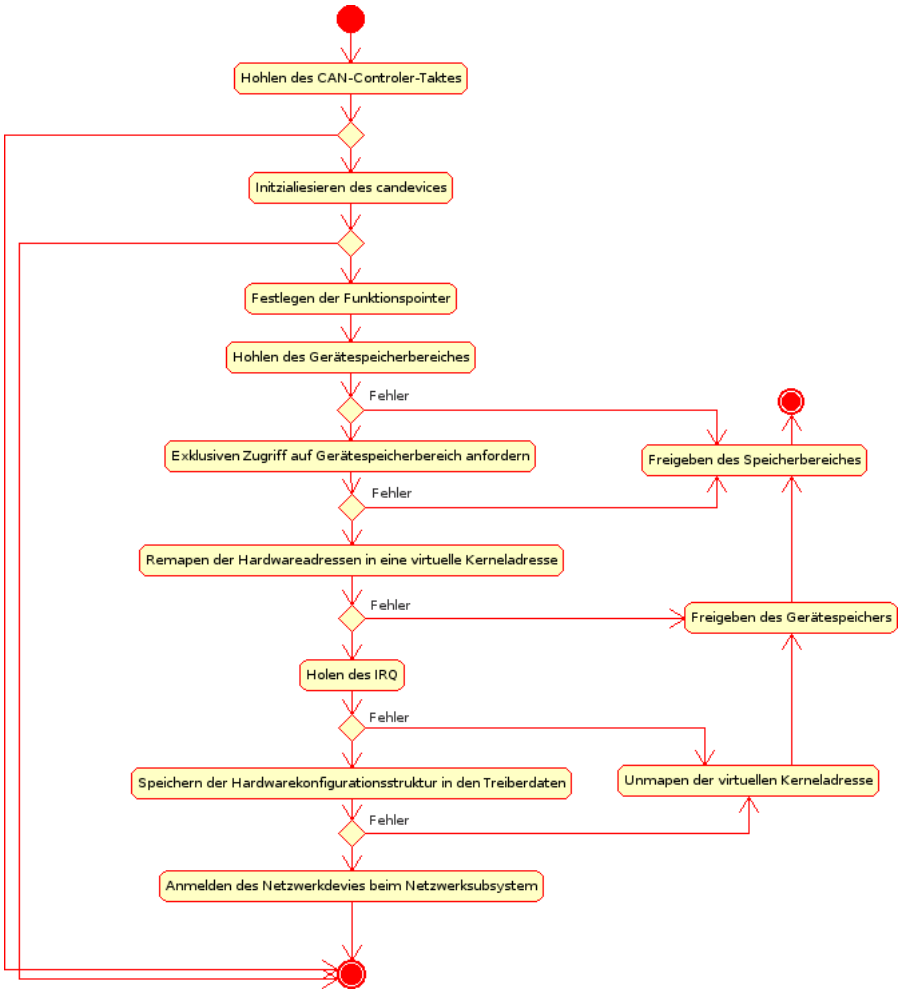


Abbildung 4.3: Aktivitätsdiagramm-CAN Probe

Die Funktion alloc_candev(sizeof(struct at91can_priv)); ist äquivalent der Funktion alloc_netdev (...) ; und verwendet sie auch intern. Zusätzlich werden noch die für einen CAN-Treiber wichtigen Konfigurationen in die Struktur geschrieben.

Die Hardwarekonfiguration des CAN-Treibers besteht aus einer Funktion zum An- und Abschalten des CAN-Trancivers. Diese Funktion steht nicht bei jedem CAN-Tranciever zur Verfügung oder wird nicht genutzt somit muss sie hardwarespezifisch implementiert werden. Eine Implementierung für den AT91SAM9263-EK ist in Listing 4.4.

```
1  static void sam9263ek_transceiver_enable(int enable)
2  {
3      if (enable) {
4          at91_set_gpio_output(AT91_PIN_PA18, 1); /* CANRXEN */
5          at91_set_gpio_output(AT91_PIN_PA19, 0); /* CANRS */
6      } else {
7          at91_set_gpio_output(AT91_PIN_PA18, 0); /* CANRXEN */
8          at91_set_gpio_output(AT91_PIN_PA19, 1); /* CANRS */
9      }
10 }
11
12 static struct at91_can_data ek_can_data = {
13     .transceiver_enable = sam9263ek_transceiver_enable,
14 };
```

Listing 4.4: Die enable/disable Signale sind hardwarespezifisch. Sie stehen in der Hardwarekonfiguration und werden dem Treiber beim Laden übergeben.

Das Netzwerksubsystem erwartet in der Hardwarekonfiguration Funktionspointer für verschiedene Funktionen.

Öffnen und Schließen Diese Funktion wird bei einem Aktivieren/Deaktivieren des Netzwerkgerätes mit ifconfig <dev> up/down (oder ip link set <dev> up/down) aufgerufen.

Transmit Wird aufgerufen wenn Daten versendet werden sollen. Siehe 4.5.5.

Bittiming Setzt das Bittiming in der Hardware. Siehe 4.5.2.

4.5.2 Bittiming

Das Bittiming legt die Länge eines einzelnen Bits auf dem CAN-Bus fest. Diese Zeit muss bei allen Stationen nahezu gleich sein. Bei einer CAN-Busgeschwindigkeit von 125kBit/s ist die maximale Abweichung mit 1,58% angegenben Robert Bosch GmbH (1991). Bei höheren Geschwindigkeiten liegt der maximal zulässige Fehler deutlich niedriger. In den Hawks-Rennwagen kommt zur Zeit eine CAN-Busgeschwindigkeit von 500kBit/s zum Einsatz. Bei dieser Geschwindigkeit beträgt die Länge eines einzelnen Bits $\frac{1}{500kBit} = 2ns$.

Um den optimalen Abtastzeipunkt für das CAN-Bussignal zu finden, wird das Bit in 4 Segmente unterschiedlicher Länge unterteilt Hartwich und Bassemir (1999). Die Dauer der einzelnen Segmente wird in Time Quanta angegeben. Zum Setzen des Bittimings muss ein

Teiler (BRP) für den Systemtakt (MCK) gefunden werden, der das Bit in eine möglichst hohe Anzahl von Time Quanta unterteilt. Bei einen unrunden Systemtakt wie bei unserem System muss eine möglichst gute Annäherung gefunden werden. Der Teiler kann nur einen ganzahligen Wert annehmen.

$$\frac{1}{\frac{MCK}{round(BPR)}} * TimeQuanta \simeq 2ns \tag{4.1}$$

Die Berechnung der einzelnen Teiler für den Systemtakt des AT91SAM9263-EK (99,959589 MHz) und des CSB737 (99,328 MHz) finden sich in Tabelle 4.3. In der Tabelle wird das Bit in unterschiedliche viele Time Quanta zerlegt und anschließend errechnet welcher Teiler für diese Time Quanta-Länge nötig wäre.

$$\frac{1}{\frac{500kBit/s}{TimeQunata}} * MCK \tag{4.2}$$

Anschließend wird mit der Formel 4.1 die sich daraus ergebende Bitlänger errechnet. Der Teiler mit der kleinsten Abweichung und der größten Time Qunata Anzahl ist die beste Einstellung für den CAN-Bus.

Wie am Anfang des Abschnittes bereits erwähnt, gibt es im Socketcan-Subsystem eine Funktion zur Berechnung des Bittimings. Über die Berechnung des Bittimings gibt es zur Zeit auf der Mailingliste vom Socketcan eine rege Diskusion. Bei dieser Diskusion geht es darum, ob eine Funktion zur Berechnung in der Kernel oder in den Userspace gehört. Erst wenn diese Frage entschieden ist, soll eine neue Version Einzug in der Kernel erhalten. Zur Zeit gibt es diese Funktion. Für die Berechnung des Bittiming für das AT91SAM9263-EK funktioniert sie. Für das CSB737 funktioniert sie nicht. Der Grund liegt im unterschiedlichen Systemtakt. Während das AT91SAM9263-EK einen Systemtakt von 99,959589 MHz hat sind es beim CSB737 nur 99,328 MHz. Eigenlich ist der Unterschied nicht besonders groß. Lässt aber beim Bittiming für den CAN-Bus den Fehler so groß werden, dass die Funktion ihn als nicht gültig erkennt. Aus diesem Grund wurde für das CSB737 das Bittiming von Hand errechnet.

4.5.3 Interuptbehandlung

Für den CAN-Controller steht eine einziger Interupt zur Verfügung. So müssen Datenempfangs / Sendeinterupts gemeinsam mit den Fehlerinterupts in einer Interuptservicerotine (ISR) abgearbeitet werden. Auf die Behandlung der Datenempfangs / Sendeinterupts wird in den nachfolgenden Abschnitten 4.5.4 und 4.5.5 eingegangen.

	MCK 99,959589 MHz		MCK 99,328 MHz	
Time Quanta	*BRP*	*Bit Time*	*BRP*	*Bit Time*
8	24,990	2,001ns	24,832	2,014ns
9	22,213	1,981ns	22,073	1,993ns
10	19,992	2,001ns	19,866	2,014ns
11	18,174	1,981ns	18,060	1,993ns
12	16,660	2,041ns	16,555	2,054ns
13	15,378	1,951ns	15,281	1,963ns
14	14,280	1,961ns	14,190	1,973ns
15	13,328	1,591ns	13,244	1,963ns
16	12,495	1,921ns	12,416	1,933ns
17	11,760	2,041ns	11,686	2,054ns
18	11,107	1,981ns	11,036	1,993ns
19	10,522	2,091ns	10,456	1,913ns
20	9,996	2,001ns	9,933	2,014ns
21	9,520	2,101ns	9,460	1,903ns
22	9,087	1,981ns	**9,030**	**1,993ns**
23	8,692	2,071ns	8,637	2,084ns
24	8,330	1,921ns	8,277	1,933ns
25	**7,997**	**2,001ns**	7,946	2,014ns

Tabelle 4.3: CAN-Bus Timing

Der CAN-Controler kann sich in verschieden Zuständen befinden:

- Aktiv
- Error Passiv
- Bus off

Wechselt er den Zustand löst er einen Interupt aus. Das Statusregister bleibt solange gesetzt wie sich der CAN-Controller in dem Zustand befindet. Um einen Interrupt nicht doppelt zu bearbeiten wird am Anfang der ISR die Statusflags von allen Interupts, die nicht aktiviert sind ausgeblendet. Wird ein Zustandswechsel-Interrupt ausgelöst wird der Interrupt des Zustandes in den gewechselt wurde, deaktivert und die der anderen Zustände aktiviert. So wird verhindert das der Interrupt noch mal behandelt wird.

