Ingeniermässige Arbeit: Implementation eines CAN-Librarys für den c515c Microcontroller

Daniel Schramm

Wintersemester 2002 / 2003

Inhaltsverzeichnis

1 Einleitung **3**

1.1 Aufgabenstellung 4

1.1.1 Entwicklung einer CAN-Library für den Infineon Microcontroller c515c unter Verwendung des Keil C-Compilers. 4

1.1.2 Entwicklung einer Beispielapplikation unter Verwendung des Microcontrollerboards des FB3 und dem Echtzeitbetriebssystem RTX-Tiny von Keil. 4

1.2 Hardware 4

1.2.1 Experimentierboards 5

1.2.2 CAN-Dongles 6

1.3 Debugging-Werkzeuge 10

2 Projektstufen **13**

2.1 Entwicklung einfacher Sendefunktion 13

2.2 Einrichtung einer Debugging-Umgebung 13

2.3 Entwicklung der Library 18

2.4 Kommunikation zwischen PC und SysTec-Board 18

2.5 Beispielanwendung für das FB3-Board 18

3 CAN-API **20**

3.1 Funktionsgruppen 20

3.2 Beschreibung der Funktionen 21

3.2.1 Beschreibung der Funktionen zum Einstellen 21

3.2.2 Beschreibung der Funktionen zum Senden 22

3.2.3 Beschreibung der Funktionen zum Setzen bzw. Vorbereiten 23

3.2.4 Beschreibung der Funktionen zum Empfangen 24

3.2.5 Funktion zum Interrupt-Handling 25

3.2.6 Funktion zur Fehlererkennung 25

4 Anwendung **28**

4.1 Kommunikation SysTec-Board mit PC. 28

4.2 Kommunikation zwischen 2 FB3-Boards 35

5 Zusammenfassung und Ausblick **38**

5.1 Zusammenfassung 38

5.2 Ausblick 39

6 Anhang **40**

6.1 Quellcode canlib.h 40

6.2 Quellcode canlib.c 42

6.3 Anmerkungen zum Quellcode 51

7 Referenzen / Literatur **52**

Kapitel 1

Einleitung

Im Rahmen der Vorlesung Hard- und Softwareentwicklung wurden sowohl der 8Bit Microcontroller c515c von Infineon behandelt, als auch der CAN-Bus.

In den Praktika wurden verschiedene Versuche mit dem Microcontroller durchgeführt, wobei zur Programmierung der Keil C-Compiler und das Echtzeitbetriebssystem RTX-Tiny eingesetzt wurde.

Da das RTX-Tiny keine Routinen für den im c515c enthaltenen CAN-Controller enthält, entstand die Idee, diese Funktionen im Rahmen einer Projektarbeit in diese Umgebug zu integrieren.

1.1 Aufgabenstellung

1.1.1 Entwicklung einer CAN-Library für den Infineon Microcontroller c515c unter Verwendung des Keil C-Compilers.

1.1.2 Entwicklung einer Beispielapplikation unter Verwendung des Microcontrollerboards des FB3 und dem Echtzeitbetriebssystem RTX-Tiny von Keil.

1.2 Hardware

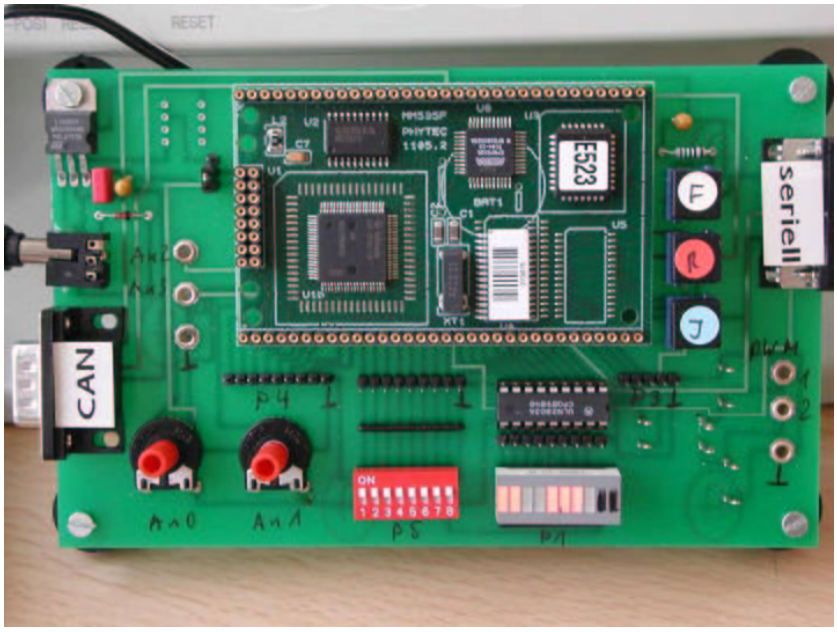
In dem c515c Microcontroller von Infineon, der an der FH-Dortmund in verschiedenen Praktika eingesetzt wird, ist ein Full-CAN-Controller integriert. Dieser Controllertyp enthält 15 parallele Nachrichten-Puffer, von denen 14 eine eindeutige CAN-ID zugewiesen bekommen, unter der sie Nachrichten senden, bzw. für die sie Nachrichten empfangen. Der letzte Puffer verfügt über einen Empfangsfilter, so das Nachrichten mit unterschiedlichen IDs empfangen werden können.

Der CAN-Controller kommuniziert über einen 256 Byte langen Speicherbereich mit der CPU, wobei je der Speicher wie folgt aufgeteilt ist. Die ersten 16 Byte sind Register zur Konfiguration des Controllers, danach folgen je Nachrichten-Objekt 16 Byte, von denen 8 Byte zur Aufnahme der Nutzdaten vorgesehen sind, die restlichen 8 Byte dienen zur Parametrisierung der Nachricht.

In diesem Abschnitt werden zuerst zwei Experimentierboards vorgestellt, die auf dem c515c basieren. Nachfolgend wird die Hard- und Software beschrieben, die zum Analysieren der übertragenen CAN-Pakete und zum Generieren von Referenzpaketen verwendet wurde

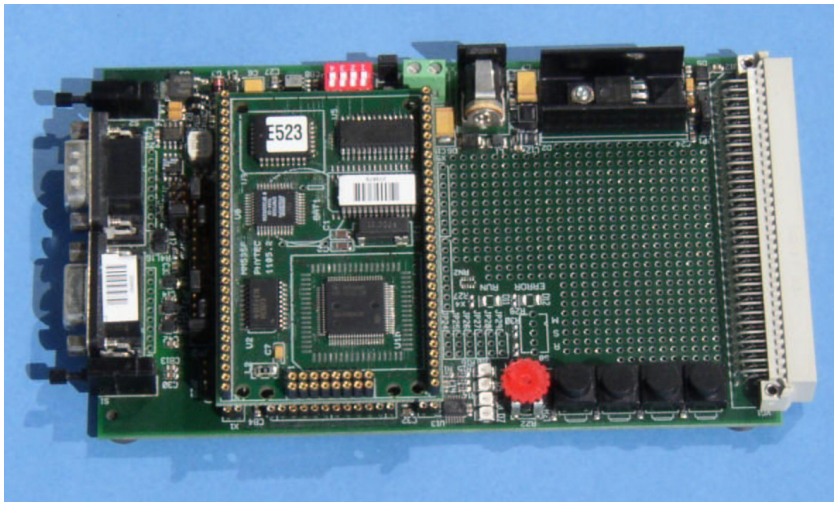
1.2.1 Experimentierboards

Im Rahmen der Arbeit werden zwei Experimentierboards verwendet, die mit dem gleichen Prozessormodul bestückt sind. Die Implementation der Library sollte primär für das nachfolgend vorgestellte FB3-Board geschehen, da dieses jedoch während der Arbeit nur teilweise zur Verfügung stand, wurde ebenfalls auf das SysTec-Board zurückgegriffen.



Auf dieser Abbildung ist das FB3 Board zu sehen. Hierbei handelt es sich um ein Microcontroller-Experimentierboard, das nit einem Pytec Microcontrollermodul bestückt ist. Es verfügt außerdem über Peripherie wie, LED-Anzeigen, DIP-Schalter, Potentiometer, serieller Schnittstelle und CAN-Interface.

Zu beachten ist bei diesem Board, das der CAN-Anschluß nicht standardkonform beschaltet ist.



Das SysTec Board in dieser Abbildung, basiert auf dem gleichen Pytec-Modul, wie das FB3-Board, verfügt allerdings über weniger Peripherie.

1.2.2 CAN-Dongles

Die CAN-Dongles dienen zum Anschluß eines PCs an einen CAN-Bus. Mit passender Software kann der Busverkehr mitgeschnitten werden. Dadurch ist es möglich, vom Microcontroller-Board gesendete Pakete zu kontrollieren. Für einige dieser Dongles gibt es neben der Diagnose-Software noch Betriebssystem-Schnittstellen für Windows und Linux, wodurch der PC mit eigener Software als vollwertiger Busteilnehmer arbeiten kann, das heißt Steuer- und Überwachungsaufgaben übernehmen kann.

