

basicCAR 3085

USB 2.0 Stand-alone Gerät für
CAN/ LIN/ K-LINE/ J1850 Interfaces
Nutzerhandbuch Version 1.1

© 2010 GÖPEL electronic GmbH. Alle Rechte vorbehalten.

Die in diesem Handbuch beschriebene Software sowie das Handbuch selbst dürfen nur in Übereinstimmung mit den Lizenzbedingungen verwendet oder kopiert werden.
Zu Sicherungszwecken darf der Käufer eine Kopie der Software anfertigen.

Der Inhalt des Handbuchs dient ausschließlich der Information, ist nicht als Verpflichtung der GÖPEL electronic GmbH anzusehen und kann ohne Vorankündigung verändert werden.
Hard- und Software unterliegen ebenso möglichen Veränderungen im Sinne des technischen Fortschritts.

Die GÖPEL electronic GmbH übernimmt keinerlei Gewähr oder Garantie für Genauigkeit und Richtigkeit der Angaben in diesem Handbuch.

Ohne vorherige schriftliche Genehmigung der GÖPEL electronic GmbH darf kein Teil dieser Dokumentation in irgendeiner Art und Weise übertragen, vervielfältigt, in Datenbanken gespeichert oder in andere Sprachen übersetzt werden (es sei denn, dies ist durch die Lizenzbedingungen ausdrücklich erlaubt).

Die GÖPEL electronic GmbH haftet weder für unmittelbare Schäden noch für Folgeschäden aus der Anwendung ihrer Produkte.

Gedruckt: 09.06.2010

Alle in diesem Handbuch verwendeten Produkt- und Firmennamen sind Markennamen oder eingetragene Markennamen ihrer jeweiligen Eigentümer.

Stand: Juni 2010

1	INSTALLATION	1-1
1.1	HARDWAREINSTALLATION	1-1
1.2	TREIBERINSTALLATION	1-2
2	HARDWARE	2-1
2.1	BESTIMMUNG	2-1
2.2	TECHNISCHE DATEN	2-3
2.2.1	<i>Abmessungen</i>	2-3
2.2.2	<i>Kennwerte</i>	2-3
2.3	AUFBAU	2-4
2.3.1	<i>Allgemeines</i>	2-4
2.3.2	<i>Relais</i>	2-5
2.3.3	<i>Kommunikationsschnittstellen</i>	2-6
2.3.4	<i>Peripherie-versorgung</i>	2-8
2.3.5	<i>Adressierung</i>	2-8
2.3.6	<i>Bestückung</i>	2-9
2.3.7	<i>Belegung Frontsteckverbinder</i>	2-10
2.3.8	<i>LED Anzeige</i>	2-11
2.4	LIEFERHINWEISE	2-12
3	ANSTEUERSOFTWARE	3-1
3.1	PROGRAMMIEREN ÜBER G-API	3-1
3.2	PROGRAMMIEREN ÜBER DLL-FUNKTIONEN	3-1
3.2.1	<i>Windows Device Treiber</i>	3-2
3.2.1.1	<i>Driver_Info</i>	3-3
3.2.1.2	<i>DLL_Info</i>	3-4
3.2.1.3	<i>Write_FIFO</i>	3-5
3.2.1.4	<i>Read_FIFO</i>	3-6
3.2.1.5	<i>Read_FIFO_Timeout</i>	3-7
3.2.1.6	<i>Write_COMMAND</i>	3-8
3.2.1.7	<i>Read_COMMAND</i>	3-9
3.2.1.8	<i>Xilinx_Download</i>	3-10
3.2.1.9	<i>Xilinx_Version</i>	3-11
3.3	PROGRAMMIEREN MIT LABVIEW	3-12
3.3.1	<i>LabVIEW über G-API</i>	3-12
3.3.2	<i>LLB unter Verwendung des Windows Device Treibers</i>	3-12
3.4	WEITERE GÖPEL SOFTWARE	3-12
3.5	STEUERBEFEHLE USB CONTROLLER	3-13
3.5.1	<i>USB Befehlsaufbau</i>	3-13
3.5.2	<i>USB Antwortaufbau</i>	3-13
3.5.3	<i>USB Befehle</i>	3-13

1 Installation

1.1 Hardwareinstallation

Die Hardware-Installation beschränkt sich bei basicCAR 3085 i. Allg. auf den Austausch von Transceivermodulen.



Stellen Sie bitte unbedingt sicher, dass alle Installationsarbeiten im **ausgeschalteten** Zustand Ihres Systems erfolgen!

Wenn es notwendig ist, Transceivermodule zu tauschen, wird das entsprechende Gerät gemäß seinen Gegebenheiten geöffnet.

Dabei sind die allgemeinen Regeln zur Vermeidung von elektrostatischen Aufladungen zu beachten.

Transceivermodule dürfen nie unter Spannung gezogen oder gesteckt werden!

Außerdem ist unbedingt ein lagerichtiges Stecken der Module zu realisieren (siehe [Bestückung](#)).

1.2 Treiberinstallation

Um die GÖPEL electronic USB-Treiber auf Ihrem System einzurichten, muss das GUSB Treiber-Setup ausgeführt werden. Starten Sie dazu das auf der mitgelieferten CD enthaltene Setup Programm *GUSB-Setup-*.exe* (der Stern steht für die Versionsnummer) und folgen Sie den Anweisungen.



Der zur Verfügung stehende Devicetreiber unterstützt gegenwärtig ausschließlich Windows® 2000/ XP-Systeme!

Wenn Sie eigene Software für basicCAR 3085-Baugruppen erstellen wollen, benötigen Sie ggf. zusätzliche Dateien für die anwenderspezifische Programmierung (*.LLB, *.H). Diese werden nicht automatisch übernommen und müssen deshalb manuell von der mitgelieferten CD in Ihr Entwicklungsverzeichnis kopiert werden.



Die USB-Schnittstelle nutzt, falls möglich, die high-speed Datenrate entsprechend USB2.0 Spezifikation (ansonsten full-speed).