4.5.4 Datenempfang

Für den Datenempfang werden 8 der 16 Message-Obejct-Buffer (MOBs) verwendet. Der Datenemfpang geschieht in der Interuptservicerotine (ISR). Für jeden MOB gibt es eine Status

bit im Statusregister. Diese Bits müssen einzeln abgeprüft werden und wenn das Bit gesetzt ist die ausgelagerte Recivefunktion mit der Nummer des MOBs aufgerufen werden. Meistens werden nur die ersten MOBs verwendet. Um nicht immer alle MOBs prüfen zu müssen werden die schon abgearbeiten Bits rausgeschoben. Durch diese Maßnahme kann das Prüfen abgebrochen werden sobald kein Bit mehr 1 ist. Es müssen nicht alle MOB-Statusbits einzel geprüft werden. Siehe Listing 4.5.

```
1 mrxsr = statusregister & AT91_CAN_MB_RX;
2 i = 0;
3 while(mrxsr && i < 8) {
4     if((mrxsr >> i) & 0x1) {
5         can_rx(dev, i);
6     }
7     i++;
8 }
```

Listing 4.5: Abprüfen der MOBs in der ISR

In der Recivefunktion muss zu erst Speicher für die empfangenen Daten beim Netzwerk-Subsystem angefordert werden.

```
skb = dev_alloc_skb(sizeof(struct can_frame));
```

In die Struktur muss zuerst geschrieben werden von welchem Device die Daten stammen und da wir nicht ein standart Ethernet haben auch das Protokoll zu dem das Paket gehört.

```
skb->dev = dev;
skb->protocol = htons(ETH_P_CAN);
```

Anschließend können wir uns mit cf = (struct can_frame *)skb_put(skb, sizeof(struct can_frame)); den Speicherbereich des CAN-Frames aus der Struktur geben lassen und diesen mit den CAN-Daten füllen. Danach wird der Status des MOBs wieder auf Empfang gestellt (hiermit auch wird der Interupt zurückgesetzt) und die Daten mit netif_rx (skb); wie beim Netzwerktreiber an das Netzwerk-Subsystem übergeben.

Anschließend muss noch die Statistik aktualiesiert werden.

```
dev->last_rx = jiffies;
stats->rx_packets++;
stats->rx_bytes += cf->can_dlc;
```

4.5.5 Senden von Daten

Für das Versenden von Daten wird von dem Netzwerk-Subsystem eine bei der Initialisierung des Treibers festgelegte Funktion aufgerufen. Für das Versenden der Nachrichten werden

die anderen 8 MOB verwendet. Zuerst muss der erste freie MOB gefunden werden. Wenn der gefundene freie MOB der letzte von den 8 ist, muss dem Netzwerk-Subsystem mitgeteilt werden das vorerste keine weiteren Daten angenommen werden können. Dies geschieht mit der Funktion netif_stop_queue(dev);.

Ist der MOB mit den zu versendenden Daten gefüllt teilt man dem CAN-Controller dies durch setzen eines Statusflags mit. Gleichzeitig muss auch der TX-Interrupt für den MOB aktiviert werden, damit wir mitbekommen wenn die Nachricht versendet wurde.

Machmal ist ein locales Echo der CAN-Daten erwünscht. Dies ist meist dann der Fall, wenn auf dem System mehre Anwendugen laufen und diese an den Daten einer anderen Anwendung interesiert sind. Mit der Funktion can_put_echo_skb(skb, dev, 0); werden die Daten bei Bedarf zurück an das Netzwerk-Subsystem geben, welches sie als ankommende Nachricht an die Anwendungen verteilt.

Wenn die Nachricht versendet wurde wird ein Interrupt ausgelöst. In der Interruptbehandlung wird der Interrupt wieder deaktiviert. Sollte das Versenden von Nachrichten gestoppt worden sein kann es nun wieder gestartet werden.

```
if (netif_queue_stopped(dev))
    netif_wake_queue(dev);
```

Das TimeTrigert versenden von Daten wurde nicht Implementiert.

4.5.6 Test

Aus Zeitmangel konnte der CAN-Treiber nicht ausgibig getestet werden. Folgende Funktionen wurden getestet:

- Empfange von Nachrichten mit einer Buslast von >85% ohne das Nachrichten verlohren gehen.
- Versenden von einzelnen Nachrichten

5 Ausblick

Diese Kapitel fasst die Entwicklung des Lenkraddisplays zusammen und Zeigt auf, welche Änderungen in der Zukunft wünschenswert sind.

5.1 Fazit

In dieser Bachelorarbeit wurde der erste Prototyp eines Lenkraddisplays mit TFT-Display erstellt. Durch die Integration in das bestehende Telemetriesystem ist das Lenkraddisplay sehr flexibel. Es ließe sich ohne große Probleme auch im Hawk07 einsetzen. Hierfür müssten lediglich die angezeigten Sensoren umkonfiguriert werden, da im Hawk07 noch nicht alle Sensoren zur Verfügung stehen.

Bei der Umsetzung der Arbeit haben sich einige Probleme gezeigt, die vor allem mit der Fertigung der Hardware zusammenhängen. Diese Probleme zu lösen ist für die Telemetrie des Hawksrennwagens sehr wichtig, da die Miniturisierung der Bauteile weiter vorranschreitet. Desweiteren hat sich gezeigt, dass die Entwicklung von Prototypen ein kostspieliges Unterfangen ist. Die Entwicklung dieses Lenkraddisplays war nur durch die Unterstützung der Karl H. Ditze-Stiftung möglich.

5.2 Vorschläge für Änderungen und die Weiterentwicklung in der Zukunft

In den Vorrangegangenem Kapitel zur Hard und Software wurden schon einige Probleme angesprochen. Im folgenden werden diese noch ein mal zusammengefasst und ergänzt. Desweiteren wird aufgezeigt, welche Weiterentwicklungen in der Zukunft wünschenswert wären. Zusätzlich zu den unten aufgefürten Verbessurungsvorschlägen gibt es weitere, die sich aus der Bachelorabeit von Daniel Lorenz (2008) ergeben. Diese werden dort beschrieben.

5.2.1 Stromversorgung Display

Die Auslastung der Spannungen 5V, 15V und -15V ist nicht optimal. Das Display zieht nur wenige mA bzw μA (siehe Tabelle 3.3). Die Spannungswandler haben bei der geringen Auslastung eine schlechte Effektivität (siehe Effektivitätsdiagramme in Linear Technology (a), Maxim (1993) und Maxim (1994)). Auch würde es eine Platzersparnis bringen wenn man diese 3 Spannungswandler durch einen speziellen für TFT-Displays ersetzen würde. Als Ersatz bietet sich der LTC3450 von Linear Technology (c) an. Zusätzlich verfügt er über einen Shutdown-Eingang über den sich das Display abschalten lässt.

5.2.2 Hintergrundbeleuchtung Display

Wie in 3.2.4 erwähnt ist die Lösung mit dem Spannungsteiler nicht optimal. Besser wäre es eine Konstantstromquelle bzw. einen extra für die Hintergrundbeleuchtung entwickelten LED-Treiber wie den LT3465 von Linear Technology (b) zu verwenden.

5.2.3 Display Flimmern

Gegen das Flimmern des Displays muss wie in 3.2.4 beschreiben ein Potentiometer zwischen die beiden Widerstände des Spannungsteilers eingefügt werden. Siehe Blockdiagramm in Hitachi (2006).

5.2.4 Montage im Lenkrad

Aus den Erfahrungen mit dem Lenkrad im Hawk08 hat sich gezeigt das es wichtig ist, dass die Platine fest im Lenkrad verschraubt ist. Das nächste Lenkrad muss also auf jeden Fall im Bereich der Platine aufschraubbar sein. Die Gefahr der Beschädigung der Platine / Display bei der Montage über einen seitlichen Einschub ist zu groß. Siehe 3.2.4 Probleme bei der Montage im Lenkrad. Ein weiterer Vorteil eines aufschraubbaren Lenkrades ist, dass sich der MiniSD-Kartenslot an einer stelle positionieren ließe, an der er von außen ohne Öffnen des Lenkrades zugänglich ist. So ließen sich die Konfigurationsdateien leichter austauschen.

5.2.5 Nutzung der 2D Beschleunigung

Der AT91SAM9263 verfügt über einen 2D Beschleunieger. Dieser wird zur Zeit nicht verwendet. Zur Entlastung ist die Implemtentation des Treibers zur Nutzung der 2D-Beschleunigung sinnvoll.

5.2.6 Implemtentation von TTCAN

Um Konfigurationseinstellungen auf dem CAN-Bus versenden zu können ist eine Unterstüzung nötig. Zur Zeit lassen sich mit dem CAN-Treiber und dem Socketcan-Subsystem keine Timetriggert-Nachrichten verschicken.

Die einfachste Möglichkeit der Implemtentation ist, die Timetriggert-Funktionalität nur im Treiber zu implementieren. Für das Lenkraddisplay würde in jedem Basiszyklus ein Zeitslot eingeräumt werden, wo er seine Nachricht versenden darf. Die Messageboxen des CAN-Controler würde vom Treiber so konfiguriert, dass die Nachricht in diesem Zeitslot versendet werden. Das Netzwerksubsystem und die Anwendung würden hiervon nichts mitbekommen.

Vom Konzept sauberer wäre es, die Timetriggert-Funktionalität in das Socketcan-Subsystem zu integrieren. Dies hätte den Vorteil dass es sich um keine anwendungsfallbezogenen Anpassung des Treibers handelt. Der Nachteil ist, dass die Implemtentation nicht ganz einfach ist. Es muss eine neue Transportprotokollschicht in den Treiber eingefügt werden.

5.2.7 Verkürzung der Bootzeit

In Kapitel 4.2 wurde erwähnt das es noch Möglichkeiten der Optimierung der Bootzeit gibt. Das Warten auf ein System, was bootet ist einen äußerst lästige Angelegenheit. Aus diesem Grunde ist es wichtig, die Bootzeit weiter zu verkürzen.