Alle diese Interfaces verwenden als Kommunikationsschnittstelle zum PC ein bidirektionales Parallelport-Interface, was zu Kommunikatonsproblemen führen kann. Nicht jeder Dongle ist an jedem PC lauffähig, bzw. je nach PC nur im ECP- oder EPP-Betriebsmodus. Diese Probleme sind auf die geringfügigen Unterschiede in der Implementierung moderner Parallelport-Schnittstellen zu suchen.

Peak-CAN-Dongle

Der Peak-CAN-Dongle zum Anschluß an den Parallelport eines PCs basiert auf dem Basic-CAN-Controller PCA82C200 von Philips. Dieses Interface alleine reichte zum Debugging der Library nicht aus, da der Controller nur 11Bit Identifier unterstützt.

Software ist lediglich für DOS erhältlich.

can200

Hierbei handelt es sich um ein von Martin Homuth-Rosemann entwickeltes Interface, das nur wenige Bauteile umfasst. Es basiert auf dem Philips SJA1000 PeliCan-Controller.

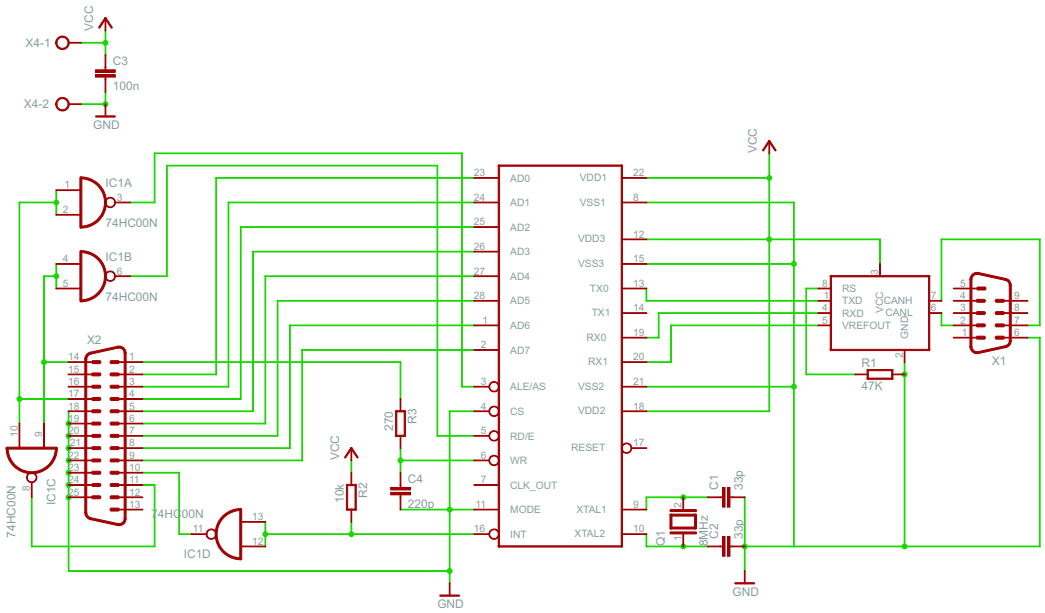
Dieser Controller ist Pin-Kompatibel zu dem PCA82C200, unterstützt jedoch zusätzlich CAN 2.0B und somit 29Bit Identifier. Ein PeliCan-Controller ist ein um einen 16 Byte langen FIFO erweiterter Basic-Can-Controller.

An diesem Interface fehlt eine Schutzschaltungen zwischen Parallelport und Controller, so dass unter Umständen die CMOS-Eingänge des CAN-Controllers

offen liegen. Offene CMOS-Eingaenge neigen generell zu Schwingungen einzufangen, so dass es leicht zu Zerstoerung des Bauteils kommen kann.

Dieses Interface hat eine sehr gute Softwareunterstützung, sowohl unter Windows, als auch unter Linux.

Der Schaltplan zu dem Interface ist auf der Webseite des Entwicklers veröffentlicht. [1]



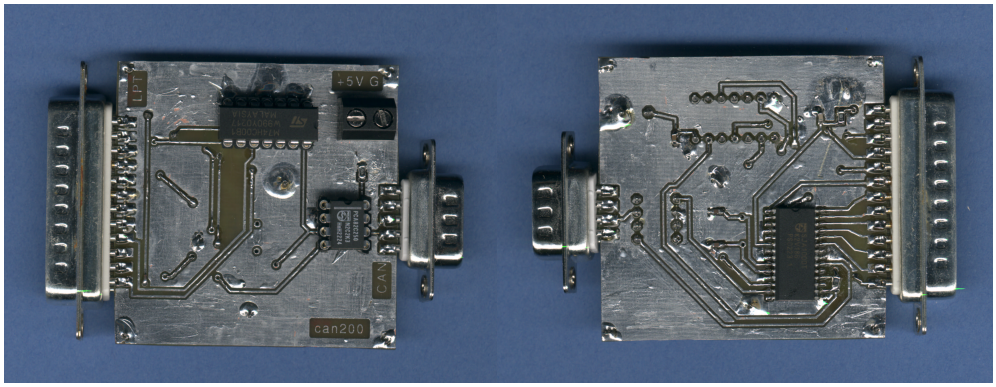
In dem Schaltplan ist rechts neben dem SJA1000 CAN-Controller noch der PCA82C250 zu erkennen, hierbei handelt es sich um einen CAN-Transceiver-Baustein. Dieser wandelt die TTL-Pegel vom CAN-Controller in differentielle Signale um, die auf den CAN-Bus eingespeist werden können.

Aus dem gezeigten Schaltplan wurde unter Verwendung der CAD-Software EAGLE ein Platinenlayout erstellt.

Basierend auf diesem Layout wurde mit Fotolitografie und Nassätzen eine Prototyp-Platine erstellt. Diese wurde anschließend gebohrt und bestückt.

Die fertige Schaltung ist auf der folgenden Abbildung gezeigt.

[1] http://private.addcom.de/horo/can200/

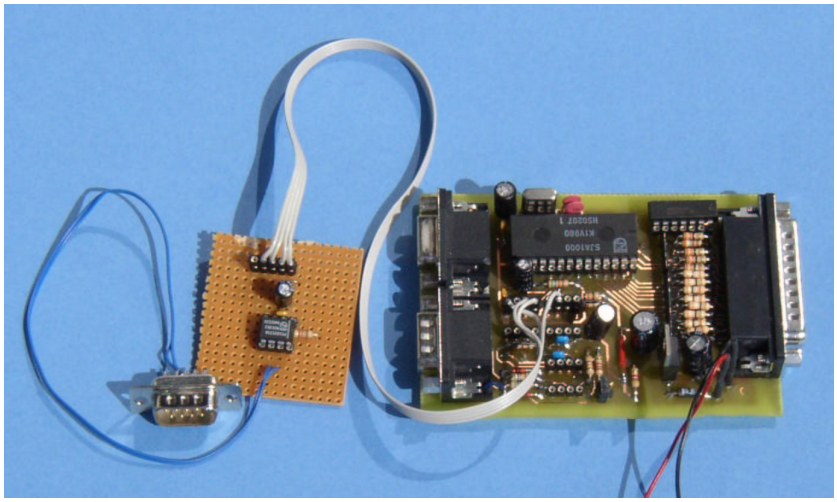


Elektor-Can-Interface

Ein weiteres auf dem SJA1000 basierendes CAN-Interface ist in der Zeitschrift Elektor 6/2000 vorgestellt. Da es auf dem gleichen Controller basiert, kann die gleiche Software zum Ansteuern verwendet werden. Das Interface enthält im Gegensatz zu dem CAN200 allerdings Schutzschaltungen, wie eine galvanische Trennung der CAN-Schnittstelle und eine Terminierung des Datenbusses am Anschluß zum PC-Druckerpoprt.

Der Betrieb dieses Interfaces ist nicht unproblematisch. Zum Teil sind diese Probleme auf Designfehler zurückzuführen, wie z.B. falsch dimensionierte Arbeitspunkteinstellung des CAN-Tranceivers, die nicht der entspricht, die im Datenplatt des Herstellers empfohlen wird. Aus diesem Grund wurde, wie auf dem folgenden Bild zu sehen ist, der CAN-Tranceiver auf eine Lochrasterplatine ausgelagert und entsprechend der von Phillips empfohlenenen Peripherie verschaltet,

Die Kommunikation zwischen PC und dem SJA1000 ist ebenfalls problematisch, dies ist durch unterschiedliche interne Beschaltung der Parallelports in PCs bedingt. Diese Probleme wurden von Jürgen Eder, dem Entwickler der Treibersoftware, bestätigt. Die von ihm vorgeschlagene Änderung an der Terminierung führt zwar dazu, das das Interface sporadisch erkannt wird, aber eine zuverlässige Kommunikation zwischen PC und CAN-Controller ist nicht gegeben.