Nach der Treiberinstallation können Sie überprüfen, ob die Baugruppen einwandfrei vom System eingebunden worden sind. Die folgende Abbildung zeigt die erfolgreiche Einbindung von vier basicCAR 3085-Geräten:

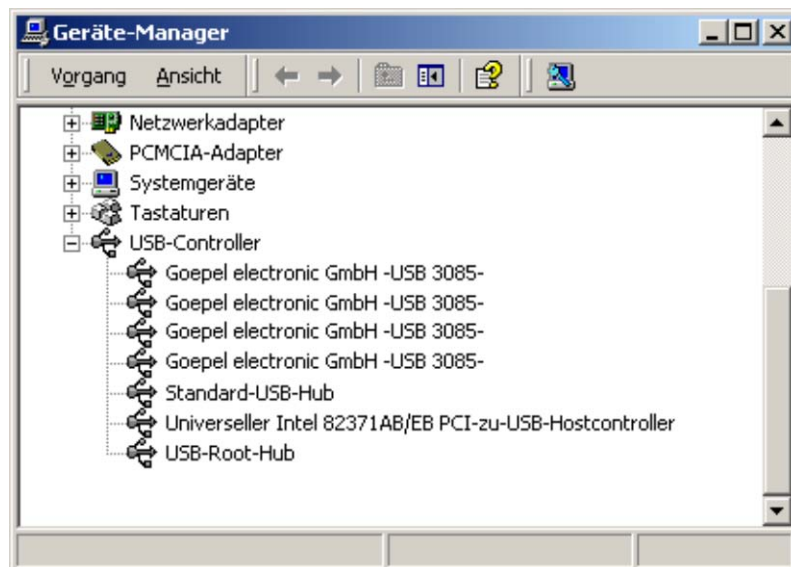


Abbildung 1-1:
Anzeige Geräte-Manager



Beachten Sie bitte, dass der Geräte-Manager ALLE USB-Controller anzeigt.

2 Hardware

2.1 Bestimmung

basicCAR 3085 ist ein GÖPEL electronic GmbH stand-alone USB-Gerät zum Anschluss an einen PC oder Laptop, das für den eigenständigen Einsatz außerhalb komplexer Testsysteme entwickelt wurde. Gegenüber USB 3080/ basicCAR 3080 besitzt das basicCAR 3085 Anschlüsse zur Verteilung und Steuerung der Betriebsspannung von Peripheriegeräten.

Die externe Spannungszufuhr erlaubt die Nutzung dieses Gerätes zur Datenaufnahme und Signalkontrolle bei einer Vielzahl von Anwendungen, z.B. in Kraftfahrzeugen.



Abbildung 2-1:
basicCAR 3085



Beachten Sie bitte, dass ein Download des Xilinx FPGAs für die Funktion einer basicCAR3085-Baugruppe unabdingbar ist (siehe [Xilinx Download](#) unter [Windows Device Treiber](#))!

Die Stromversorgung mit 8-25 VDC (und ca. 350 mA bei 12 V) erfolgt über die Buchse ext. Power basicCAR an der dem Steckverbinder für die Kommunikationsschnittstellen gegenüber liegenden Geräteseite:



Abbildung 2-2:
Rückseite

Diese Buchse wird zur Versorgung der internen Logik genutzt. Der GND-Anschluss dieser Buchse ist mit dem GND-Anschluss der USB-Schnittstelle verbunden.

Die beiden Eingangsbuchsen für die Peripherieversorgung ext. Power Front (rot = plus/ schwarz = minus) befinden sich daneben. Alle Anschlüsse der Kommunikationsschnittstellen sowie der Peripheriespannung sind galvanisch vom USB-Interface und von der internen Logik getrennt.

Ebenfalls auf der Rückseite des basicCAR 3085 finden Sie den Anschluss für das USB-Kabel.

basicCAR 3085-Baugruppen bieten in der maximalen Ausbaustufe folgende Ressourcen:

- ◆ 2 x CAN
- ◆ 2 x LIN oder K-Line
- ◆ 1 x J1850 VPW
- ◆ 1 x J1850 PWM
(in diesem Fall ist nur EIN LIN oder K-Line Interface möglich)
- ◆ 4 x digital Input
- ◆ 4 x digital Output
- ◆ 2 x analog Input
- ◆ 1 x Weckleitung
- ◆ 1 x Peripheriespannungsausgang (2-pol.)
- ◆ 3 x abschaltbare Peripheriespannungsausgänge (1-pol.)



Die Kommunikation für J1850 PWM erfolgt über die Schnittstelle für K-Line/LIN2!!!

2.2 Technische Daten

- 2.2.1 Abmessungen (Breite x Höhe x Tiefe):
- ♦ basicCAR 3085: 145 mm x 70 mm x 220 mm

2.2.2 Kennwerte

Symbol	Kennwert	Min.	Typ.	Max.	Einheit	Bemerkung
V _{EXT}	Externe Versorgungsspannung	8	12	25	V _{DC}	
V _{BAT}	Batteriespannung		12	27	V _{DC}	Abh. v. Transceivertyp
V _{PER_IN}	Peripheriespannungseingang			30	V _{DC}	
I _{PER_IN}	Peripheriestromeingang			10	A	
I _{PER_OUT}	Peripheriestromausgang			5 (10)	A	10A NUR für nicht abschaltbaren Ausgang
	Übertragungsrate			1	MBaud	CAN
	Übertragungsrate			22	kBaud	LIN
R _{bus}	Abschlusswiderstand 1		120		Ohm	CAN Jumper gesteckt
R _{bus}	Abschlusswiderstände 2		10		kOhm	CAN Jumper gesteckt
R _{Pullup}	Pullup-Widerstand		680		Ohm	K-Line Jumper gesteckt
V _{in}	Eingangsspannung	3,3		50	V _{DC}	Digital Input
V _{out}	Ausgangsspannung			V _{BAT}	V _{DC}	Digital Output, OC
V _{in}	Eingangsspannung		20	26	V _{DC}	Analog Input
V _{iso}	galvanische Trennung	560			V _{DC}	USB In-/Output



Die analogen Eingangs-Kanäle (analog inputs) sind mit dem Schaltkreis LTC 1400 (AD-Wandler) von Linear Technology realisiert. Dieses Bauelement weist eine Auflösung von 12 Bit sowie einen Eingangsspannungsbereich von 0..4,095V auf.

Durch den Eingangs-Spannungsteiler (122K/22K) ergibt sich für die gemessene Spannung:

$$V_{\text{mess}} = \text{AD-Wandlerwert} * 1\text{mV} * (122\text{K}/22\text{K}).$$



Beachten Sie bitte, dass die Höhe der Versorgungsspannung für die Transceiver (Batteriespannung V_{Bat}) von dem Transceiver bestimmt wird, der die NIEDRIGSTE maximale Versorgungsspannung hat!

2.3 Aufbau

2.3.1 Allgemeines

Bei den einzelnen basicCAR 3085-Baugruppen dient ein ASIC (TC1775) als Interface zum USB-Bus. Dieser beinhaltet alle notwendigen Funktionsblöcke, die für eine Kommunikation mit dem Rechner-Bus notwendig sind.