5.2.8 Build envirement

Um den Aufwand beim Update zu minimieren, ist ein automatisch erzeigtes Image von Vorteil. Hierfür muss ein Build envirement konfiguriert werden. Siehe Kapitel 4.1.

5.2.9 Export des Speichers über USB

Eine Möglichkeit um einfach Datein im Flashfilesystem auszutaschen, ist der Export des Dateisystems über die USB-Schnittstelle. Dies könnte dann wie ein USB-Stick von PC aus beschrieben werden.

Literaturverzeichnis

[Linux-RT-Sched 2008] *Real-Time group scheduling*, Juli 2008. – Linux Kernel Dokumentation 2.6.26

[Abbott 2006] ABBOTT, Dooug: *Linux for Embedded and Real-Time Applications*. 2. Elsevier, 2006. – ISBN 978-0-7506-7932-9

[Atmel 2007a] Atmel: *AT91SAM9263-EK Evaluation Board Rev. B User Guide*. Oct. 2007. – URL http://www.atmel.com/dyn/resources/prod_documents/doc6341.pdf

[Atmel 2007b] Atmel: *Datasheet AT91SAM9263*. 2007. – URL http://www.atmel.com/dyn/resources/prod_documents/doc6249.pdf

[Atmel 2008] Atmel: *AT91SAM9263-EK Hardware Files Rev. B*. 2008. – URL http://www.atmel.com/dyn/resources/prod_documents/AT91SAM9263-EKHardwarefilesrevB.zip

[Benvenuti 2005] BENVENUTI, Christian: *Understanding Linux Networks Internals*. O'Reilly, 2005. – ISBN 978-0-596-00255-8

[Bovet und Cesati 2005] BOVET, Daniel P. ; CESATI, Marco: *Understanding the Linux Kernel*. 3. O'Reilly, 2005. – ISBN 978-0-596-00565-8

[Bruyninckx 2002] BRUYNINCKX, Herman: Real-Time and Embedded Guide. (2002), Juli. – URL http://people.mech.kuleuven.be/~bruyninc/rthowto/rtHOWTO.pdf. – 0.04-build-20021211-1936

[Cogent Computer Systems 2008a] Cogent Computer Systems: *Cogent CSB703 - Hardware Reference Manual*. 1. April 2008. – URL http://www.cogcomp.com/pdfs/MicrosoftWord-csb703_hw_reference_manual_p1_0.pdf

[Cogent Computer Systems 2008b] Cogent Computer Systems: *Cogent CSB737 - Hardware Reference Manual*. 2.1. Mai 2008. – URL http://www.cogcomp.com/pdfs/MicrosoftWord-csb737_hw_ref_manual_p2_1.pdf

[Groÿ 2006] GROŸ, Carsten: *Bootloader und embedded Linux Anpassungen für ein ARM9 basierendes Mikrocontrollerboard*, Universität Ulm, Studienarbeit, Februar 2006. – URL http://www.siski.de/~carsten/pdf/2006/02/Studienarbeit.pdf

[Haase 2007] HAASE, Sebastian: *Telemetrie im Formula Studente Rennwage auf Basis von CAN-Bus, Datenspeicherung und Wireless LAN Technologie*, Hochschule für Angewandte Wissenschaft Hamburg, Bachelorarbeit, 2007

[Hartwich und Bassemir 1999] HARTWICH, Florian ; BASSEMIR, Armin: *The Configuration of the CAN Bit Timing*, November 1999. – URL http://www.semiconductors.bosch.de/pdf/CiA99Paper.pdf

[Hitachi 2005] Hitachi: *Costumer's Acceptance Spezifications TX09D70VM1CCA*. August 2005. – URL http://www.data-modul.com/de/products/tft_displays/single_tft_small/TX09D70VM1CCA.pdf

[Hitachi 2006] Hitachi: *Costumer's Acceptance Spezifications TX07D09VM1CBB*. November 2006. – URL http://www.data-modul.com/de/products/tft_displays/single_tft_small/TX07D09VM1CBB-1.pdf

[Horowitz und Hill 1996] HOROWITZ, Paul ; HILL, Winfield: *Die Hohe Schule der Elektronik*. Bd. 1. Kap. Schaltregler und Gleichstrom-Gleichstrom-Wandler, S. 399–405, Elektro-Verlag, 1996

[Jones 2004] JONES, David L.: *Leiterplatten-Layout-Tutorial (Kapitel 12)*. Juni 2004. – URL http://server.ibfriedrich.com/wiki/ibfwikide/images/3/3a/PCB_Layout_Tutorial_d.pdf

[Kelly 2008] KELLY, Michael J.: *Cogent CSB703 - Schaltplan*. 4. Cogent Computer Systems, Mai 2008

[King 2004] KING, Russell: *Booting ARM Linux*, September 2004. – URL http://www.arm.linux.org.uk/developer/booting.php. – Linux Kernel Dokumentation 2.6.26

[Kolbe 2008] KOLBE, Felix: *Redundanzkonzept eines Time-Triggered Bussystems in einem Formula Student Rennwagen: Modellierung, Implementierung und Anwendung in der Antriebsschlupfregelung*, Hochschule für Angewandte Wissenschaft Hamburg, Bachelorarbeit, August 2008

[Linear Technology a] Linear Technology: *LT1765-5 - Monolithic 3A, 1.25MHz Step-Down Switching Regulator*. – URL http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1003,C1042,C1032,C1064,P2154,D3864. – Zugriff: Aug. 2008

[Linear Technology b] Linear Technology: *LT3465/LT3465A - 1.2MHz/2.4MHz White LED Drivers with Built-in Schottky in ThinSOT*. – URL http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1003,C1094,C1766,P1929,D3108. – Zugriff: Aug. 2008

[Linear Technology c] Linear Technology: *LTC3450 - Triple Output Power Supply for Small TFT-LCD Displays*. – URL http://www.linear.com/pc/

downloadDocument.do?navId=H0,C1,C1003,C1042,C1035,P2443,D1759. – Zugriff: Aug. 2008

[Lorenz 2008] LORENZ, Daniel: *Lenkraddisplays eines Formula Student Rennwagens: von der Usability Analyse, über das Interface- und Funktionsdesign bis zur QT basierten Realisierung der Softwarearchitektur*, Hochschule für Angewandte Wissenschaft Hamburg, Bachelorarbeit, August 2008

[Love 2005] LOVE, Rober: *Linux-Kernel-Handbuch*. 2. Addison-Wesley, 2005. – ISBN 3-8273-2247-2

[Mauerer 2004] MAUERER, Wolfgang: *Linux Kernelarchitektur*. Hanser Verlag, 2004. – ISBN 3-446-22566-8

[Maxim 1993] Maxim: *MAX762 - 12V/15V or Adjustable, High-Efficiency, Low IQ, Step-Up DC-DC Converters*. November 1993. – URL http://datasheets.maxim-ic.com/en/ds/MAX761-MAX762.pdf

[Maxim 1994] Maxim: *MAX766 - -5V/-12V/-15V or Adjustable, High-Efficiency, Low IQ DC-DC Inverters*. Juni 1994. – URL http://datasheets.maxim-ic.com/en/ds/MAX764-MAX766.pdf

[Nachbaur 2008] NACHBAUR, Oliver: Scharfe Bilder - TFT-Displays richtig versorgen. In: *Elektroniknet* (2008). – URL http://www.elektroniknet.de/home/optoelektronik/fachwissen/uebersicht/l/displays/scharfe-bilder-tft-displays-richtig-versorgen/. – Zugriff: 2008-07-28

[Opdenacker 2007] OPDENACKER, Michael: *Real Time in Embedded Linux Systems*. 2007. – URL http://free-electrons.com/articles/realtime. – Creative Commons Attribution-ShareAlike 2.5 license

[Plate 2007] PLATE, Jürgen: *Linux Hardware Hackz*. Hanser Verlag, 2007. – ISBN 978-3-446-40783-1

[Quade und Kunst 2006] QUADE, Jürgen ; KUNST, Eva-Katharina: *Linux-Treiber entwickeln*. 2. dpunkt.verlag, 2006. – ISBN 3-89864-392-1

[Rebe 2008a] REBE, René: Kraftvolle Schnellstarter. In: *Linux-Magazin* 07 (2008), S. 108–109. – ISSN 1432-640 x

[Rebe 2008b] REBE, René: Prêt-à-porter. In: *Linux-Magazin* 08 (2008). – URL http://www.linux-magazin.de/themengebiete/special/embedded/pret_a_porter. – Sonderbeilage Embedded Linux. – ISSN 1432-640 x

[Rebe 2008c] REBE, René: Überall Linux. In: *Linux-Magazin* 03 (2008), S. 3–7. – URL http://www.linux-magazin.de/themengebiete/special/embedded/ueberall_linux/. – Sonderbeilage Embedded Linux. – ISSN 1432-640 x

[Rink u. a. 1998] RINK, Dr. J. ; BECKER, Michael ; FROHNA, Michael: Aus der neuen Wellt. In: *c't* (1998), Nr. 6, S. 230 ff.