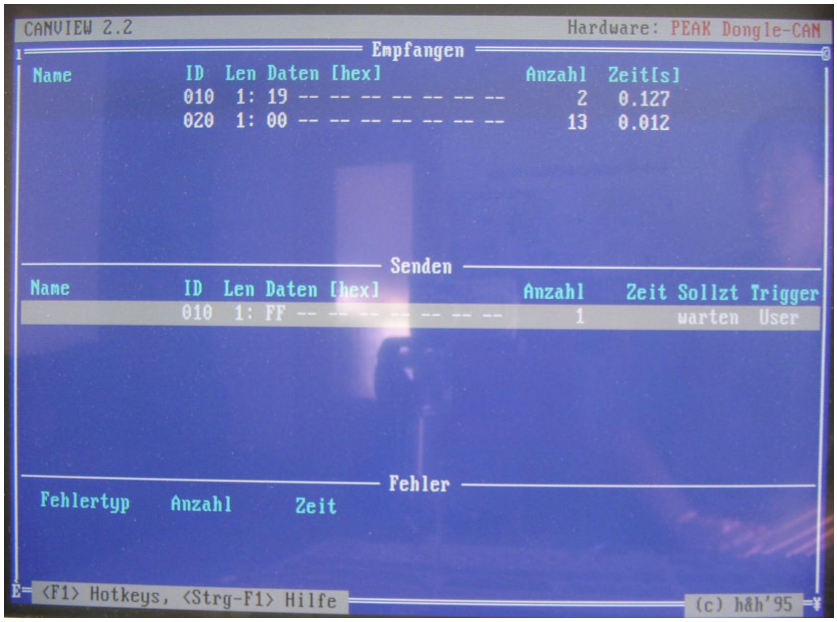
1.3 Debugging-Werkzeuge

Zu den verschiedenen CAN-Dongles existieren jeweils Softwarepakete, mit denen es möglich ist, übertragene Datenpakete auf dem CAN-Bus zu beobachten, beziehungsweise selbst Referenzpakete zu generieren. Damit ist es möglich zu kontrollieren, ob die Funktionen der CAN-Library funktionieren.

CanView

CanView ist die zum PEAK-Can-Donkle passende DOS-Software. Sie zeigt in zwei Fenstern empfangene und gesendete Pakete an, in einem weiteren aufgetretene Übertragungsfehler.

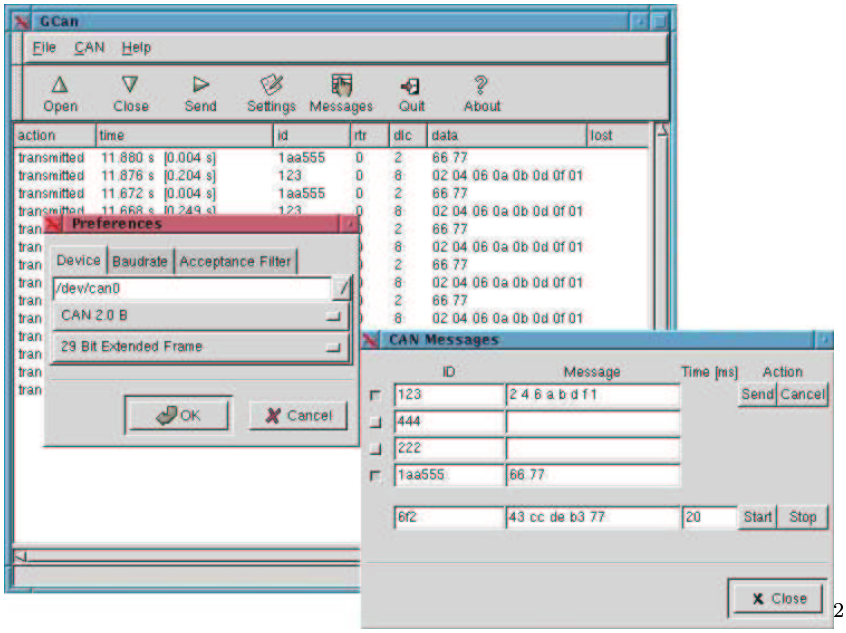
Es kann eine Eingabemaske aufgerufen werden, um zu sendende Pakete zu parametrisieren.



GCan

Bei GCan handelt es sich um eine Linux-Software, die auf eine Betriebssystem-Schnittstelle aufsetzt, für die es unter anderem Treiber für das CAN200 und das Elektor-Interface gibt. Sie erfüllt die gleichen Aufgaben, wie CanView.

Für die sehr einfach strukturierte Betriebssytem-Schnittstelle können auch effizient eigene Testapplikationen geschrieben werden, wie z.B. die in Kapitel 4.1 gezeigte Perl-Anwendung.



[2]

[2]Bild: Quelle: http://canbus4linux.sourceforge.net/

Kapitel 2

Projektstufen

Im folgenden wird der Ablauf der Arbeit geschildert. Es wird insbesondere auf den Einsatz verscheidener Messeinrichtungen zur Fehlersuche eingegangen und dargestellt, wie diese Messmittel zum Teil vorher erstellt wuden.

2.1 Entwicklung einfacher Sendefunktion

Zuerst wurden einfache Sendefunktionen für den CAN-Controller implementiert, mit denen später die Debugging-Umgebung kontrolliert wird.

Da zu diesem Zeitpunkt noch kein Referenzempfänger zur Verfügung stand, von dem bekannt war, das er fehlerfrei arbeitet, kommt ein Oszilloskop zum Einsatz, um die ersten ausgesendeten Pakete zu überprufen. Nachdem der Controller fehlerfreie CAN-Pakete mit einer Baudrate von 125kBit senden kann, wird mit der Einrichtung und den Tests der Debugging-Umgebung begonnen.

2.2 Einrichtung einer Debugging-Umgebung

Nachdem es nicht auf Anhieb möglich war, den von der Fachhochschule bereitgestellte Peak-CAN-Dongle an einem aktuellen PC mit der mitgelieferten

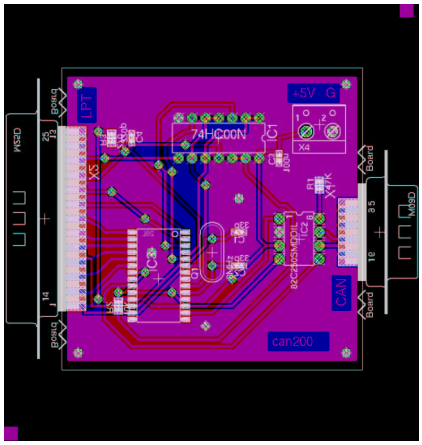
DOS-Software in Betrieb zu nehmen, muss nach Alternativen gesucht werden, da bereits absehbar ist, daß der Peak-Dongle aufgrund seines veralteten Controllers nicht ausreichen wird, um die CAN 2.0-Funktionalität des c515c zu testen.

Als Alternativen bieten sich 2 weitere CAN-Dongle an, von denen die Schaltpläne und Software leicht verfügbar sind.

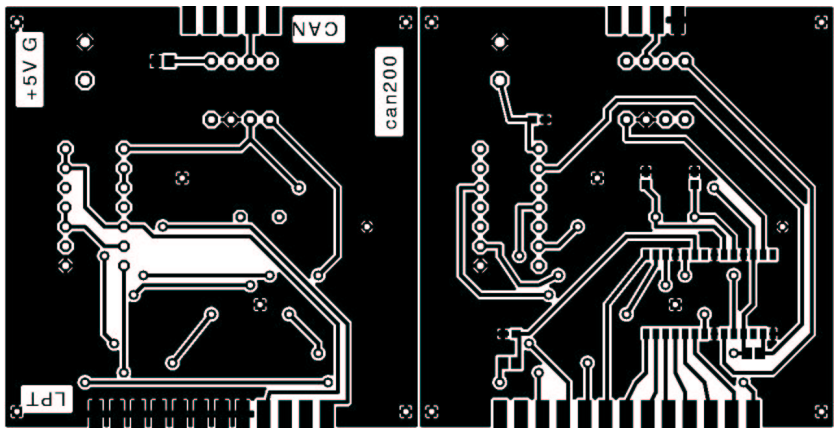
Dies sind das CAN200- und das Elektor-Interface, beide basieren auf einem ähnlichen Konzept und verwenden den gleichen CAN-Controller.

Für das CAN200-Interface stand lediglich der Schltplan zur Verfügung, so das hier zuerst ein Platinenlayout mit dem CAD-Programm Eagle, erstellt werden muss, um einen Prototypen aufzubauen.