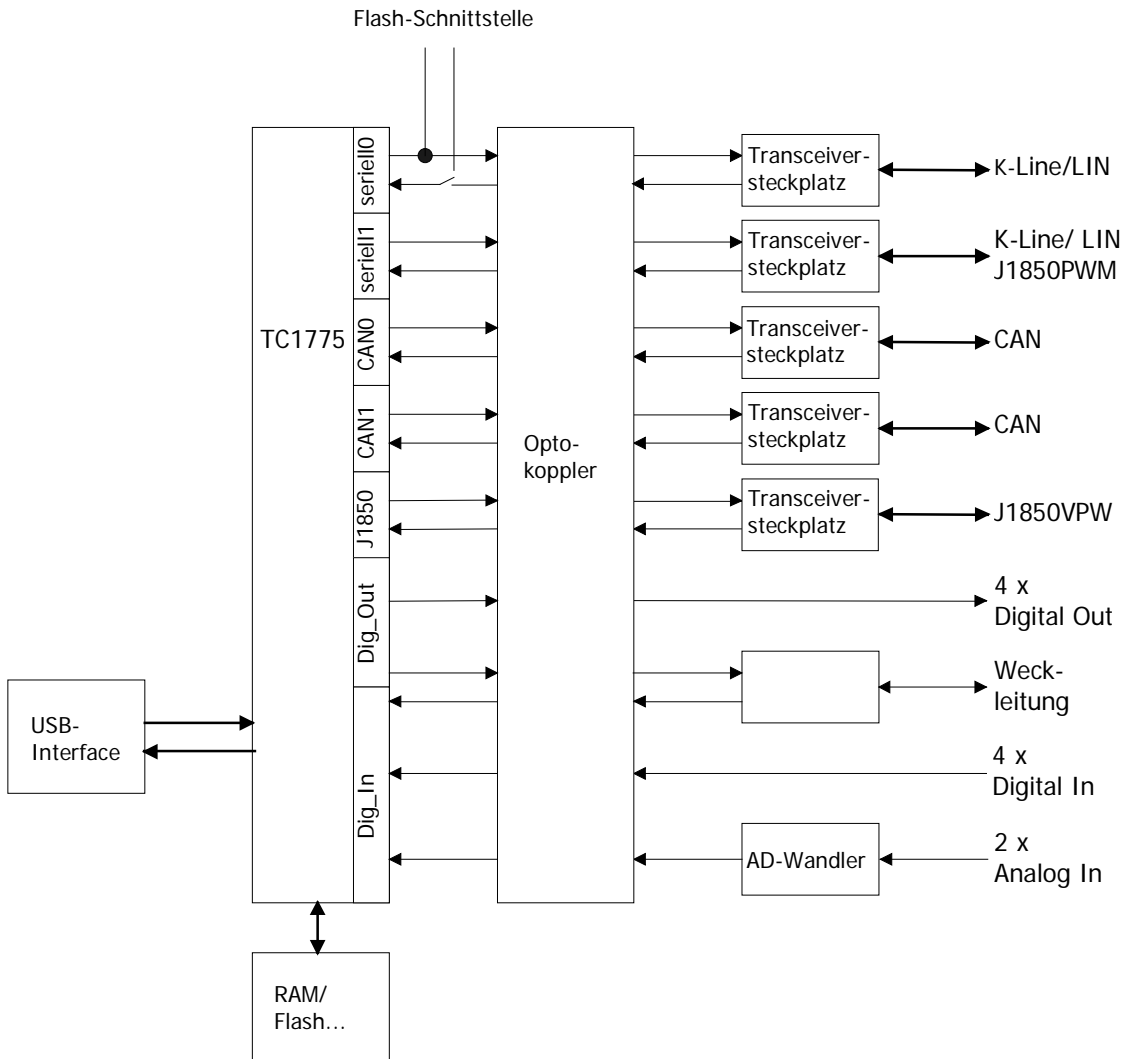


Abbildung 2-3: Blockdiagramm des Kommunikationsboards für basicCAR 3085



warnung

Bitte verwenden Sie zum Anschluss der Baugruppen an die USB-Schnittstelle des PCs die im Lieferumfang enthaltenen USB-Kabel. Andere Kabel sind u. U. nicht geeignet!

2.3.2 Relais

Ihr basicCAR 3085 verfügt über fünf Relais, mit denen sowohl die Master-/Slave-Umschaltung der LIN-Schnittstellen realisiert wird als auch das Schalten der abschaltbaren Peripheriespannungen.

Mit den Firmwarebefehlen 0x81 ... 0x84 der Digital Befehle werden die Relais geschaltet (siehe Dokumentation GÖPEL Firmware).

Bitte verwenden Sie für diese Befehle die Relais-Nummern entsprechend folgender Tabelle:

Relais	Bedeutung
1	LIN1 Master-/ Slave-Konfiguration Siehe Kommunikationsschnittstellen / 2 x LIN-Interface
2	LIN2 Master-/ Slave-Konfiguration Siehe Kommunikationsschnittstellen / 2 x LIN-Interface
3	Externer Relais-Ausgang Rel1
4	Externer Relais-Ausgang Rel2
5	Externer Relais-Ausgang Rel3



Die Master-/ Slave-Konfiguration der LIN-Schnittstellen kann auch über die LIN Befehle 0x81 ... 0x84 erfolgen (siehe Dokumentation GÖPEL Firmware).

2.3.3 Kommunikations-schnittstellen

2 x CAN-Interface:

Für die uneingeschränkte Funktion eines CAN-Interfaces an einem Netzwerk ist der verwendete Transceiver entscheidend. Häufig funktionieren CAN-Netzwerke nur, wenn alle Teilnehmer kompatible Transceiver im Netz haben.

Damit die Nutzer eines basicCAR 3085 keinen Einschränkungen unterliegen, sind die Transceiver als steckbare Module ausgeführt. Dabei stehen verschiedene Varianten (Highspeed, Lowspeed, Single-Wire u.a.) zur Auswahl, die einfach auszutauschen sind.

Neben dem Transceiver ist der Busabschlusswiderstand für die einwandfreie Funktion des CAN-Netzwerkes wichtig.

Werden Highspeed CAN-Transceiver verwendet, ist i. Allg. ein 120 Ohm Widerstand für jede CAN-Schnittstelle aktiv. Diese Widerstände können durch Ziehen der Jumper J1401 bzw. J1501 deaktiviert werden.

Bei Verwendung von Lowspeed CAN-Transceivern sind i. Allg. zwei Abschlusswiderstände von 10 kOhm für RTH und RTL für jede CAN-Schnittstelle aktiv (durch Bestücken der Jumper J1402/ J1403 bzw. J1502/ J1503). In diesem Fall müssen die Jumper J1401 bzw. J1501 geöffnet sein.