[Robert Bosch GmbH 1991] Robert Bosch GmbH: *CAN Specification*. 2.0. September 1991. – URL http://www.semiconductors.bosch.de/pdf/can2spec.pdf

[SAE 2007] SAE: *2008 Formula SAE® Rules*. 2007. – URL http://students.sae.org/competitions/formulaseries/rules/rules.pdf

[Sanders 2004] SANDERS, Vincent: *Booting ARM Linux*. Juni 2004. – URL http://www.simtec.co.uk/products/SWLINUX/files/booting_article.html

[Saxena 2004] SAXENA, Deepak: *Porting Linux to a New ARM Platform*. Presented at Linux Bangalore 2004. 2004. – URL http://www.embedded-kernel-track.org/2005/porting_to_arm.pdf

[Schuckert 2007] SCHUCKERT, Simon M.: *Microcontrollerbasierte Telemetrie und Echtzeitauswertung von Sensordaten im Formula Student Rennwage*, Hochschule für Angewandte Wissenschaft Hamburg, Bachelorarbeit, 2007

[Schwebel 2002] SCHWEBEL, Robert: Echtzeit unter Linux mit RTAI. In: *Elektronik* (2002), Nr. 7, S. 72–77. – URL http://www.schwebel.de/authoring/elektronik-rtai.pdf

[Sutter 2008] SUTTER, Ed: *MicroMonitor User Manual*, Mai 2008

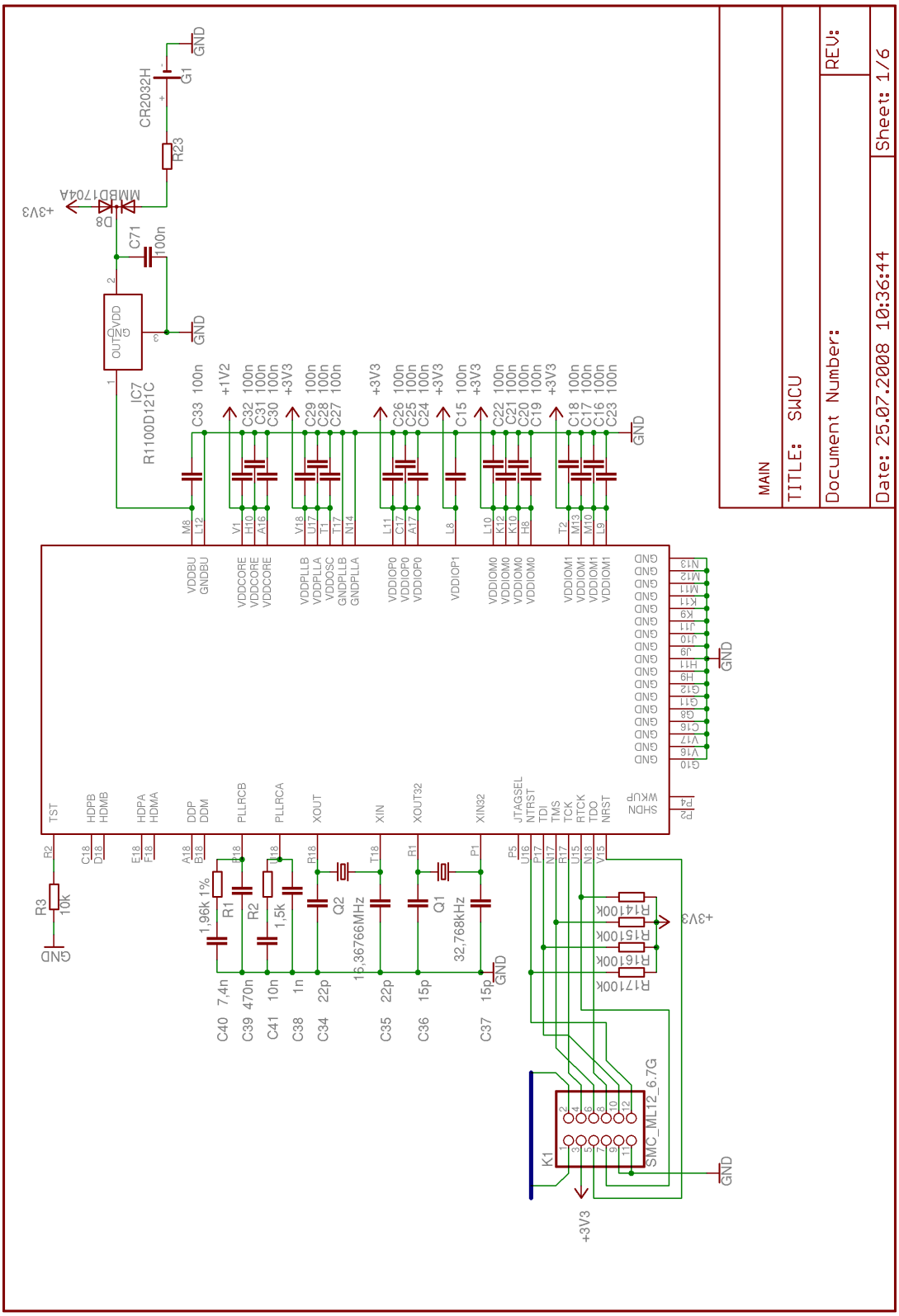
[Tanenbaum 2003] TANENBAUM, Andres S.: *Moderne Betriebsysteme*. 2. Pearson Studium, 2003. – ISBN 3-8273-7019-1

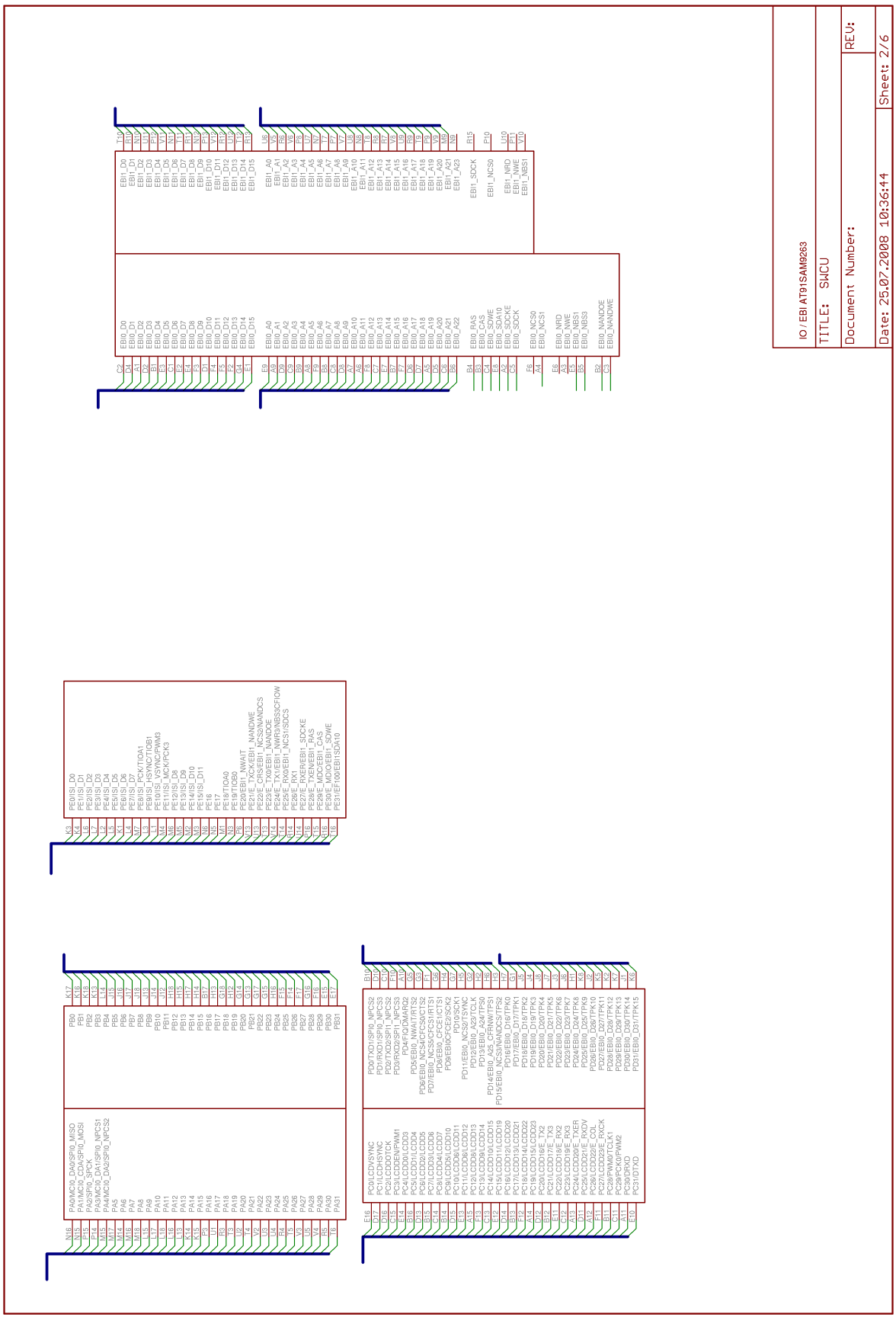
[Texas Instruments 2008] Texas Instruments: *3.3-V CAN TRANSCEIVERS*. August 2008. – URL http://focus.ti.com/lit/ds/symlink/sn65hvd234.pdf

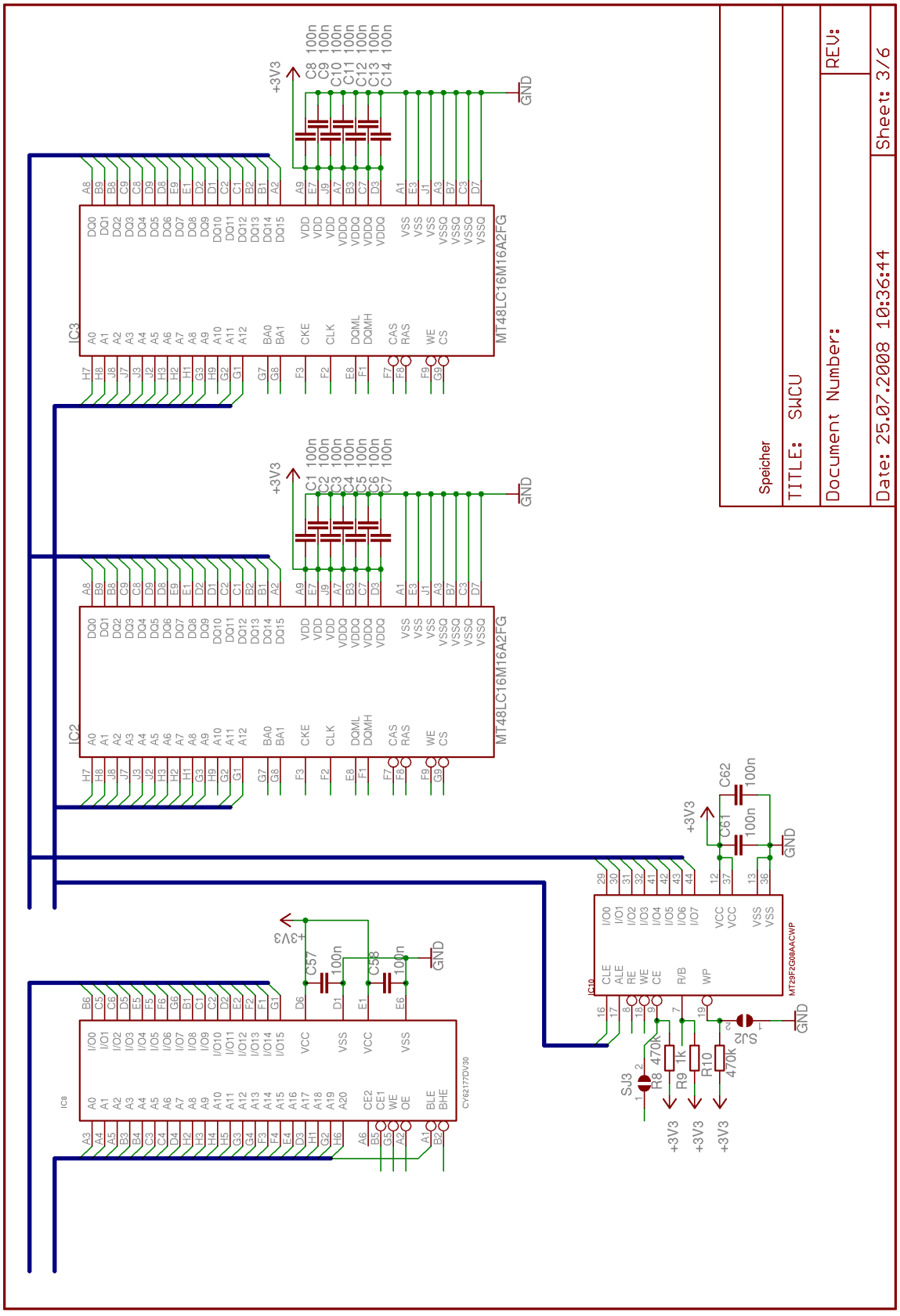
[Woodhouse] WOODHOUSE, David: *JFFS : The Journalling Flash File System*. Red Hat, Inc.. – URL http://sources.redhat.com/jffs2/jffs2.pdf. – Zugriff 2008-06-08