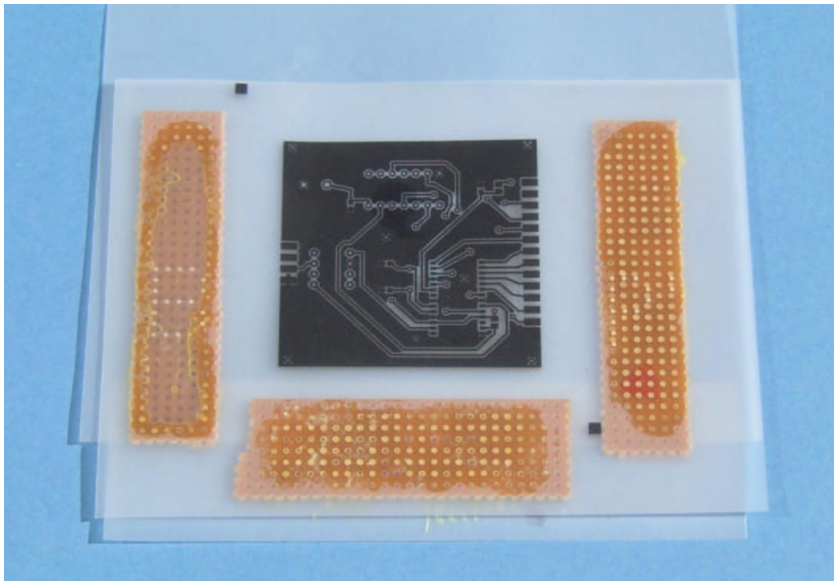
Die folgende Abbildung zeigt das gesamte Layout in der Arbeitsansicht von Eagle, wobei der obere Layer in rot, der untere in blau dargestellt wird. Bauelemente werden weiß angezeigt und Durchkontaktierungen in grün.



Aus der obigen Arbeitsansicht werden 2 monochrome Abbildungen genereiert, in denen lediglich noch die späteren Kupferflächen der jeweiligen Ebenen zu sehen sind. Bei der Layout-Erstellung wird darauf geachtet, das möglichst große Kupferflächen stehen bleiben, die entweder mit Masse oder der Versorgungsspannung verbunden sind. Dies hat neben dem Umweltaspekt, das Ätzmittel gespart wird, noch EMV-Vorteile, da das Übersprechen zwischen Signalleitungen minimiert wird.



Wie in den Abbildungen zu sehen, handelt es sich um ein doppelseitiges Layout, das für eine gemischte Bestückung von SMD- und konventionellen Bauelementen ausgelegt ist. Diese gemischte Bestückung ist durch die Lieferbarkeit der Bauformen bedingt. Sofern beide Varianten verfügbar waren, wurde SMD bevorzugt, da SMD-Bauteile weniger anfällig gegenüber EMV sind und die Zahl der Bohrungen niedriger ist. Das fertige Layout wird anschließend mit einem Laserdrucker auf Folie zur Druckformerstellung übertragen. Aus dieser Folie wird, wie auf der nächsten Abbildung zu sehen, eine Tasche geformt, indem die beiden Lagen mit Platinenresten als Abstandhalter so verklebt wurden, dass später das fotobeschichtete Basismaterial dazwischen eingeschoben werden kann.



Auf diese Weise ist es möglich, mit einem einseitigen Belichtungsgerät beide Lagen nacheinander deckungsgleich zu belichten.

Bei dem verwendeten Belichtungsgerät handelt es sich um einen Eigenbau, der aus 4 UV-Leutstoffröhren mit Vorschaltgeräten in einem Stahlblech-Gehäuse besteht. Um eine konstante Belichtungszeit zu erreichen verfügt das Gerät über einen Belichtungstimer, der ebenfalls eine Eigenentwicklung[1] ist und auf einem Atmel AVR-Microcontroller basiert.

Nachdem die Platine von beiden Seiten nacheinander je 8 Minuten belichtet ist, muß sie in einer Lösung von 10g NaOH auf 0,5l ca. 25 Grad warmen Wasser entwickelt werden. Diese Angaben beziehen sich auf das verwendete Basismaterial von Bungard, andere Materialien benötigen andere Konzentrationen.

Die Entwicklung dauert etwa 20-40s, je nach Temperatur des Entwicklers.

Nach dem Entwickeln muß die Platine unter Wasser gereinigt werden und kann anschließend geätzt werden. Hierzu wird ein Ätzbad aus einer wässrigen Fe_3Cl-Lösung verwendet.

[1] Dokumentation des Belichtungstimers: http://www.fset.de/~daniel/timer.html

Nach dem Ätzen wird die Platine gründlich mit Wasser gereinigt. Nach dem Trocknen wird der Rest der Fotobeschichtung mit Spiritus entfernt und die Platine anschließend gebohrt. Nun kann die Platine bestückt werden, wobei die Durchkontaktierung mit dünnem Draht erfolgt.

Auf die gleiche Weise wird auch die Platine für das Elektor-Interface erstellt, wobei hier jedoch auf die Layout-Erstellung verzichtet werden kann, da der Elektor-Verlag dieses in digitaler Form auf seiner Webseite[2] bereitstellt.

Bei der Inbetriebnahme der Interfaces stellt sich heraus, dass die Interfaces nicht an jedem PC gleich gut erkannt werden, so dass mehrere PCs und Mainboards, sowie Parallelport-Karten ausprobiert wurden, um die Interfaces anzusprechen. Dabei gelang es schließlich den CAN200- und den Peak-Dongle zuverlässig zu betreiben.

Somit stand nach der Software-Installation eine gute Testumgebung zur Verfügung.

Die Inbetriebnahme des Elektor-Interfaces gestaltet sich schwieriger. Mit der vom Verlag beschriebenen Variante ist es nicht möglich, CAN-Pakete zu empfangen. Wie sich bei Messungen herausstellt, kann der CAN-Tranceiver-Baustein die empfangenen Pakete nicht in CMOS-Pegel umsetzen. Eine Rechersche im Datenblatt des Tranceiverbausteins ergab, dass ein Widerstand für die Betriebsart-Einstellung einen Wert hat, der in einem undefinierten Bereich liegt. Mit einem Neuaufbau des Tranceivermoduls auf einer expernen Platine kann dieses Problem behoben werden. Ein weiteres Problem mit dieser Hardware, besteht darin, dass die Kommunikation mit dem PC sehr unzuverlässig ist. Das führt dazu, dass das Modul nicht immer von dem Treiber erkannt wird. Durch eine Änderung an der Terminierung des Datenbuses kann eine leichte Verbesserung erreicht werden.

Insgesamt zeigt sich, daß das CAN200 Modul an den zur Verfügung stehenden PCs zuverlässiger läuft.

[2] http://www.elektor.de/

2.3 Entwicklung der Library

Nachdem eine Testumgebung zur Verfügung steht, mit der empfangene Pakete analysiert und Referenzpakete gesendet werden können, wird mit der Implementierung der Kommunikationsfunktionen begonnen, anschließend mit den Funktionen zum Interrupt-Handling, Fehlerbehandlung und zur Konfiguration des CAN-Controllers fortgefahren. Alle Funktionen wurden in diesem Zusammenhang einzeln gestestet, wobei jeweils kleine Testprogramme erstellt werden, die Datenpakete mit der Referenz austauschten. Als Referenz kommt hierzu das CAN200-Modul mit der Software GCan zum Einsatz.

2.4 Kommunikation zwischen PC und SysTec-Board

Nach Fertigstellung der Library, wird eine erste Testanwendung entwickelt, bei der das SysTec-Microcontroller-Board über den CAN-Bus mit einem Linux-PC kommuniziert, auf dem gleichzeitig der CAN-Monitor GCan und eine selbstgeschriebene Testanwendung lief.

Als Programmiersprache kommt für die PC-Anwendung die Scriptsprache Perl mit dem Grafikmodul TK zum Einsatz.

Die Anwendung wird in Kapitel 4.1 beschrieben.

2.5 Beispielanwendung für das FB3-Board

Nachdem die Library erfolgreich in einer ersten Anwendung getestet ist und die FB3-Boards in den Semesterferien exclusiv zur Verfügung stehen, kann mit der Entwicklung einer Anwendung für diese Module begonnen werden. Hierbei zeigt sich zuerst, dass eine Kommunikation mit einem bereits vorher getesteten Programm nicht möglich ist, was auf ein Hardwareproblem schließen lässt. Bei genauer Betrachtung zeigt sich, dass der CAN-Anschluss auf

diesen Boards nicht standardkonform beschaltet ist. Über eigens angefertigte Adapterkabel werden 2 dieser Experimentierboards an den bestehenden Bus angepasst.

Anschließend wird eine Beispielanwendung in C entwickelt, die in Kapitel 4.2 vorgestellt wird.

Kapitel 3

CAN-API

3.1 Funktionsgruppen

Im folgenden Kapitel werden die verschiedenen Funktionen der Library vorgestellt. Diese sind in folgende Kategorien unterteilt.

Einstellungen

Mit dieser Gruppe können Einstellungen am CAN-Interface vorgenommen werden, sowie die einzelnen Message-Objekte des Can-Controllers definiert werden.

Die Funktionsnamen beginnen mit can_define_*().

Jedes Nachrichten-Objekt muß zuerst definiert werden, bevor eine der anderen Funktionen damit arbeiten kann.

Senden von Nachrichten

Mit den Funktionen can_send_*() können Nachrichten direkt verschickt werden.

Festlegen von Nachrichten

Die Funktionen can_set_*() füllen Nachrichten-Objekte mit einem Inhalt, der zu einem anderen Zeitpunkt mit can_trigger() versendet werden kann, bzw. von einem anderen CAN-Busteilnehmer angefordert werden kann.

Empfangen

can_poll_*()-Funktionen überprüfen, ob eine Nachricht empfangen wurde und lesen diese aus.