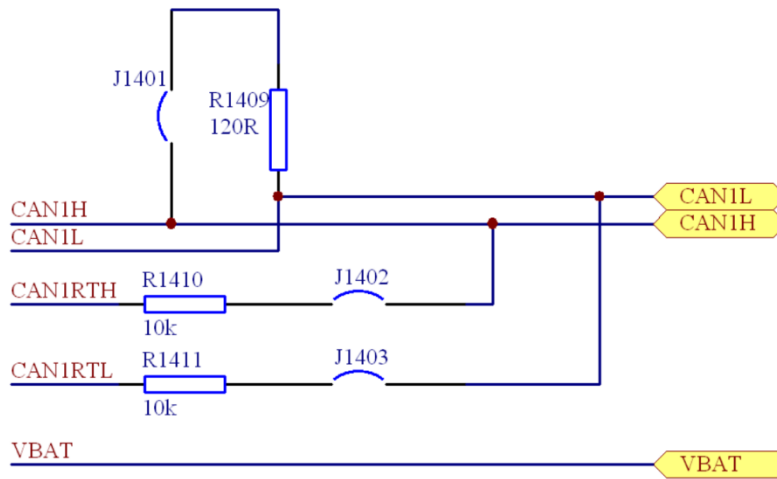


Abbildung 2-4:
CAN Schnittstelle

2 x LIN-Interface oder 2 x K-Line Interface (ISO 9141)

LIN:

Die Transceiver sind als steckbare Module ausgeführt. I. Allg. wird der TJA1020 für diese Transceiver verwendet.

In der Standardausführung der Transceivermodule kann per Software für LIN1 bzw. LIN2 zwischen Master- und Slave-Konfiguration umgeschaltet werden. Für die Master-Konfiguration ist das entsprechende Relais zu setzen, für die Slave-Konfiguration zurückzusetzen (siehe [Relais](#)).

Die Pullup-Widerstände für LIN befinden sich auf dem Transceivermodul, sodass die Jumper J1601 bzw. J1701 NICHT bestückt werden dürfen!

Über die Anschlüsse V_{BAT} wird die Versorgungsspannung der Transceivermodule angeschlossen. Gemäß der LIN-Spezifikation soll diese Versorgung über eine Verpolschutzdiode erfolgen, sodass die Jumper J1602 bzw. J1703 NICHT bestückt werden dürfen.

K-Line:

Die Transceiver sind als steckbare Module ausgeführt. I. Allg. wird der L9637 für diese Transceiver verwendet.

Über die Anschlüsse V_{BAT} wird die Versorgungsspannung der Transceivermodule angeschlossen. Zum Überbrücken der Verpolschutzdiode für V_{BAT} für LIN müssen die Jumper J1602 bzw. J1703 bestückt sein.

Falls der Pullup-Widerstand gegen V_{BAT} aktiviert werden soll, müssen die Jumper J1601 bzw. J1701 bestückt werden.

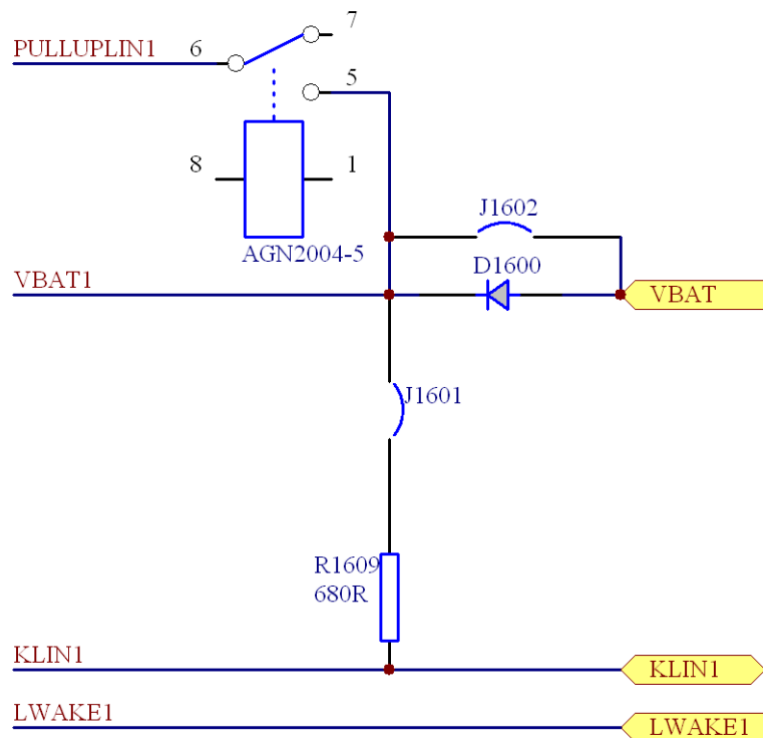


Abbildung 2-5:
LIN/ K-Line Schnittstelle

J1850 Interfaces:

Die Transceiver sind als steckbare Module ausgeführt.

I. Allg. wird der AU5780 für J1850 VPW Transceiver verwendet.

Die Ausgangsschaltung für den J1850 PWM Transceiver ist mit diskreten Bauelementen realisiert.

Falls ein J1850 VPW Interface realisiert werden soll, muss der Transceiver auf dem Steckplatz für den J1850 Transceiver bestückt werden.

Um ein J1850 PWM Interface zu realisieren, wird der Transceiver auf dem Steckplatz für K-Line/ LIN 2 Transceiver bestückt (siehe Abbildung 2-6).



J1701 darf bei einem J1850 PWM Interface NICHT bestückt sein!

2.3.4 Peripherie- versorgung

Die Versorgung externer Geräte ist über die fünf Spannungsbuchsen möglich. Die rote (+) und die schwarze (-) Buchse führen die ungeschaltete Peripheriespannung vom Eingang.

Die gelben Buchsen (Rel 1...Rel 3) führen den Pluspol der abschaltbaren Peripheriespannungen. Nach dem Einschalten des basicCAR 3085 sind diese Spannungen abgeschaltet.

Über die Digital Befehle 0x81...084 können sie aktiviert werden (siehe auch Dokumentation GÖPEL Firmware).

Als Masse-Pol zu den abschaltbaren Spannungen kann die schwarze Buchse (-) verwendet werden.

Die LEDs daneben zeigen den Zustand der Relais an.

2.3.5 Adres- sierung

Die Adressierung von basicCAR 3085-Baugruppen erfolgt ausschließlich über deren Seriennummern (siehe [Ansteuersoftware](#)): Die Baugruppe mit der KLEINSTEN Seriennummer ist immer das Gerät Nummer 1.

2.3.6 Bestückung

Abbildung 2-6 zeigt schematisch die Bestückungsseite eines basicCAR 3085-Boards. In der Abbildung ist die Lage der Transceivermodule, Steckverbinder, DIP-Schalter und Jumper zu erkennen.