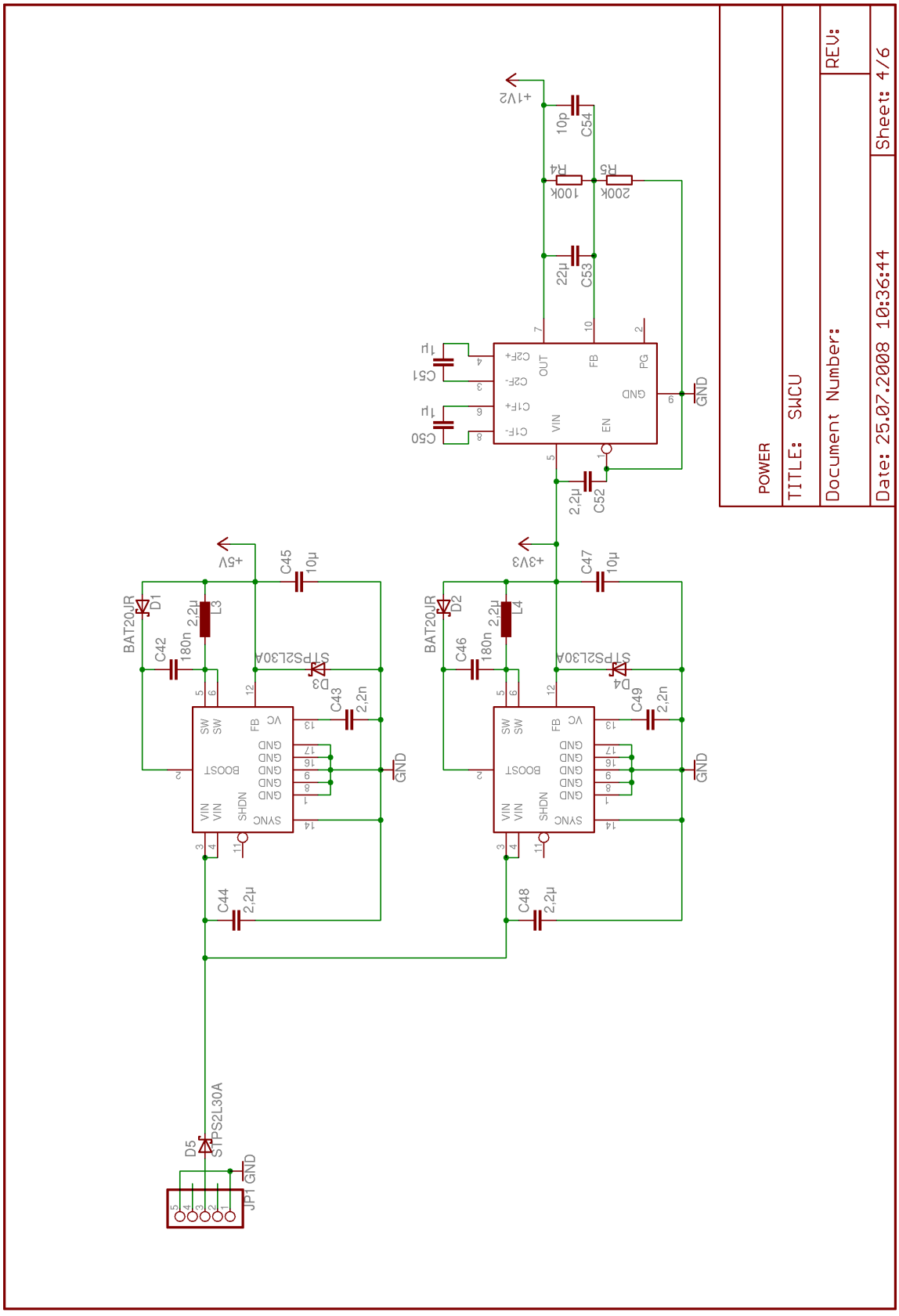
A Schaltpläne

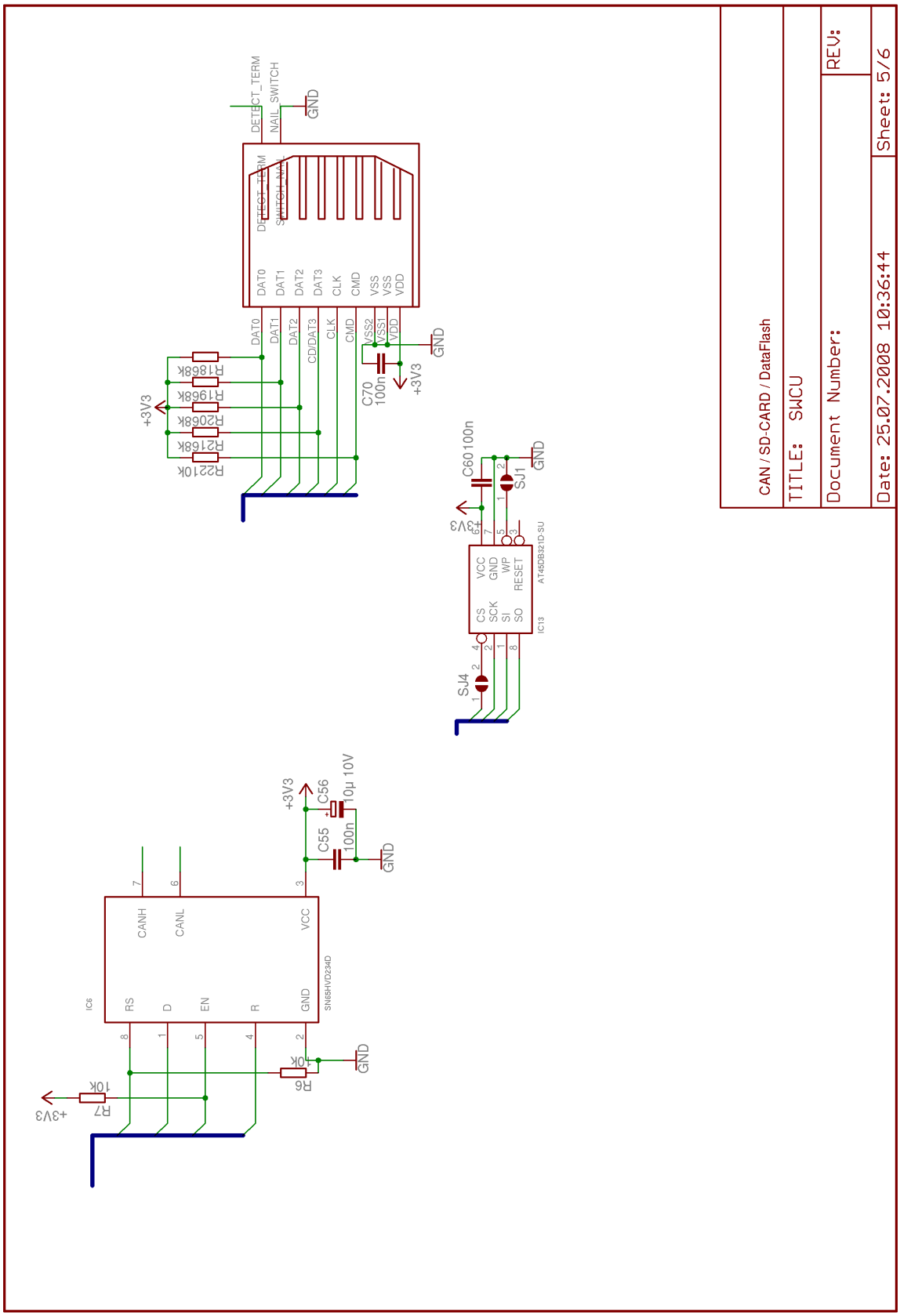
A.1 SWCU

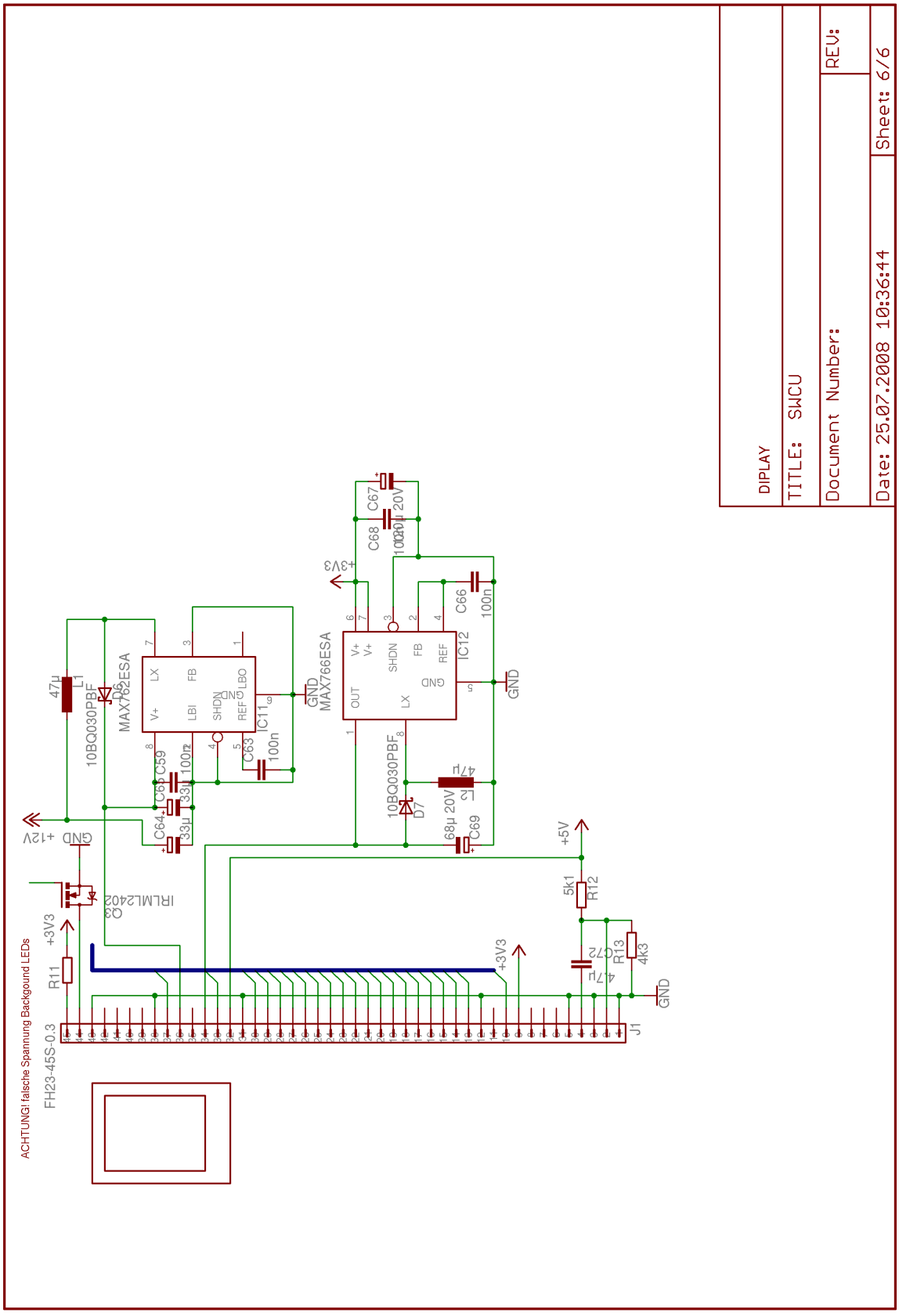



IO/EBI AT91SAM9263
TITLE: SWCU
Document Number:
REV:
Date: 25.07.2008 10:36:44
Sheet: 2/6

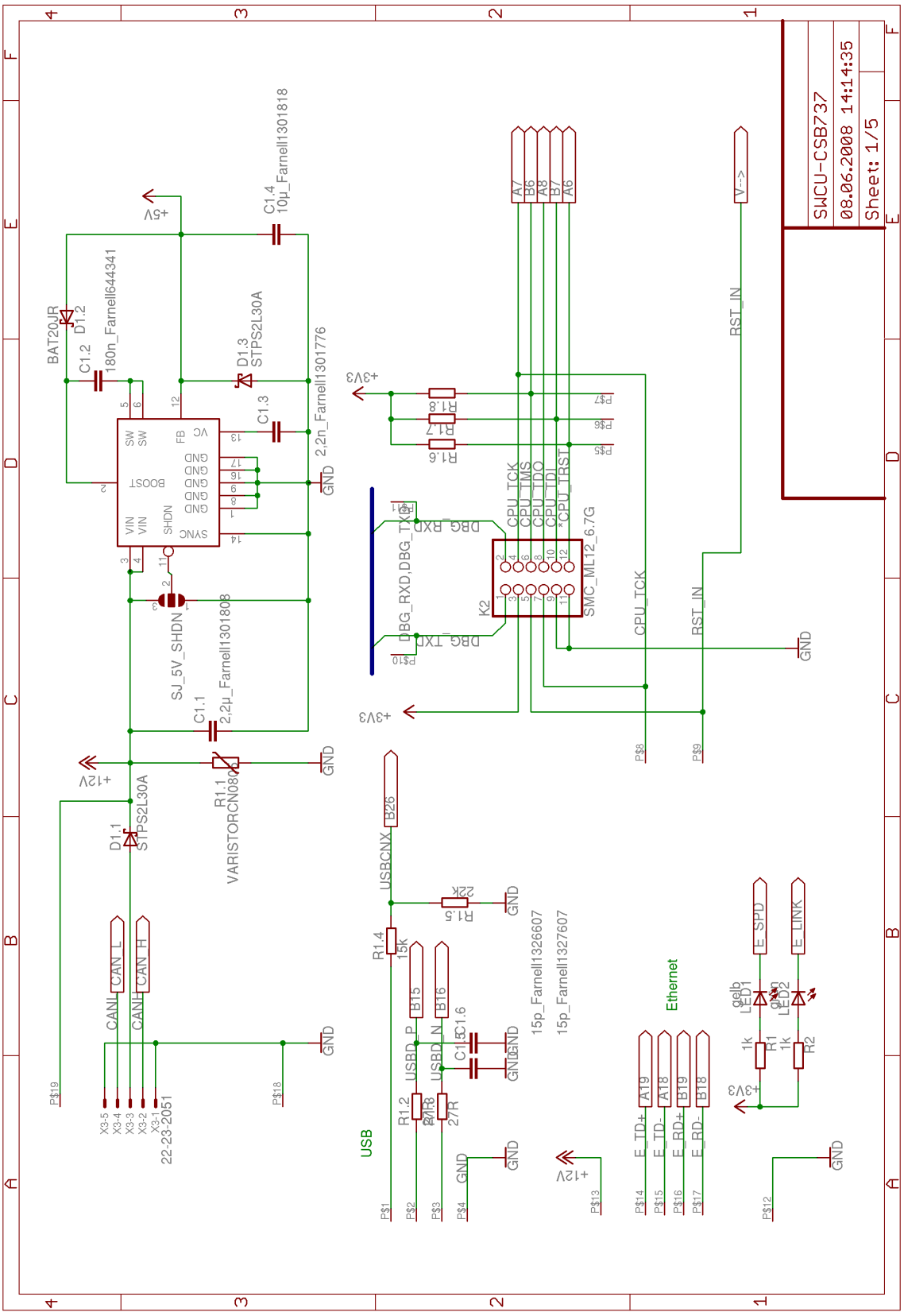

Speicher
TITLE: SWCU
Document Number:
REV:
Date: 25.07.2008 10:36:44
Sheet: 3/6
IC3
IC2
IC8
IC10
MT48LC16M16A2FG
CY62177DV30
MT29F2G08AACWP
+3V3
GND
C1 100n
C2 100n
C3 100n
C4 100n
C5 100n
C6 100n
C7 100n
C8 100n
C9 100n
C10 100n
C11 100n
C12 100n
C13 100n
C14 100n
C57 100n
C58 100n
C61 100n
C62 100n
R8 470k
R9 1k
R10 470k
SJ3
SJ4