Interrupt

Fehlerbehandlung

3.2 Beschreibung der Funktionen

3.2.1 Beschreibung der Funktionen zum Einstellen

In den folgenden Funktionen wird der Parameter "char obj_nr" zur Auswahl des Nachrichten-Objektes verwendet.

void can_init(int baudrate);

Initialisieren des CAN-Controllers und setzen der Baudrate.

Mögliche Baudraten sind: 10, 20, 100, 125, 250, 500 und 1000 kBit/s

void can_define_msg(char obj_nr, char type, short id);

Definieren einer 11Bit CAN 1.1 oder CAN 2.0(A) Nachricht.

Übergeben wird die Objekt-Nr. 0-15, der Typ TRANSMIT oder RECEIVE, sowie die CAN-ID der Nachricht.

Bei dem Objekt 15 entspricht die CAN-ID dem Eingangsfilter.

void can_define_extmsg(char obj_nr, char type, long extid);

Definieren einer 27Bit CAN 2.0(B) Nachricht.

void can_define_irq(bit IE, bit SIE, bit EIE);

Setzen der Interrupt-Einstellungen des CAN-Controllers.

void can_define_txi(char obj_nr, bit enable);

Aktivieren oder Deaktivieren des Sende-Interrupts für eine bestimmte Nachricht.

Übergabewerte sind die Nachrichten-Nummer als erster Parameter.

Als 2. Parameter eine 1 zum aktivieren, eine 0 zum deaktivieren.

void can_define_rxi(char obj_nr, bit enable);

Aktivieren oder Deaktivieren des Empfangs-Interrupts für eine bestimmte Nachricht.

Übergabewerte sind die Nachrichten-Nummer als erster Parameter.

Als 2. Parameter eine 1 zum aktivieren, eine 0 zum deaktivieren.

3.2.2 Beschreibung der Funktionen zum Senden

void can_send_pulse(char obj_nr);

Absenden einer leeren Nachricht mit der für das Nachrichten-Objekt (obj_nr) definierten ID.

void can_send_byte(char obj_nr, char db);

Senden eines Byetes.

void can_send_ptr(char obj_nr, const void* tdata, char n);

Senden eines beliebigen Datentyps, der nicht mehr als 8 Bytes lang ist.

Dazu wird neben der Nachrichten-Nummer ein Zeiger auf die Daten, sowie die Länge der zu übertragenen Daten übergeben.

Sollten die Daten mehr als 8 Byte lang sein, werden nur die ersten 8 Byte übertragen und die restlichen ignoriert.

Beispiel zum Versenden eines Integers:

int i=1;

can_send_ptr(1,&i,sizeof(int));

Beispiel zum Versenden eines Strings:

char str[]="test";

can_send_ptr(1,str,strlen(str));

3.2.3 Beschreibung der Funktionen zum Setzen bzw. Vorbereiten

void can_set_byte(char obj_nr, char db);

Ablegen eines Bytes im Datenregister eines Nachrichtenobjektes.

void can_set_ptr(char obj_nr, const void* tdata, char n);

Ablegen eines beliebigen Datentypes in einem Nachrichtenobjekt.

void can_trigger(char obj_nr);

Absenden eines Nachrichtenobjektes. Diese Funktion kann verwendet werden um ein mit can_set_* vorbereiteten Nachrichtenobjet abzusenden, oder um ein bereits mit can_send_* übertragenes Paket zu wiederholen.

3.2.4 Beschreibung der Funktionen zum Empfangen

int can_poll_pulse(char obj_nr);

Nachsehen, ob bei dem angegebenen Objekt eine Nachricht angekommen ist.

Rückgabewerte:

0 Nachricht empfangen

-1 Keine Nachricht vorhanden.

int can_poll_byte(char obj_nr, char *db);

Empfangen eines Bytes. Übergeben werden muß hierzu ein Zeiger auf das zu schreibende Byte. (Cann by Reference)

Rückgabewert:

0 Nachricht empfangen

-1 keine Nachricht vorhanden.

int can_poll_ptr(char obj_nr, void *rdata);

Rückgabewert ist die länge der empfangenen Daten.

Diese Information kann z.B. auch zur Fehlererkennung eingesetzt werden:

int i;

if (can_poll_ptr(1,&i) != sizeof(int)) {

// Fehler: Empfangene Daten passen von der Länge her nicht zu einem Integer

}

Rückgabewert:

-1 kein Nachricht vorhanden

>=0 Länge der ampfangenen Daten.

int can_poll_string(char obj_nr, char *rdata);

Empfangen eines Strings.

Die empfangenen Daten werden automatisch mit '\0' abgeschlossen.

Zurückgegeben wird die Stringlänge oder -1, wenn keine Nachricht vorliegt.

void can_remote(char obj_nr);

Absenden eines Remote-Frames, um Daten anzufordern.

Hierzu muß die Funktion auf ein "RECEIVE"-Objekt angewendet werden.

void can_wait(char obj_nr);

Warten auf die Ankunft eines Objektes.

3.2.5 Funktion zum Interrupt-Handling

int can_handle_irq(void);

Auslesen des Interrupt-Registers.

Rückgabewerte:

0 : no interrupt pending

1 : status change

2 : Message 15

(2+N): Message N

3.2.6 Funktion zur Fehlererkennung

Diese Funktionen konnten nicht getestet werden, da keine Datenquelle zur Verfügung stand, die geziehlt Fehlerhafte Datenpakete generieren kann.

int can_check_lec(void);

Auslesen des letzten Fehlerstatus (LEC-Register)

Die Rückgabewerte sind in canlib.h wie folgt definiert:

#define NO_ERROR 0

#define STUFF_ERROR 1

#define FORM_ERROR 2

#define ACK_ERROR 3

#define BIT1_ERROR 4

#define BIT0_ERROR 5

#define CRC_ERROR 6

bit can_check_txok(void);

Statusregister SR überprüfen, ob der letzte Sendebefehl erfolgreich war.

Dieser Befehl muß in Kombination mit can_clear_txok() eingesetzt werden.

void can_clear_txok(void);

Rücksetzen des TXOK-Bits im Statusregister.

bit can_check_rxok(void);

Statusregister SR überprüfen, ob Pakete fehlerfrei empfangen wurden.

Dieser Befehl muß in Kombination mit can_clear_rxok() eingesetzt werden.

void can_clear_rxok(void);

Rücksetzen des RXOK-Bits im Statusregister.

bit can_check_boff(void);

BOFF-Bit aus dem Statusregister SR ausgeben.

bit can_check_ewrn(void);

EWRN-Bit aus dem Statusregister SR ausgeben.

Kapitel 4

Anwendung

4.1 Kommunikation SysTec-Board mit PC.

Für die erste praktische Anwendung der CAN-Library wurde die folgende Kommunikation zwischen einem PC und einem Microcontroller-Board entwickelt.

Hierbei kommen auf der Microcontroller-Seite ein C-Programm und auf der PC-Seite ein Perl/TK-Programm zum Einsatz.

Die Kommunikation soll den Status von 4 LEDs auf dem Board am PC anzeigen. Eingeschaltet und ausgeschaltet werden können die LEDs sowohl über Taster auf dem Microcontroller-Board, als auch über die PC-Software. Die Anzeige am PC erfolgt über Buttons, die nach einer ueber den Bus empfangenen Statusmeldung entweder gedrueckt, oder erhoeht dargestellt werden.

Die PC-Software stellt ebenfalls einen Analogwert dar, der von dem Microcontrollerboard immer dann ueber den CAN-Bus gesendet wird, wenn sich dieser Wert ändert.