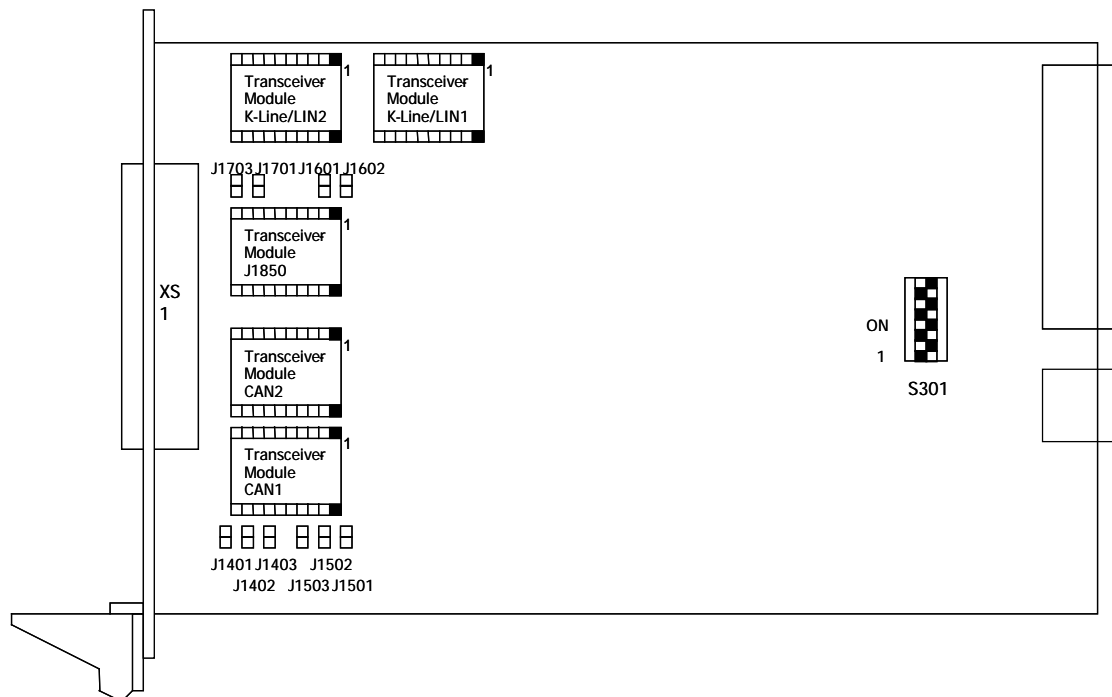


Abbildung 2-6: Schematischer Bestückungsplan eines basicCAR3085-Boards

Die Konfigurationselemente aus Abbildung 2-6 sowie die Bezeichnungen der Steckverbinder für die Transceiverbestückung werden in der folgenden Tabelle erläutert:

XS1401	Transceivermodul für CAN1
J1401	Jumper zum Aktivieren des 120Ω Abschlusswiderstandes für CAN1
J1402	Jumper zum Aktivieren des 10kΩ Abschlusswiderstandes RTH für CAN1
J1403	Jumper zum Aktivieren des 10kΩ Abschlusswiderstandes RTL für CAN1
XS1501	Transceivermodul für CAN2
J1501	Jumper zum Aktivieren des 120Ω Abschlusswiderstandes für CAN2
J1502	Jumper zum Aktivieren des 10kΩ Abschlusswiderstandes RTH für CAN2
J1503	Jumper zum Aktivieren des 10kΩ Abschlusswiderstandes RTL für CAN2
XS1601	Transceivermodul für LIN1/ K-Line1
J1601	Jumper zum Aktivieren des 680Ω Pullupwiderstandes gegen V_{BAT} für K-Line1
J1602	Jumper zum Überbrücken der Verpolschutzdiode für V_{BAT} für LIN1
XS1701	Transceivermodul für LIN2/ K-Line2/ J1850 PWM
J1701	Jumper zum Aktivieren des 680Ω Pullupwiderstandes gegen V_{BAT} für K-Line2
J1703	Jumper zum Überbrücken der Verpolschutzdiode für V_{BAT} für LIN2
XS1801	Transceivermodul für J1850 VPW
S301	DIP-Schalter der basicCAR 3085-Boards zur Konfiguration des Mikrocontrollers. Die Einstellung darf nicht verändert werden!

2.3.7 Belegung Front- steckverbinder

Typ: DSub 25-polig Buchse

Die Schnittstellen stehen über diesen Steckverbinder an der Frontseite des basicCAR 3085 zur Verfügung.

Die Belegung ist in der folgenden Tabelle dargestellt:

lfd. Nr.	Anschluss XS1	Signalname	Bemerkung
1	1	CAN1_High	CAN-Bus-Leitung High
2	14	CAN1_Low	CAN-Bus-Leitung Low
3	2	CAN2_High	CAN-Bus-Leitung High
4	15	CAN2_Low	CAN-Bus-Leitung Low
5	3	VBAT	Versorgungsspannungseingang für Transceiver (siehe Kennwerte)
6	16	GND	Massepotenzial Kommunikationsschnittstellen
7	4	LIN1/ K-Line1	abhängig von Transceiver-Bestückung
8	17	L-Line1/ WAKE1	abhängig von Transceiver-Bestückung
9	5	LIN2/ K-Line2/ J1850 PWM+	abhängig von Transceiver-Bestückung
10	18	L-Line2/ WAKE2/ J1850 PWM-	abhängig von Transceiver-Bestückung
11	6	J1850 VPW	
12	19	-	Bitte nicht belegen!
13	7	VBAT	Versorgungsspannungseingang für Transceiver (siehe Kennwerte)
14	20	GND	Massepotenzial Kommunikationsschnittstellen
15	8	Analog Input1	
16	21	Analog Input2	
17	9	Digital Input1	
18	22	Digital Output1	
19	10	Digital Input2	
20	23	Digital Output2	
21	11	Digital Input 3	
22	24	Digital Output3	
23	12	Digital Input 4	
24	25	Digital Output4	
25	13	Weckleitung	



Bei K-Line sind ggf. auf PIN 17/ 18 die Anschlüsse für die L-Line verdrahtet (in Abhängigkeit von der Ausgangsbeschaltung).



Bei LIN sind ggf. auf PIN 17/ 18 die WAKE-Anschlüsse verdrahtet (in Abhängigkeit von der Transceiver-Wahl).



Pin 3 und 7 (V_{BAT}) sowie Pin 16 und 20 (GND) sind auf jedem Board für basicCAR 3085 gebrückt!

USB-Schnittstelle

Die USB-B-Buchse (USB-Standardbelegung) für das USB 2.0 Interface befindet sich an der dem Steckverbinder für die Kommunikationsschnittstellen gegenüber liegenden Seite des basicCAR 3085.