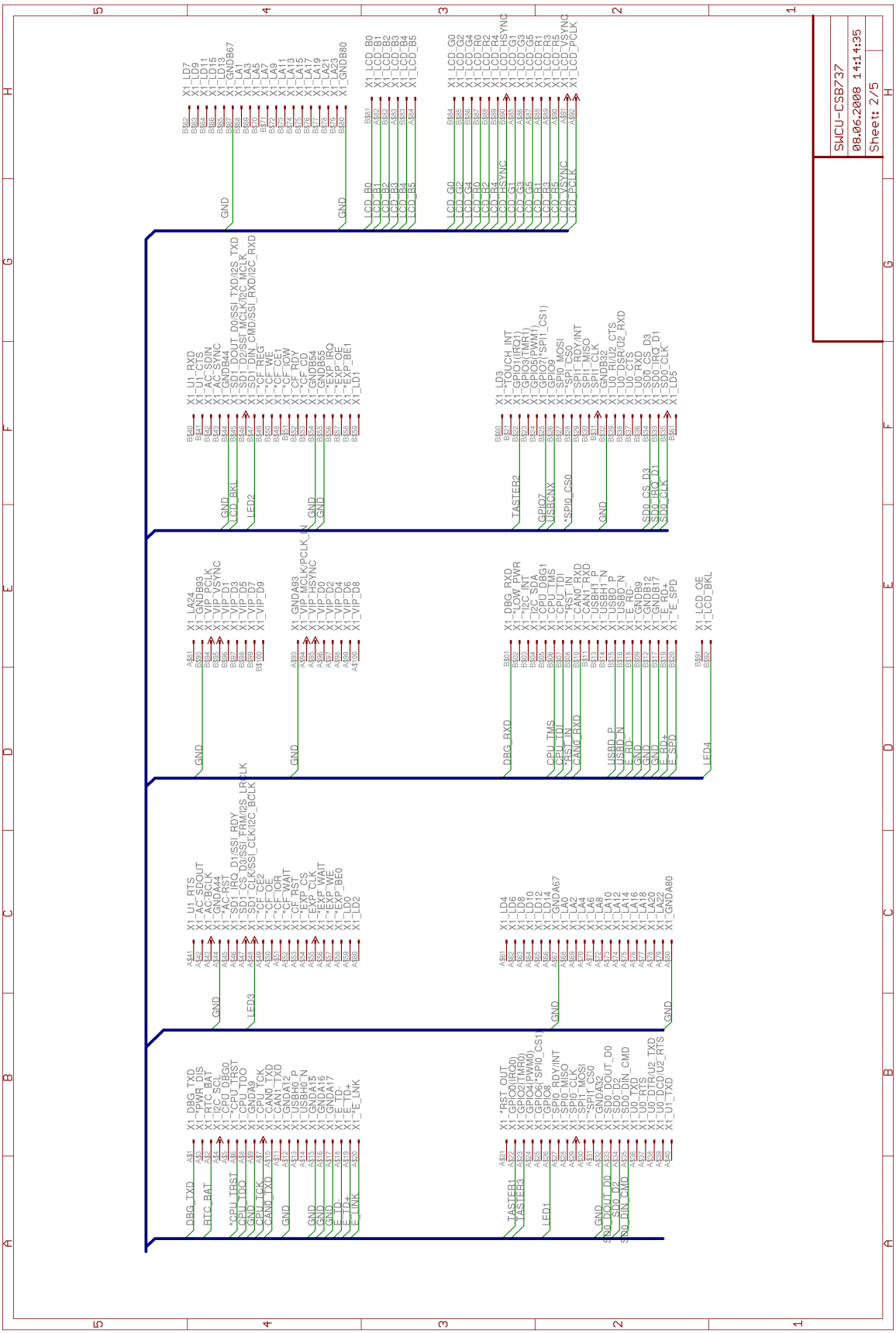

POWER
TITLE: SWCU
Document Number:
REV:
Date: 25.07.2008 10:36:44
Sheet: 4/6
+1V2
10p
C54
100K
R4
200K
R5
22µ
C53
1µ
C51
1µ
C50
C2F+
C2F-
C1F+
C1F-
OUT
FB
PG
GND
VIN
EN
2,2µ
C52
GND
+5V
C45
10µ
BAT20JR
D1
2,2µ
L3
C42
180n
STPS2L30A
D3
C43
2,2n
BOOST
SW
SW
FB
VC
GND
VIN
VIN
SHDN
SYNC
C44
2,2µ
+3V3
C47
10µ
BAT20JR
D2
2,2µ
L4
C46
180n
STPS2L30A
D4
C49
2,2n
C48
2,2µ
D5
STPS2L30A
JP1
GND

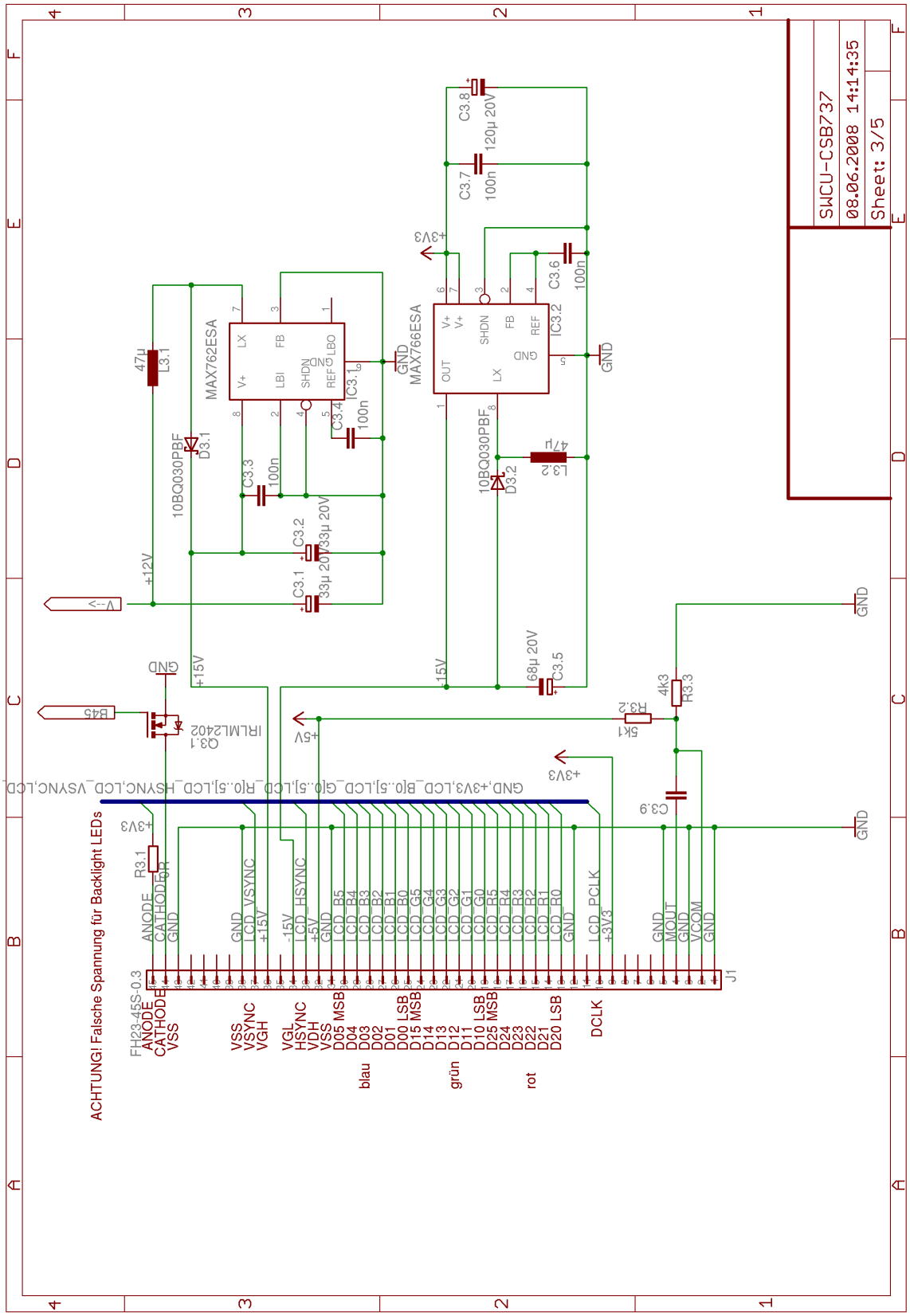

CAN / SD-CARD / DataFlash
TITLE: SWCU
Document Number:
REV:
Date: 25.07.2008 10:36:44
Sheet: 5/6
DETECT_TERM
NAIL_SWITCH
GND
DAT0
DAT1
DAT2
DAT3
CLK
CMD
VSS
VDD
CD/DAT3
VSS2
VSS1
+3V3
C70
100n
R18
68k
R19
68k
R20
68k
R21
68k
R22
10k
C60
100n
SJ1
SJ4
IC13
AT45DB321D-SU
VCC
GND
WP
RESET
CS
SCK
SI
SO
C56
10µ 10V
C55
100n
IC6
SN65HVD234D
CANH
CANL
RS
D
EN
R
R6
10k
R7
10k

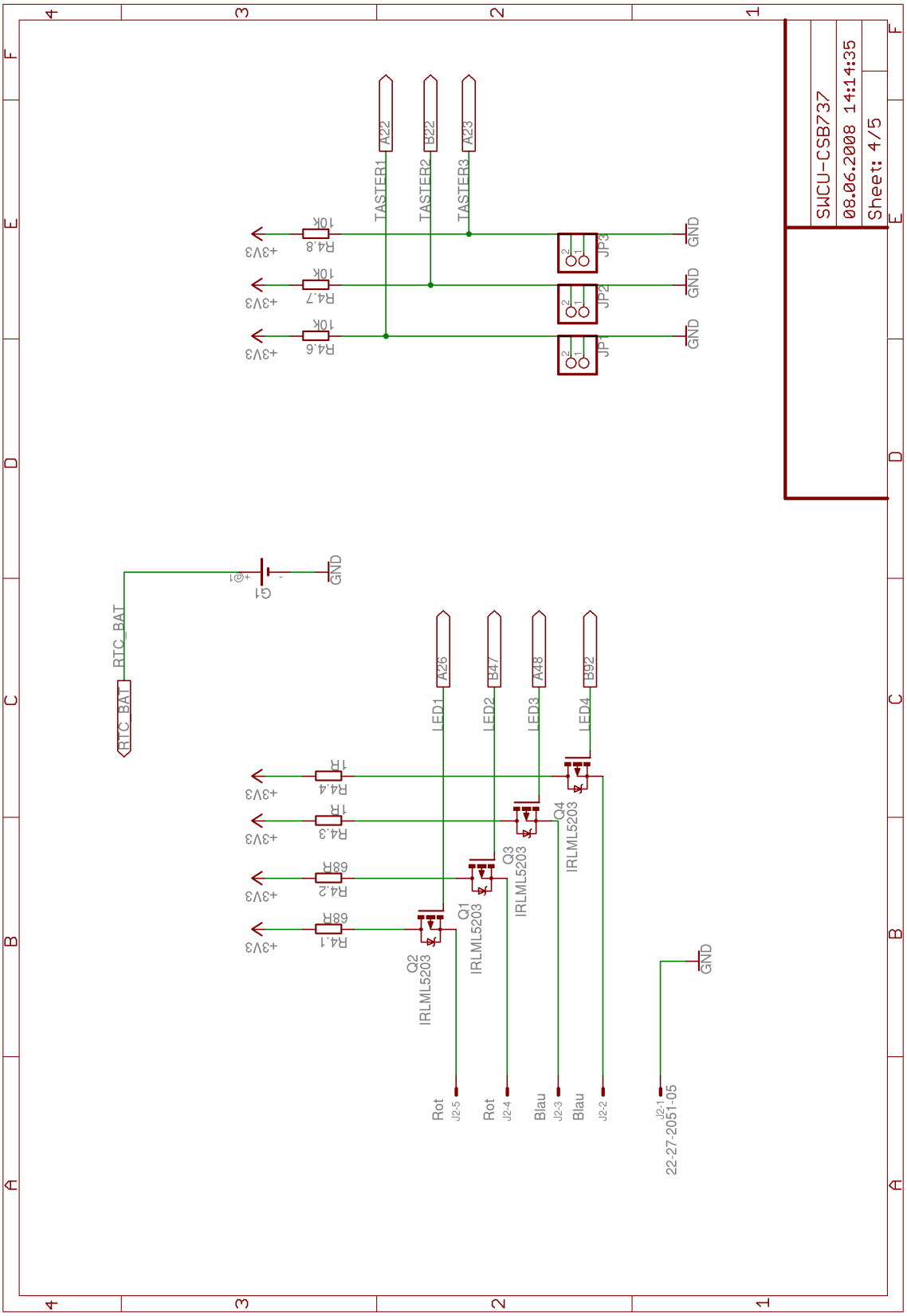

DIPLAY
TITLE: SWCU
Document Number:
REV:
Date: 25.07.2008 10:36:44
Sheet: 6/6
ACHTUNG! falsche Spannung Backgound LEDs
FH23-45S-0.3
J1
+3V3
R11
+12V
GND
IRLML2402
Q3
47µ
L1
10BQ030PBF
D6
MAX782ESA
IC11
C59
100n
C63
100n
C64
33µ
C65
33µ
+3V3
C67
100µ 20V
C68
100n
C66
100n
IC12
MAX766ESA
V+
V+
SHDN
FB
REF
GND
OUT
LX
10BQ030PBF
D7
47µ
L2
68µ 20V
C69
+5V
5k1
R12
C72
4µ7
R13
4k3
GND