Bildschirmfoto der PC-Software

Quellcode der PC-Software Programmiersprache Perl mit Grafiklibrary TK

```
#!/usr/bin/perl
use Tk;

open(CANDEV, "/dev/can0");

$mw = new MainWindow;
$ANALOG = $mw->Label( -text => "Analog: 0", -font => "Helvetica 30 normal" )->pack


$S1 = $mw->Button( -text => "S1" )->pack(side => 'left');
$S1->bind("<Button-1>",\&S1_callback);
$S2 = $mw->Button( -text => "S2" )->pack(side => 'left');
$S2->bind("<Button-1>",\&S2_callback);
$S3 = $mw->Button( -text => "S3" )->pack(side => 'left');
$S3->bind("<Button-1>",\&S3_callback);
$S4 = $mw->Button( -text => "S4" )->pack(side => 'left');
$S4->bind("<Button-1>",\&S4_callback);
$mw->Label()->pack;

$TICK = $mw->Label( -text => "." )->pack;


$mw -> fileevent(CANDEV,'readable',[\&can_read]);
$mw -> repeat(100,[\&run_tick]);

MainLoop;

close CANDEV;
```

```
29
30 sub run_tick {
31   if ( $tick_counter ) {
32     $TICK -> configure( -text => ".");
33     $tick_counter = 0;
34   } else {
35     $TICK -> configure( -text => " ");
36     $tick_counter = 1;
37   }
38 }
39
40 sub can_read {
41   $line = <CANDEV>;
42 #  print $line;
43   ($type, $time, $id, $rtr, $length, @msg) = split ' ', $line;
44   if ( $id == 5 ) {
45     $value =  hex("$msg[0]$msg[1]");
46     $ANALOG -> configure ( -text => "Analog: $value" );
47   }
48
49   if ( $id == 1 ) {
50     if ( $msg[0] == 1 ) {
51       $S1 -> configure ( -relief => 'sunken' );
52     } else {
53       $S1 -> configure ( -relief => 'raised' );
54     }
55   }
56   if ( $id == 2 ) {
57     if ( $msg[0] == 1 ) {
58       $S2 -> configure ( -relief => 'sunken' );
59     } else {
60       $S2 -> configure ( -relief => 'raised' );
61     }
62   }
63   if ( $id == 3 ) {
64     if ( $msg[0] == 1 ) {
65       $S3 -> configure ( -relief => 'sunken' );
66     } else {
67       $S3 -> configure ( -relief => 'raised' );
```

```
68      }
69    }
70    if ( $id == 4 ) {
71      if ( $msg[0] == 1 ) {
72        $S4 -> configure ( -relief => 'sunken' );
73      } else {
74        $S4 -> configure ( -relief => 'raised' );
75      }
76    }
77  }
78
79  sub S1_callback {
80    open(CANOUT, ">> /dev/can0");
81    print CANOUT "6 0 1\n";
82    close CANOUT;
83    print "S1\n";
84  }
85  sub S2_callback {
86    open(CANOUT, ">> /dev/can0");
87    print CANOUT "6 0 2\n";
88    close CANOUT;
89    print "S2\n";
90  }
91  sub S3_callback {
92    open(CANOUT, ">> /dev/can0");
93    print CANOUT "6 0 3\n";
94    close CANOUT;
95    print "S3\n";
96  }
97  sub S4_callback {
98    open(CANOUT, ">> /dev/can0");
99    print CANOUT "6 0 4\n";
100   close CANOUT;
101   print "S4\n";
102 }
```

Quellcode für das Microcontroller-Board

```
1  //#pragma SRC // generates assembly file (Achtung: keinen Maschinencode)
2  #include <reg515c.h>  // üFr Zugriff auf SFR Register des C515C
```

```
#include <rtx51tny.h> // RTX-51 tiny functions, 10 MHz, intern 20 MHz;
                      // machine cycle= 0.5 us;
                      // tick= 10 000 machine = 5ms; (gemessen 5 ms);
                      // Timeout = 5 ticks= 25 ms
#include <stdio.h>    // üfr Zugriff auf Standard In Out
#include "canlib.h"

// sfr   P6     = 0xDB;
//Deklarationen
// Deklaration der äbinren Ein- äAusgnge
sbit S3 = P4 ^4;
sbit S4 = P4 ^5;
//sbit S5 = P6 ^2;
//sbit S6 = P6 ^1;

//sbit DIP1 = P6 ^6;
//sbit DIP2 = P6 ^5;
//sbit DIP3 = P6 ^4;
//sbit DIP4 = P6 ^3;

sbit D7 = P4 ^0;
sbit D8 = P4 ^1;
sbit D9 = P4 ^2;
sbit D10 = P4 ^3;

#define TASK_INIT      0
#define TASK_CAN_TIME  1
#define TASK_CAN_POLL  2

     // Deklaration der externen Funktionen
extern  int mess_ad_kanal(unsigned char MessKanalNr);
extern void com0_init(void);

void Task_init(void) _task_ TASK_INIT
   {
      D7 = 0;
      D8 = 0;
      D9 = 0;
      D10 = 0;
```

```
42        com0_init();
43
44        can_init(125);
45        can_define_irq(1,0,0);
46
47        can_define_msg(1,TRANSMIT,1);
48        can_define_msg(2,TRANSMIT,2);
49        can_define_msg(3,TRANSMIT,3);
50        can_define_msg(4,TRANSMIT,4);
51        can_define_msg(5,TRANSMIT,5);
52        can_define_msg(6,RECEIVE, 6);
53        can_define_msg(7,TRANSMIT,7);
54
55        os_create_task(TASK_CAN_TIME);      //Starts task
56        os_create_task(TASK_CAN_POLL);      //Starts task
57        while (1)
58           {
59                      // do nothing
60                          os_wait (K_IVL, 170, 0);
61           }
62     }
63
64  void Task_CAN_TIME(void) _task_ TASK_CAN_TIME
65     {
66        char db;
67        int analog, old_analog;
68        unsigned char i=0;
69        unsigned char j=0;
70        double k=100000.0;
71        while (1)
72           {
73                     old_analog=analog;
74                     analog = mess_ad_kanal(7);
75                     if (old_analog != analog) {
76                     printf("%i\n", analog);
77                       can_send_ptr(5,&analog,sizeof(int));
78                     }
79
80                     if (S3) {
```

```
                    D7 = !D7;
                        can_send_byte(1,D7);
                  }
                  if (S4) {
                    D8 = !D8;
                        can_send_byte(2,D8);
                  }
                  if (P6 & 0x04) {
                    D9 = !D9;
                        can_send_byte(3,D9);
                  }
                  if (P6 & 0x02) {
                    D10 = !D10;
                        can_send_byte(4,D10);
                  }
                  can_send_pulse(7);

                  os_wait (K_IVL, 70, 0);

         }

   }

void Task_CAN_POLL(void) _task_ TASK_CAN_POLL
   {
   while (1) {
      char db;
           if (can_poll_byte(6,&db) >=0 ) {
             printf("r: %i\n",(int)db);
             switch (db) {
                           case 1:
                                    D7 = !D7;
                                        can_send_byte(1,D7);
                                        break;
                           case 2:
                                    D8= !D8;
                                        can_send_byte(2,D8);
                                        break;
                           case 3:
```

```
120                                         D9 = !D9;
121                                             can_send_byte(3,D9);
122                                             break;
123                                 case 4:
124                                         D10 = !D10;
125                                             can_send_byte(4,D10);
126                                             break;
127                                 }
128                             }
129
130                              os_wait (K_IVL, 30, 0);
131                 }
132
133              }
134
135
136 void ISR_can (void) interrupt 17 {
137   char db;
138   int returnval;
139   while ( returnval = can_handle_irq() )
140     printf("interrupt: %i\n",returnval);
141         if ( returnval == 8 ) {
142                 // Hier koennte Interrupt-Betrieb stattfinden.
143         }
144 }
```

4.2 Kommunikation zwischen 2 FB3-Boards

Für eine Kommunikation zwischen den beiden FB3-Boards, wurde folgende Anwendung entwickelt.

Die Setllung der 8 DIP-Schalter wird bei Veränderung über den CAN-Bus an das jeweils andere Board übertragen und dort mittels der LEDs dargestellt.

Änderungen der Analogwerte werden ebenfalls über den CAN-Bus ausgesendet und an dem anderen Board über die serielle Schnittstelle an einen angeschlossenem PC ausgegeben.

Quellcode

```
#include <reg515c.h>  // üFr Zugriff auf SFR Register des C515C
#include <rtx51tny.h> // RTX-51 tiny functions, 10 MHz, intern 20 MHz;
                      // machine cycle= 0.5 us;
                      // tick= 10 000 machine = 5ms; (gemessen 5 ms);
                      // Timeout = 5 ticks= 25 ms
#include <stdio.h>    // üfr Zugriff auf Standard In Out
#include <string.h>
#include "canlib.h"


#define BASE 0x20
#define REMOTE 0x10

// Deklaration der äbinren Ein- äAusgnge
#define DIP P5
#define LED P1

#define TASK_INIT      0
#define TASK_TIME      1


// Deklaration der externen Funktionen
//extern int mess_ad_kanal(unsigned char MessKanalNr);

void Task_init(void) _task_ TASK_INIT
   {
      can_init(125);
      can_define_irq(1,0,0);

      can_define_msg(1,RECEIVE,BASE);      // Empfangen: LED-Status
      can_define_msg(2,TRANSMIT,REMOTE);   // Senden: Schalterstellung
      can_define_msg(3,TRANSMIT,BASE+1);   // Senden: time-tick


      can_define_rxi(1,1);

      os_create_task(TASK_TIME);           //Starts task

```

```
39        while (1)
40           {
41                                                  // do nothing
42              os_wait (K_IVL, 170, 0);
43           }
44     }
45
46  void Task_TIME(void) _task_ TASK_TIME {
47        static unsigned char dip_old = 0;
48        unsigned char dip;
49
50        while (1)
51           {
52             dip = DIP;
53             if ( dip != dip_old ) {
54               can_send_byte(2,dip);
55               dip_old = dip;
56               printf("Switch: %i\n",(int)dip);
57             }
58           }
59
60     }
61
62  void ISR_can (void) interrupt 17 {
63    unsigned char db;
64    int returnval;
65    while ( returnval = can_handle_irq() ) {
66      printf("Interrupt: %i\n",returnval);
67          if ( returnval == 3 ) {
68             if (can_poll_byte(1,&db) >=0 ) {
69               printf("Received: %i\n",(int)db);
70               LED = db;
71                    }
72           }
73    }
74  }
```

Kapitel 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

Im Rahmen der Arbeit ist es gelungen, eine vollständige Layer2-C-Library für den vorgegebenen CAN-Controller zu schreiben, der in dem Infineon c515c Microcontroller integriert ist. Es wurde erreicht, dass die vom Controller gebotene Funktionalität voll unterstützt wird und über einfache Funktionsaufrufe nutzbar ist.