2.3.8 LED Anzeige

Die LEDs zeigen folgende Zustände an:

grüne LED Rel1: Relais 1 (Peripheriversorgung) an
grüne LED Rel2: Relais 2 (Peripheriversorgung) an
grüne LED Rel3: Relais 3 (Peripheriversorgung) an
unbeschriftete grüne LED: keine Funktion

Rote LED Reset: /HDRST

Hardware Reset Indication-Ausgang des Mikrocontrollers

grüne LED Ub1: Zustandsanzeige Spannung 5V (intern)

grüne LED Ub2: Zustandsanzeige Spannung 3,3V (intern)

grüne LED Ub3: Zustandsanzeige Spannung 2,5V (intern)

gelbe LED 1: Zustandsanzeige CAN 1

gelbe LED 2: Zustandsanzeige CAN 2

gelbe LED 3: Zustandsanzeige K-Line/LIN 1

gelbe LED 4: Zustandsanzeige K-Line/LIN 2

Die LEDs sind folgendermaßen auf der Frontplatte angeordnet:

Rel1

Rel2

Rel3

unbeschriftet

Reset 4

Ub1 3

Ub2 2

Ub3 1

2.4 Lieferhinweise

basicCAR 3085-Baugruppen werden in folgenden Basisvarianten geliefert:

- ◆ 1x CAN Schnittstelle und 1x LIN Schnittstelle oder
- ◆ 1x CAN Schnittstelle und 1x K-Line Schnittstelle

Diese Basisvarianten können durch folgende Optionen erweitert werden:

- ◆ 1x Zusätzliche CAN Schnittstelle
- ◆ 1x Zusätzliche LIN Schnittstelle oder K-Line Schnittstelle
- ◆ 1x Zusätzliche J1850 VPW Schnittstelle
- ◆ 1x Zusätzliche J1850 PWM Schnittstelle



Wenn Sie die Option 1x Zusätzliche J1850 PWM Schnittstelle wählen, ist die Option 1x Zusätzliche LIN Schnittstelle oder K-Line Schnittstelle NICHT möglich.

Außer der Auswahl der Schnittstelle selbst muss auch der Typ des zugehörigen CAN/ LIN/ K-Line/ J1850 Transceivers für jede Schnittstelle festgelegt werden.

Für jede CAN/ K-Line-Schnittstelle sind außerdem die erforderlichen Funktionalitäten anzugeben.

3 Ansteuersoftware

Zur Einbindung der basicCAR 3085-Hardware in eigene Applikationen existieren drei Möglichkeiten:

- ♦ [Programmieren über G-API](#)
- ♦ [Programmieren über DLL-Funktionen](#)
- ♦ [Programmieren mit LabVIEW](#)

3.1 Programmieren über G-API

Das bevorzugte User Interface für diese GÖPEL Hardware ist die G-API (GÖPEL-API).

Sie finden alle benötigten Informationen im Ordner *G-API* der mitgelieferten CD.

3.2 Programmieren über DLL-Funktionen



Die Programmierung über DLL-Funktionen ist weiterhin für bestehende Projekte möglich, bei denen noch nicht mit der GÖPEL G-API gearbeitet werden kann.

Die Dokumentation *GÖPEL Firmware* senden wir Ihnen auf Anforderung gern zu. Bitte setzen Sie sich bei Bedarf mit unserem Vertrieb in Verbindung.



Der in der folgenden Funktionsbeschreibung verwendete Begriff *GUSB_Platform* ist der Name eines USB Treibers der GÖPEL electronic GmbH.

Informationen zu den Strukturen, Datentypen und Error-Codes enthalten die Header – die entsprechenden Dateien finden Sie auf der mitgelieferten CD.

3.2.1 Windows Device Treiber

Die für die Programmierung unter Verwendung des Windows Device Treibers nutzbaren DLL-Funktionen sind in den folgenden Abschnitten beschrieben:

- ◆ [Driver_Info](#)
- ◆ [DLL_Info](#)
- ◆ [Write_FIFO](#)
- ◆ [Read_FIFO](#)
- ◆ [Read_FIFO_Timeout](#)
- ◆ [Write_COMMAND](#)
- ◆ [Read_COMMAND](#)
- ◆ [Xilinx_Download](#)
- ◆ [Xilinx_Version](#)

3.2.1.1 Driver_Info

Die Funktion `GUSB_Platform_Driver_Info` dient zur Status-Abfrage des Hardware-Treibers und zur internen Initialisierung der erforderlichen Handles.



Diese Funktion MUSS einmalig vor dem Aufruf aller anderen Funktionen des `GUSB_Platform` Treibers ausgeführt werden.

Format:

```
int GUSB_Platform_Driver_Info(GUSB_Platform_DriverInfo *pDriverInfo,
                             unsigned int LengthInByte)
```

Parameter:

Zeiger, z.B. `pDriverInfo`
auf eine Datenstruktur

Zur Struktur siehe das File `GUSB_Platform.h` auf der mitgelieferten CD

`LengthInByte`

Größe des Speicherbereiches, auf den `pDriverInfo` zeigt, in Bytes

Beschreibung:

Die Funktion `GUSB_Platform_Driver_Info` gibt Informationen über den Status des Hardware-Treibers zurück.

Dazu muss der Funktion die Adresse des Zeigers `pDriverInfo` übergeben werden. Mit Hilfe des Parameters `LengthInByte` prüft die Funktion intern den korrekt initialisierten Anwenderspeicher.

Die Funktion füllt die Struktur, auf die `pDriverInfo` zeigt, mit Angaben zur Treiberversion, der Anzahl aller sich im System befindenden **USB Controller** (die von diesem Treiber unterstützt werden), und Informationen darüber, wie z.B. die Seriennummer(n).



Die Bereitstellung der Hardwareinformationen und die Initialisierung der zugehörigen Handles ist für die weitere Nutzung der USB-Hardware zwingend erforderlich.

3.2.1.2 DLL_Info Die Funktion `GUSB_Platform_DLL_Info` dient zur Abfrage von Informationen über die DLL.

Format:

```
int GUSB_Platform_DLL_Info(GUSB_Platform_DLLInfo *DLLinformation)
```

Parameter

Zeiger, z.B. `DLLinformation`
auf eine Datenstruktur
Zur Struktur siehe das File `GUSB_Platform.h` auf der mitgelieferten CD

Beschreibung:

Die Funktion `GUSB_Platform_DLL_Info` gibt die Struktur `DLLInfo` zurück. Der erste Integerwert enthält die Versionsnummer der `GUSB_Platform.dll`.