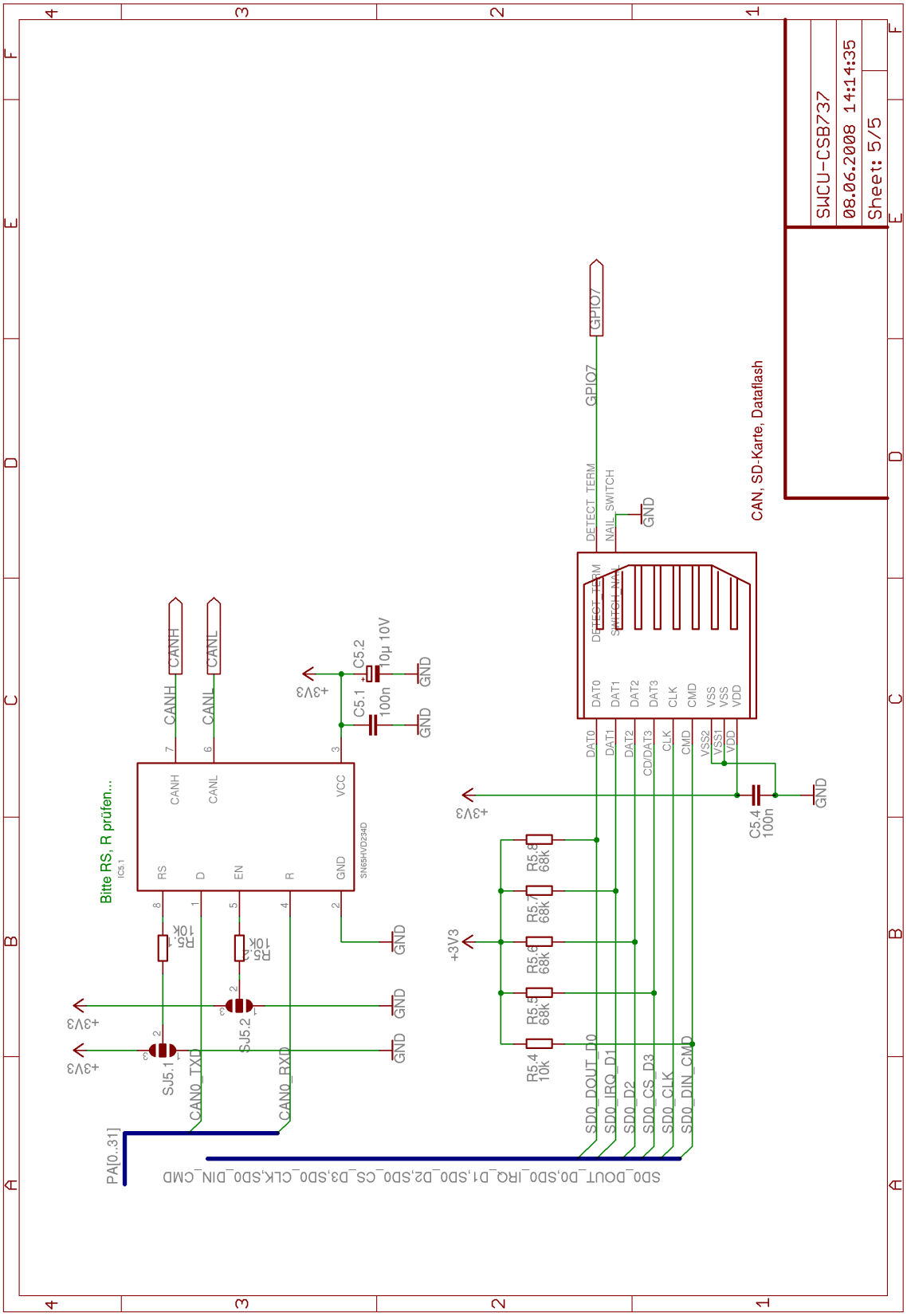
A.2 SWCU-CSB737




SWCU-CSB737
08.06.2008 14:14:35
Sheet: 2/5


SWCU-CSB737
08.06.2008 14:14:35
Sheet: 3/5
ACHTUNG! Falsche Spannung für Backlight LEDs
FH23-45S-0.3
ANODE
CATHODE
VSS
VSS
VSYNC
VGH
VGL
HSYNC
VDH
VSS
D05 MSB
D04
D03
D02
D01
D00 LSB
D15 MSB
D14
D13
D12
D11
D10 LSB
D25 MSB
D24
D23
D22
D21
D20 LSB
DCLK
blau
grün
rot
J1
MAX762ESA
MAX766ESA
10BQ030PBF
IRLML2402
+12V
+15V
-15V
+5V
+3V3
GND


SWCU-CSB737
08.06.2008 14:14:35
Sheet: 4/5
TASTER1
TASTER2
TASTER3
A22
B22
A23
JP3
JP2
JP1
GND
+3V3
10K
RTC_BAT
G1
LED1
LED2
LED3
LED4
A26
B47
A48
B92
Q1
Q2
Q3
Q4
IRLML5203
68R
1R
Rot
Blau
J2-5
J2-4
J2-3
J2-2
J2-1
22-27-2051-05


SWCU-CSB737
08.06.2008 14:14:35
Sheet: 5/5
CAN, SD-Karte, Dataflash
Bitte RS, R prüfen...
IC5.1
SN65HVD234D
CANH
CANL
VCC
RS
D
EN
R
GND
R5.1 10K
R5.2 10K
SJ5.1
SJ5.2
CAN0_TXD
CAN0_RXD
C5.1 100n
C5.2 10µ 10V
C5.4 100n
+3V3
GND
PA[0..31]
SD0_DOUT_D0,SD0_IRQ_D1,SD0_D2,SD0_CS_D3,SD0_CLK,SD0_DIN_CMD
R5.4 10k
R5.5 68k
R5.6 68k
R5.7 68k
R5.8 68k
GPIO7
DETECT_TERM
NAIL_SWITCH
DAT0
DAT1
DAT2
DAT3
CLK
CMD
VSS
VDD
VSS2
CD/DAT3

B Pinbelegung CSB737 / AT91SAM9263

Diese Tabelle stellt dar welche Pinbelegungen sich beim Wechsel von der komplten Eigenentwicklung zum CSB737 basierten Version geändert haben.

CSB737	*AT91SAM9263*	*Funktion*	*Peripheral*
A1	PC31	DBG_TXD	A
B1	PC30	DBG_RXD	A
A10	PA13	CAN_TX	A
B10	PA14	CAN_RX	A
A22	PD04	Taster 1	
B22	PA15	Taster 2	
A23	PB17	Taster 3	
A33	PA08	SD_D0	A
B33	PA09	SD_D1	A
A34	PA10	SD_D2	A
B34	PA11	SD_D3	A
A35	PA07	SD_CMD	A
B35	PA06	SD_CLK	A
A26	PB27	LED 1	B (PWM)
B47	PB08	LED 2	B (PWM)
B48	PB07	LED 3	B (PWM)
B92	PB29	LED 4	B (PWM)

Das Display ist jetzt über den 16bit-Bus der Peripheral B statt den 24bit-Bus der Peripheral A angeschlossen.

CSB737	*AT91SAM9263*	*Funktion*	*Peripheral*
B81	GND	LCD_B0	
A82	PC04	LCD_B1	B
B82	PC05	LCD_B2	B
A83	PC06	LCD_B3	B
B83	PC07	LCD_B4	B
A84	PC08	LCD_B5	B
B84	PC09	LCD_G0	B
A85	PC10	LCD_G1	B
B85	PC11	LCD_G2	B
A86	PC12	LCD_G3	B
B86	PC13	LCD_G4	B
A87	PC14	LCD_G5	B
B87	GND	LCD_R0	
A88	PC15	LCD_R1	B
B88	PC16	LCD_R2	B
A89	PC17	LCD_R3	B
B89	PC18	LCD_R4	B
A90	PC19	LCD_R5	B

Glossar

BGA Beim **B**all **G**rid **A**rray werden die Kontakte zur Platine durch feine Kügelchen auf der Unterseite des IC's hergestellt. Diese Kügelchen sind in einer Gitterstruktur angeordnet.

CAD Beim **C**omputer **A**ided **D**esign (Computer gestützten Kontruktion) werden die Baupläntе für Teile am Computer erstellt.

Differenzsignal Bei einem Differenzsignal wird das Signal auf zwei Leitungen invertiert angelegt. Durch dieses Verfahren lassen sich Störungen minimieren da die Differenz der Signale gleich bleibt wenn bei Leitungen gleichmäßig gestört werden.

EBI **E**xtended **B**us **I**nterface ist ein Bus zum Anbinden von externen Speicher.

Git Versionskontrollsystem welches für den Linux-Kernel entwickelt wurde. Mitlerweile wird es auch von vielen anderen Projekten verwendet.

GPIO **G**eneral **P**urpose **I**n/**O**ut sind Pins die sich sowohl als Input als auch als Output umschallten lassen und für beliebige Aufgaben zu verwenden sind.

MMU Von der **M**emory **M**anagement **U**nit werden die virtuellen Adressen des Mikrocontrollers in die physikalischen Adressen des Speichers übersetzt.

Shell Komandozeile

SoC **S**ystem **o**n **C**hip Zusammenfassung von Mikroprozessor, Speicher und Perepherie zu einem Mikrocontroller.

Via Durchkontaktierung einer Platine. Hierbei werden die Leiterbahnen unterschiedlicher Layer miteinander Verbunden.

Index

Tabellenverzeichnis

Abbildungsverzeichnis

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 21. August 2008	
Ort, Datum	Unterschrift