Zur Entwicklung der Library wurde eine Testumgebung aufgebaut, wobei 2 verschiedene Interfaces selbst gebaut wurden und ein älteres kommerzielles Geät zum Einsatz kam. Insbesondere bei dem Elektor-Interface konnten verschiedene Designfehler erfolgreich beseitigt werden.

Mit dieser Testumgebung wurde jede Funktion der Library überprüft.

Desweiteren wurden für 2 verschiedene Microcontroller-Expeimentierboards Beispielanwendungen implementiert, sowie Lösungen für Kompatibilitätsprobleme bei der Hardware erarbeitet.

5.2 Ausblick

Dieses Projekt könnte z.B. als Praktikumsversuch zur CAN-Kommunikation weitergeführt werden, dazu müsste allerdings ein stabiler CAN-Monitor zur Verfügung stehen, der besser aus einem PC mit aktueller CAN-PCI-Karte, als mit Parallelport-Adaptern aufgebaut sein sollte. Für den Einsatz in Praktika reicht die Stabilität der oben aufgeführten Interfaces nicht mehr aus.

Ebenfalls wäre es denkbar, auf dieser Layer2-Library ein höheres Protokoll, wie z.B. CAN-Open aufzusetzen. Ein vielversprechender Ansatz wäre hier auf Micro-CAN-Open[1] aufzusetzen. Dieses Projekt implementiert das CAN-Open-Protokoll in teilen, die in der Automatisierung erforderlich sind, läßt aber viele Management-Funktionen entfallen, so dass es selbst bei begrenztem Speicherplatz in Microcontrollern verwendbar ist.

Bei CAN-Open werden üblicherwiese mehr IDs verwendet, als ein Full-CAN-Controller parallel verarbeiten kann, weshalb für dieses Protokoll meist nur der Basic-CAN-Mode eines Full-CAN-Controllers verwendet werden kann. Die Vorteile, die ein Full-CAN-Controller bei kleineren Automatisierungsaufgaben bietet, bleiben daher ungenutzt, außerdem kann es dazu kommen, dass Pakete verloren gehen, wenn der Prozessor ein empfangenes Paket nicht schnell genug aus dem Empfangspuffer liest, der bei diesem Controller nur eine Nachricht aufnehmen kann. Um diesem Problem entgegenzuwirken wurde für CAN-Open und andere komplexe Anwendungsprotokolle das PeliCAN-Konzept entwickelt, das über einen mehrere Byte langen FIFO verfügt.

[1] http://www.microcanopen.com

Kapitel 6

Anhang

6.1 Quellcode canlib.h

```
1  #ifndef __canlib__
2  #define __canlib__
3
4  #define TRANSMIT 0x18
5  #define RECEIVE 0x10
6
7
8  // Supported Bitrates: 10, 20, 50, 100, 125, 250, 500 and 1000 kBit/s
9  void can_init(int baudrate);
10
11
12 // obj_nr = 1..15
13
14 // define msg: no interrupt generation by default
15 void can_define_msg(char obj_nr, char type, short id);
16 void can_define_extmsg(char obj_nr, char type, long extid);
17
18 // setup interrupts
19 void can_define_irq(bit IE, bit SIE, bit EIE);
20 void can_define_txi(char obj_nr, bit enable);
21 void can_define_rxi(char obj_nr, bit enable);
22
23 // send—commands immedeately send a message
```

```
24 void can_send_pulse(char obj_nr);
25 void can_send_byte(char obj_nr, char db);
26 void can_send_ptr(char obj_nr, const void* tdata, char n);
27
28 // set-commands prepare a message for sending, or to be requested by
29 // a remote frame.
30 void can_set_byte(char obj_nr, char db);
31 void can_set_ptr(char obj_nr, const void* tdata, char n);
32
33 // can_trigger sends a prepared object (TRANSMIT-Object)
34 void can_trigger(char obj_nr);
35
36 // poll-commands return -1 is no message is pending and >=0 if a message
37 // is recieved
38 // can_poll_string() and can_poll_ptr() return the number of data-bytes
39 // receivied
40 // can_poll_string writes the received bytes to a 0-terminated string (rdata)
41 //
42 int can_poll_pulse(char obj_nr);
43 int can_poll_byte(char obj_nr, char *db);
44 int can_poll_ptr(char obj_nr, void *rdata);
45 int can_poll_string(char obj_nr, char *rdata);
46
47 // send a remote request (RECEIVE-Object)
48 void can_remote(char obj_nr);
49
50 // polls the message object until a message is reveived.
51 // between polling the task is halted by the os.
52 void can_wait(char obj_nr);
53
54 void printbin(char x);
55
56 // can_handle_irq returns:
57 // 0    : no interrupt pending
58 // 1    : status change
59 // 2    : Message 15
60 // (2+N): Message N
61 int can_handle_irq(void);
62
```

```
63 // can_check_lec: return last error code
64 // this function is untested because i had no possibility to
65 // generate errors
66 // Possible error codes are:
67 #define NO_ERROR 0
68 #define STUFF_ERROR 1
69 #define FORM_ERROR 2
70 #define ACK_ERROR 3
71 #define BIT1_ERROR 4
72 #define BIT0_ERROR 5
73 #define CRC_ERROR 6
74 int can_check_lec(void);
75
76 // TODO
77 bit can_check_txok(void);
78 void can_clear_txok(void);
79 bit can_check_rxok(void);
80 void can_clear_rxok(void);
81 bit can_check_boff(void);
82 bit can_check_ewrn(void);
83
84 #endif
```