Beispiel:

Die Versionsnummer `1.23` wird als Wert `123` zurückgegeben,
Version `1.60` als Wert `160`.

3.2.1.3 Write_FIFO Die Funktion `GUSB_Platform_Write_FIFO` dient zum Senden eines Befehls zum Controller.

Format:

```
int GUSB_Platform_Write_FIFO(unsigned int DeviceName,  
                             unsigned int DeviceNumber,  
                             t_USB_FIFO_Interface_Buffer *pWrite,  
                             unsigned int DataLength)
```

Parameter:

DeviceName

Typ des adressierten Gerätes (Nummer, die in `GUSB_Platform_def.h` deklariert ist, für `basicCAR 3085` = 28)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer `DeviceNumber` 1).

Zeiger, z.B. `pWrite`
auf den Bereich für Schreibdaten

DataLength

Größe des Speicherbereiches, auf den `pWrite` zeigt, in Bytes
Die Daten bestehen aus **Befehlskopf** und **Befehlsbytes**
(z. Zt. max. 1024 Byte pro Befehl)

Beschreibung:

Die Funktion `GUSB_Platform_Write_FIFO` sendet einen Befehl zum Controller.

Die allgemeine Befehlsstruktur ist im Abschnitt **Allgemeines zur Firmware** der Dokumentation **GÖPEL Firmware** beschrieben.

3.2.1.4 Read_FIFO Die Funktion `GUSB_Platform_Read_FIFO` dient zum Lesen einer Antwort vom Controller.

Format:

```
int GUSB_Platform_Read_FIFO(unsigned int DeviceName,  
                           unsigned int DeviceNumber,  
                           t_USB_FIFO_Interface_Buffer *pRead,  
                           unsigned int *DataLength)
```

Parameter:

DeviceName

Typ des adressierten Gerätes (Nummer, die in `GUSB_Platform_def.h` deklariert ist, für `basicCAR 3085 = 28`)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer `DeviceNumber 1`).

Zeiger, z.B. pRead

auf den Lesebuffer

Nach erfolgreicher Funktionsausführung befinden sich die Daten im Lesebuffer, bestehend aus `Antwortkopf` und `Antwortbytes` (z. Zt. max. 1024 Byte pro Antwort)

DataLength

Vor Funktionsaufruf: Anzugebende Größe des Lesebuffers in Bytes
Nach Funktionsausführung: Anzahl der tatsächlich gelesenen Bytes

Beschreibung:

Die Funktion `GUSB_Platform_Read_FIFO` liest die älteste vom Controller geschriebene Antwort. Ist während einer `Timeout`-Zeit von 100 ms (nicht einstellbar) keine Antwort empfangen worden, liefert die Funktion jedoch KEINEN Fehler zurück: In diesem Fall ist der Wert für die `Anzahl der tatsächlich gelesenen Bytes = 0 !!!`

3.2.1.5 Read_FIFO_Timeout Die Funktion `GUSB_Platform_Read_FIFO_Timeout` dient zum Lesen einer Antwort vom Controller, wobei ein Timeout vorzugeben ist.

Format:

```
int GUSB_Platform_Read_FIFO_Timeout(unsigned int DeviceName,
                                     unsigned int DeviceNumber,
                                     t_USB_FIFO_Interface_Buffer *pRead,
                                     unsigned int *DataLength,
                                     unsigned int Timeout)
```

Parameter:

DeviceName

Typ des adressierten Gerätes (Nummer, die in `GUSB_Platform_def.h` deklariert ist, für `basicCAR 3085 = 28`)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer DeviceNumber 1).

**Zeiger, z.B. pRead
auf den Lesebuffer**

Nach erfolgreicher Funktionsausführung befinden sich die Daten im Lesebuffer, bestehend aus Antwortkopf und Antwortbytes (z. Zt. max. 1024 Byte pro Antwort)

DataLength

Vor Funktionsaufruf: Anzugebende Größe des Lesebuffers in Bytes
Nach Funktionsausführung: Anzahl der tatsächlich gelesenen Bytes

Timeout

Angabe in Millisekunden (Standardwert: 500)

Beschreibung:

Die Funktion `GUSB_Platform_Read_FIFO_Timeout` liest die älteste vom Controller geschriebene Antwort.

Ist während der einstellbaren Timeout-Zeit keine Antwort empfangen worden, liefert die Funktion jedoch KEINEN Fehler zurück: In diesem Fall ist der Wert für die Anzahl der tatsächlich gelesenen Bytes = 0 !!!

3.2.1.6 Write_COMMAND

Die Funktion `GUSB_Platform_Write_COMMAND` dient zum Senden eines Configuration-Befehls zum USB Controller.

Format:

```
int GUSB_Platform_Write_COMMAND(unsigned int DeviceName,  
                                unsigned int DeviceNumber,  
                                t_USB_COMMAND_Interface_Buffer *pWrite,  
                                unsigned int DataLength)
```

Parameter:

DeviceName

Typ des adressierten Gerätes (Nummer, die in `GUSB_Platform_def.h` deklariert ist, für `basicCAR 3085` = 28)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer `DeviceNumber` 1).

Zeiger, z.B. `pWrite`
auf den Bereich für Schreibdaten

DataLength

Größe des Speicherbereiches, auf den `pWrite` zeigt, in Bytes
Siehe auch [Steuerbefehle USB Controller](#)
(z. Zt. max. 64 Byte pro Befehl)

Beschreibung:

Die Funktion `GUSB_Platform_Write_COMMAND` sendet einen Befehl zum USB Controller.

Die allgemeine Struktur ist im Abschnitt [Steuerbefehle USB Controller](#) beschrieben.

3.2.1.7 Read_COMMAND Die Funktion `GUSB_Platform_Read_COMMAND` dient zum Lesen einer Antwort vom USB Controller.

Format:

```
int GUSB_Platform_Read_COMMAND(unsigned int DeviceName,  
                               unsigned int DeviceNumber,  
                               t_USB_COMMAND_Interface_Buffer *pRead,  
                               unsigned int *DataLength)
```

Parameter:

DeviceName

Typ des adressierten Gerätes (Nummer, die in `GUSB_Platform_def.h` deklariert ist, für `basicCAR 3085` = 28)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer `DeviceNumber` 1).

Zeiger, z.B. `pRead`
auf den Lesebuffer

Nach erfolgreicher Funktionsausführung befinden sich die Daten im Lesebuffer, bestehend aus `Antwortkopf` und `Antwortbytes`

Siehe auch [Steuerbefehle USB Controller](#)
(z. Zt. min. 64 Byte pro Antwort)

DataLength

Vor Funktionsaufruf: Anzugebende Größe des Lesepuffers in Bytes
Nach Funktionsausführung: Anzahl der tatsächlich gelesenen Bytes

Beschreibung:

Die Funktion `GUSB_Platform_Read_COMMAND` liest die älteste vom USB Controller geschriebene Antwort zurück.

Werden mehrere Antworten vom USB Controller bereitgestellt, werden maximal zwei dieser Antworten in den Puffer des USB Controllers geschrieben.

Weitere ggf. bereitgestellte Antworten gehen verloren!

3.2.1.8 Xilinx_ Download

Die Funktion `GUSB_Platform_Xilinx_Download` dient zum Laden eines FPGA-Files in den XILINX.

Format:

```
int GUSB_Platform_Xilinx_Download(unsigned int DeviceName,  
                                unsigned int DeviceNumber,  
                                char *pFileName,  
                                unsigned char *pFirmwareErrorCode)
```

Parameter:

DeviceName

Typ des adressierten Gerätes (Nummer, die in `GUSB_Platform_def.h` deklariert ist, für `basicCAR 3085` = 28)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer `DeviceNumber` 1).

pFileName

Pfad des zu ladenden FPGA-Files

pFirmwareErrorCode

Fehlercode, der während der Abarbeitung dieser DLL-Funktion auftritt (bei Fehlercode 0 ist kein Fehler aufgetreten)
(error codes -> card firmware siehe `GUSB_Platform_def.h`)

Beschreibung:

Die Funktion `GUSB_Platform_Xilinx_Download` dient zum Laden eines FPGA-Files in den XILINX (Extension `*.cdf`).

Die geladenen Daten sind flüchtig. Deshalb muss die Funktion nach Power Off erneut ausgeführt werden.



Nach `Xilinx_Download` ist eine Wartezeit von ca. 500 ms erforderlich, da der Controller ein Power-On-Reset durchläuft.

Anschließend ist der Firmwarebefehl `0x10 Software Reset` auszuführen, um vom Bootloader-Modus in den Normal-Modus zu gelangen.

- 3.2.1.9 Xilinx_Version** Die Funktion `GUSB_Platform_Xilinx_Version` ermöglicht das Auslesen der geladenen XILINX-Firmwareversion.

Format:

```
int GUSB_Platform_Xilinx_Version(unsigned int DeviceName,  
                                unsigned int DeviceNumber,  
                                unsigned int *Version)
```

Parameter:**DeviceName**

Typ des adressierten Gerätes (Nummer, die in `GUSB_Platform_def.h` deklariert ist, für `basicCAR 3085 = 28`)

DeviceNumber

Nummer des adressierten Gerätes. Wenn mehrere Geräte gleichen Typs angeschlossen sind, erfolgt die Nummerierung in aufsteigender Reihenfolge der Seriennummern (das Gerät mit der NIEDRIGSTEN Seriennummer hat immer `DeviceNumber 1`).

Version

XILINX Softwareversion

Beschreibung:

Mit der Funktion `GUSB_Platform_Xilinx_Version` kann die Versionsnummer der im FPGA geladenen Software ausgelesen werden.

Beispiel:

Die Versionsnummer `2.34` wird als Wert `234` zurückgegeben, Version `2.60` als Wert `260`.

3.3 Programmieren mit LabVIEW

3.3.1 LabVIEW über G-API

Auf der mitgelieferten CD befindet sich eine VI-Sammlung, mit deren Hilfe basicCAR 3085-Baugruppen unter LabVIEW angesprochen werden können.

Dabei nutzen die LabVIEW VIs die Funktionen der GÖPEL G-API.

3.3.2 LLB unter Verwendung des Windows Device Treibers

Auf der mitgelieferten CD befindet sich eine VI-Sammlung, mit deren Hilfe basicCAR 3085-Baugruppen unter LabVIEW angesprochen werden können.

Dabei werden die Funktionen genutzt, die im Abschnitt [Windows Device Treiber](#) beschrieben worden sind.

3.4 Weitere GÖPEL Software

PROGRESS, Programm Generator und myCAR der GÖPEL electronic GmbH sind komfortable Programme zur Prüfung mit GÖPEL-Hardware.

Weitere Informationen zur Nutzung dieser Programme finden Sie in den entsprechenden Softwarebeschreibungen.

3.5 Steuerbefehle USB Controller

Der USB Controller ist für die Anbindung der basicCAR 3085-Baugruppe an den PC über USB 2.0 zuständig.

An diesen USB Controller können Nachrichten (i. Allg. USB Befehle) gesendet werden, die für Konfigurationszwecke benötigt werden.

3.5.1 USB Befehlsaufbau

Ein USB Befehl besteht aus vier Bytes Header und den Daten (nicht alle USB Befehle benötigen Daten!).

Der Header eines USB Befehls ist folgendermaßen aufgebaut:

Bytenummer	Bedeutung	Inhalt
0	StartByte	0x23 (ASCII-Zeichen „#“)
1	Command	(0x..) Verwendete Codes entsprechend USB Befehle
2	reserviert	0x00
3	reserviert	0x00

3.5.2 USB Antwortaufbau

Genau wie der USB Befehl, ist auch die USB Antwort in vier Bytes Header und die Daten unterteilt (nicht alle USB Befehle senden Daten zurück!).

Der Header einer USB Antwort ist folgendermaßen aufgebaut:

Bytenummer	Bedeutung	Inhalt
0	StartByte	0x24
1	Command	(0x..) Verwendete Codes entsprechend USB Befehle
2	Length	Vom Befehl abhängige Länge
3	ErrorCode	Gibt den Fehlercode des Befehls zurück

3.5.3 USB Befehle

Gegenwärtig steht nur der USB Befehl READ_SW_VERSION zur Verfügung.

Command	Bezeichnung	Bedeutung
0x04	READ_SW_VERSION	Liefert die Version des USB Controllers Antwort: Byte 4: low Byte generic Softwareversion Byte 5: high Byte generic Softwareversion Byte 6: low Byte Softwareversion des funktionellen Teiles Byte 7: high Byte Softwareversion des funktionellen Teiles

B

basicCAR 3085	
Hardwareinstallation.....	1-1
Ressourcen.....	2-2
Treibereinstallation.....	1-2

C

Controller	
Antwort	3-6
Befehl	3-5

F

Firmware	3-1
----------------	-----

G

G-API	3-1
-------------	-----

L

LabVIEW	
G-API	3-12
Windows	3-12

S

Steckverbinder	
Front	2-10

T

Transceiver	
CAN	2-6
J1850.....	2-8
K-Line	2-7
LIN	2-7

U

USB Antwortaufbau	3-13
USB Befehle	3-13
USB Befehlsaufbau	3-13
USB Controller	
Antwort	3-9
Befehl	3-8
Steuerbefehle	3-13

W

Windows Treiber	3-2
-----------------------	-----