6.2 Quellcode canlib.c

```
1  /*
2  CanController Library
3  by Daniel Schramm
4
5  Microcontroller:  c515c
6  Compiler:         Keil C-Compiler
7  Operating System: RTX-Tiny
8  */
9
10 #include "canreg.h"
11 #include "Regc515c.h"
12 #include "Intc515c.h"
13 #include <string.h>
14 #include <stdio.h>
15 #include <rtx51tny.h>
```

```
16
17 #define REMOTE 2
18
19 unsigned char pdata canreg[256];
20 extern unsigned char IE;
21 extern unsigned char SIE;
22 extern unsigned char EIE;
23
24
25 // create dynamic register names
26 #define         MCR0_n   canreg [ (obj_nr * 16) + 0 ]
27 #define         MCR1_n   canreg [ (obj_nr * 16) + 1 ]
28 #define         UAR0_n   canreg [ (obj_nr * 16) + 2 ]
29 #define         UAR1_n   canreg [ (obj_nr * 16) + 3 ]
30 #define         LAR0_n   canreg [ (obj_nr * 16) + 4 ]
31 #define         LAR1_n   canreg [ (obj_nr * 16) + 5 ]
32 #define         MCFG_n   canreg [ (obj_nr * 16) + 6 ]
33 #define         DB0_n    canreg [ (obj_nr * 16) + 7 ]
34 #define         DB1_n    canreg [ (obj_nr * 16) + 8 ]
35 #define         DB2_n    canreg [ (obj_nr * 16) + 9 ]
36 #define         DB3_n    canreg [ (obj_nr * 16) + 10 ]
37 #define         DB4_n    canreg [ (obj_nr * 16) + 11 ]
38 #define         DB5_n    canreg [ (obj_nr * 16) + 12 ]
39 #define         DB6_n    canreg [ (obj_nr * 16) + 13 ]
40 #define         DB7_n    canreg [ (obj_nr * 16) + 14 ]
41
42
43 // only for debugging
44 void printbin(char x) {
45   char i;
46   for (i=1; i<9; i++) {
47     if (x & 0x80)
48       printf("1");
49         else
50           printf("0");
51         x = x << 1;
52   }
53   printf("\n");
54 }
```

```
55
56
57  void can_init (int baudrate) {
58    char obj_nr;
59
60    SYSCON &= 0xFC;  // XMAP 0=0, XMAP 1=0
61    XPAGE = 0xF7;    // CAN memory space in XRAM
62
63    GMS0 = 0xFF;     // global mask short
64    GMS1 = 0xFF;
65
66    UGML0 = 0xFF;    // global mask long
67    UGML1 = 0xFF;
68    LGML0 = 0xFF;
69    LGML1 = 0xFF;
70
71    UMLM0 = 0xFF;    // last message mask
72    UMLM1 = 0xFF;
73    LMLM0 = 0xFF;
74    LMLM1 = 0xFF;
75
76    CR = 0x41;       // INIT=1, CCE=1 (allow access to BTR)
77
78    // Bitrate
79    switch (baudrate) {
80      case 10:
81        BTR0 = 0x67;  BTR1 = 0x7f;    // Bit Timing Register (10MHz, 10 kBaud)
82        break;
83      case 20:
84        BTR0 = 0x53;  BTR1 = 0x7f;    // Bit Timing Register (10MHz, 20 kBaud)
85        break;
86      case 50:
87        BTR0 = 0x47;  BTR1 = 0x7f;    // Bit Timing Register (10MHz, 50 kBaud)
88        break;
89      case 100:
90        BTR0 = 0x43;  BTR1 = 0x7f;    // Bit Timing Register (10MHz, 100 kBaud)
91        break;
92      case 125:
93        BTR0 = 0xC4;  BTR1 = 0x49;    // Bit Timing Register (10MHz, 125 kBaud)
```

```
94        break;
95      case 250:
96        BTR0 = 0xC1;  BTR1 = 0x6B;    // Bit Timing Register (10MHz, 250 kBaud)
97        break;
98      case 500:
99        BTR0 = 0xC0;  BTR1 = 0x6B;    // Bit Timing Register (10MHz, 500 kBaud)
100       break;
101     case 1000:
102       BTR0 = 0x80;  BTR1 = 0x25;    // Bit Timing Register (10MHz, 1 MBaud)
103       break;
104     default:
105       BTR0 = 0xC4;  BTR1 = 0x49;    // Bit Timing Register (10MHz, 125 kBaud)
106       break;
107   }
108   CR = 0x01;       // INIT=1, CCE=0 (permit access to BTR)
109   SR = 0xE7;       // clear TXOK and RXOK
110
111   for ( obj_nr = 1; obj_nr <= 15; obj_nr ++)
112     MCR0_n = 0x55;   // set Messages invalid
113
114   CR = 0;   // INIT=0 return to normal Operation
115 }
116
117
118 void can_define_irq(bit IE, bit SIE, bit EIE) {
119    IEN0 = IEN0 | 0x80;
120    IEN2 = IE ? (IEN2 | 0x02):(IEN2 & 0xfd);
121    CR = IE ? (CR | 0x02):(CR & 0xfd);
122    CR = SIE ? (CR | 0x04):(CR & 0xfb);
123    CR = EIE ? (CR | 0x08):(CR & 0xf7);
124 }
125
126
127 void can_define_txi(char obj_nr, bit enable) {
128    MCR0_n = enable?0xef:0xdf;
129 }
130
131
132 void can_define_rxi(char obj_nr, bit enable) {
```

```
133     MCR0_n = enable?0xfb:0xf7;
134 }
135 
136 
137 void can_define_msg(char obj_nr, char type, short id) {
138   char temp[2];
139   MCR1_n = 0xFB;        // CPUUPD=1
140   id = id << 5;
141   memcpy(temp, &id, 2);
142   UAR0_n = temp[0];     // idenifier
143   UAR1_n = temp[1];
144 
145   MCFG_n = type;
146   MCR0_n = 0x95;
147   MCR1_n = 0x55;
148 }
149 
150 
151 void can_define_extmsg(char obj_nr, char type, long extid) {
152   char temp[4];
153 
154   MCR1_n = 0xFB;        // CPUUPD=1
155   extid = extid << 3;
156   memcpy(temp, &extid, 4);
157   UAR0_n = temp[0];
158   UAR1_n = temp[1];
159   LAR0_n = temp[2];
160   LAR1_n = temp[3];
161 
162   MCFG_n = type | 0x04;
163   MCR0_n = 0x95;
164   MCR1_n = 0x55;
165 }
166 
167 
168 void can_set_byte(char obj_nr, char db) {
169   MCR0_n = 0x7f;            // set invalid
170   MCFG_n = MCFG_n & 0x1f;
171   DB0_n = db;
```

```
172   MCR0_n = 0xbf;            // set valid
173 }
174
175
176 void can_set_ptr(char obj_nr, const void* tdata, char n) {
177   n=(n>8)?8:n;
178   n=(n<0)?0:n;
179
180   MCR0_n = 0x7f;            // set invalid
181   memcpy (&DB0_n, tdata, n);
182   MCFG_n = (n << 4) | ( MCFG_n & 0x0f);
183   MCR0_n = 0xbf;            // set valid
184 }
185
186
187 void can_trigger(char obj_nr) {
188   MCR1_n = 0xEF;
189 }
190
191
192 void can_send_pulse(char obj_nr) {
193   MCR0_n = 0x7f;            // set invalid
194   MCFG_n = MCFG_n & 0x0f;
195   MCR0_n = 0xbf;            // set valid
196   MCR1_n = 0xEF;
197 }
198
199
200 void can_send_byte(char obj_nr, char db) {
201   can_set_byte(obj_nr, db);
202   can_trigger(obj_nr);
203 }
204
205
206 void can_send_ptr(char obj_nr, const void* tdata, char n) {
207   can_set_ptr(obj_nr, tdata, n);
208   can_trigger(obj_nr);
209 }
210
```

```
211
212
213 int can_poll_pulse(char obj_nr) {
214   if ( MCR1_n & 0x02 ) {
215     MCR0_n = 0x7f;     // set message invalid
216     MCR1_n = 0xf5;     // clear CPUUPD and NEWDAT
217     MCR0_n = 0xbf;     // set message valid
218     return(0);
219   } else
220     return(-1);
221 }
222
223
224 int can_poll_byte(char obj_nr, char *db) {
225   if ( MCR1_n & 0x02 ) {
226     MCR0_n = 0x7f;     // set message invalid
227     *db = DB0_n;       // read
228     MCR1_n = 0xf5;     // clear CPUUPD and NEWDAT
229     MCR0_n = 0xbf;     // set message valid
230     return(1);
231   } else
232     return(-1);
233 }
234
235
236 int can_poll_ptr(char obj_nr, void *rdata) {
237   char n;
238   if ( MCR1_n & 0x02 ) {
239     n=(MCFG_n >> 4);
240     MCR0_n = 0x7f;      // set message invalid
241     memcpy(rdata, &DB0_n, n);      // read
242     MCR1_n = 0xf5;      // clear CPUUPD and NEWDAT
243     MCR0_n = 0xbf;      // set message valid
244     return(n);
245   } else
246     return(-1);
247 }
248
249 int can_poll_string(char obj_nr, char *rdata) {
```

```
250   char n;
251   if ( MCR1_n & 0x02 ) {
252     n=(MCFG_n >> 4);
253     MCR0_n = 0x7f;       // set message invalid
254     memcpy(rdata, &DB0_n, n);      // read
255         rdata[n]='\0';
256     MCR1_n = 0xf5;       // clear CPUUPD and NEWDAT
257     MCR0_n = 0xbf;       // set message valid
258     return(n);
259   } else
260     return(-1);
261 }
262
263
264 void can_remote(char obj_nr) {
265   MCR1_n = 0xEF;
266 }
267
268
269 int can_handle_irq(void) {
270   unsigned char intreg;
271   char obj_nr;
272
273   if (intreg = IR) {
274     if (intreg > 2) {
275       obj_nr = intreg - 2;
276       MCR0_n = 0xfd;
277         } else if ( intreg == 2) {
278           MCR0_M15 = 0xfd;
279         }
280         return(intreg);
281   } else
282     return(0);
283 }
284
285
286 void can_wait(char obj_nr) {
287   while ( ! ( MCR1_n & 0x02) )
288     os_wait(K_TMO, 50, 0);
```

```
289 }
290
291
292 int can_check_lec(void) {
293   char temp;
294   temp = SR;
295   SR = temp | 0x07;
296   return (temp && 0x07);
297 }
298
299
300 bit can_check_txok(void) {
301   return ( SR && 0x08 );
302 }
303
304
305 void can_clear_txok(void) {
306   SR = ( SR | 0x08 ) ^ 0x08;
307 }
308
309
310 bit can_check_rxok(void) {
311   return ( SR && 0x10 );
312 }
313
314
315 void can_clear_rxok(void){
316   SR = ( SR | 0x10 ) ^ 0x10;
317 }
318
319
320 bit can_check_boff(void) {
321   return ( SR && 0x80 );
322 }
323
324
325 bit can_check_ewrn(void) {
326   return ( SR && 0x40 );
327 }
```

6.3 Anmerkungen zum Quellcode

In dem Quellcode werden dynamische Registernamen mit "_n" verwendet.

Diese Register sind abgeleitet von den Registernamen, die für jedes CAN-Objekt, wobei das "_n" andeuten soll, dass die Registeradresse anhand des Funktionsparameters obj_nr dynamisch berechnet werden.

Die Definition dieser Register-Aliases ist so erfolgt, dass sie in C wie Konstante Adressen verwendet werden können, im Hintergrund findet allerdings eine Umsetzung der Adresse für das jeweilige Objekt statt.

Kapitel 7

Referenzen / Literatur

C515C Datenblatt

Controller- Area- Network, Konrad Etschberger, ISBN: 3446217762

http://www.can.bosch.com/: Spezifikationen zu CAN.

http://private.addcom.de/horo/can200: Can200 Hardware und ein passender alter Linux-Treiber

http://canbus4linux.sourceforge.net/: Linux-Treiber für Can200 und Elektor Can-Interface

http://home.t-online.de/home/0702483486-0001/can/can.htm: Windows-Treiber für Can200 und Elektor Can-Interface

http://www.microcanopen.com/: Open-Source CAN-Open library fuer Microcontroller

http://www.fset.de/~daniel/timer.html: Beelichtungstimer