Produktbeschreibung

smartCAR

mit erweitertem Befehlsumfang
USB 2.0 Stand-alone Gerät für
CAN, LIN oder K-LINE Interface

Nutzerhandbuch Version 1.2

GÖPEL electronic

Get the total Coverage!

GÖPEL electronic GmbH
Göschwitzer Str. 58/60
D-07745 Jena
Tel.: +49-3641-6896-597
Fax: +49-3641-6896-944
E-Mail: ats_support@goepel.com
http://www.goepel.com

Gedruckt: 05.12.2011

Stand: Dezember 2011

1 Installation

1.1 Hardwareinstallation

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

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

Wenn es notwendig ist, Transceivermodule zu tauschen, ist kein Eingriff in das Gerät erforderlich (siehe Wechsel des Transceivermoduls).

Beachten Sie bitte die allgemeinen Regeln zur Vermeidung von elektrostatischen Entladungen.

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.

Ihr smartCAR kann unter Windows® 2000/ XP sowie unter Windows®7/ 32 Bit und Windows®7/ 64 Bit betrieben werden.

Wenn Sie eigene Software für ein smartCAR 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 das Gerät einwandfrei vom System eingebunden worden ist:

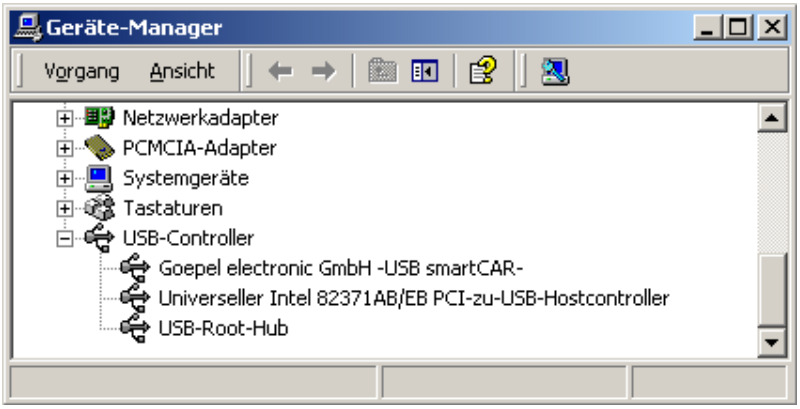

Abbildung 1-1: Anzeige Geräte-Manager

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

2 Hardware

2.1 Bestimmung

smartCAR ist ein Stand-alone-Gerät der GÖPEL electronic GmbH mit USB 2.0-Interface zum Anschluss an einen PC oder Laptop, das speziell für den eigenständigen Einsatz außerhalb komplexer Testsysteme bzw. in Werkstätten entwickelt wurde.

Abbildung 2-1: smartCAR

smartCAR bietet folgende Ressourcen:

- 1 x CAN oder 1x LIN oder 1x K-Line
- 32Bit µController onBoard
- USB 2.0 Schnittstelle
- Spannungsversorgung wahlweise über USB-Schnittstelle oder extern möglich
- Hohe Flexibilität durch austauschbare Transceivermodule

Bitte beachten Sie, dass Ihr smartCAR über KEINE galvanische Trennung zwischen USB- und Anwenderinterface verfügt.
Deshalb muss sichergestellt werden, dass das Prüfobjekt und alle anderen mit dem smartCAR verbundenen Geräte entweder über **isolierte Netzteile** gespeist werden oder alle beteiligten Geräte sternförmig an das **gleiche Massepotenzial** angeschlossen werden.

2.2 Technische Daten

2.2.1 Abmessungen

Ihr smartCAR hat folgende Abmessungen (Breite x Höhe x Tiefe):

- 75 mm x 25 mm x 110 mm

2.2.2 Kennwerte

Ein smartCAR hat folgende Kennwerte:

Symbol	Kennwert	Min.	Typ.	Max.	Einheit	Bemerkung
U_{BAT}	Versorgungsspannung		8	27	V	Stromversorgung extern oder über USB-Schnittstelle
	Übertragungsrate			1	MBaud	Für CAN oder
	Übertragungsrate			22	kBaud	Für LIN oder
	Übertragungsrate			150	kBaud	Für K-Line
R_{bus}	Abschlusswiderstand		120		Ohm	Für CAN oder
R_{Pullup}	Pullup-Widerstand		1000		Ohm	Für K-Line

2.3 Aufbau

2.3.1 Allgemeines

Abbildung 2-2 verdeutlicht den prinzipiellen Aufbau des smartCAR.

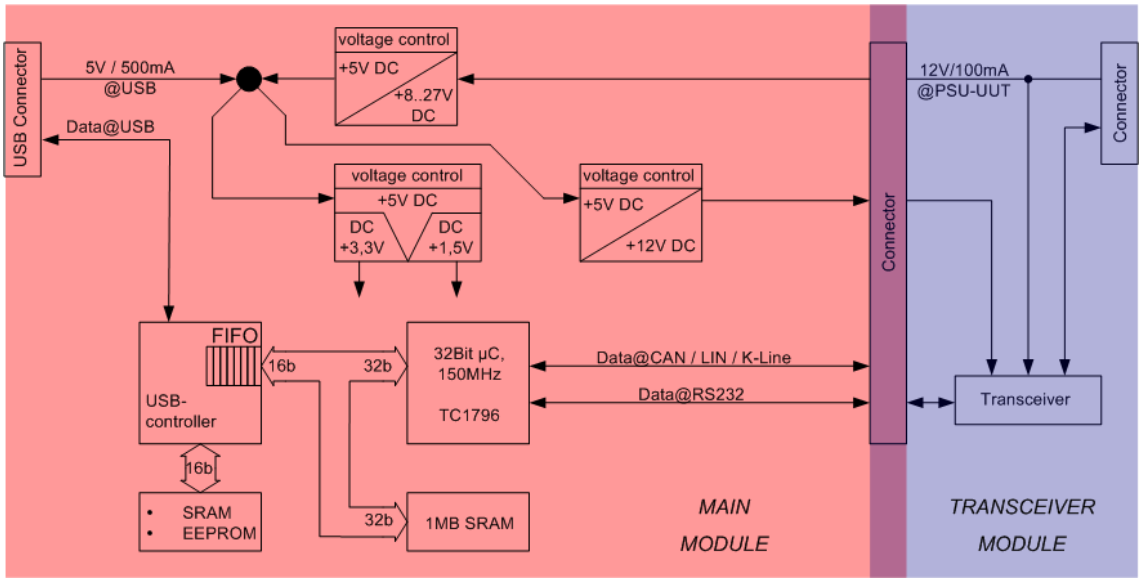

Abbildung 2-2: Blockdiagramm einer smartCAR-Baugruppe

Warnung

Bitte verwenden Sie zum Anschluss Ihres smartCAR an die USB-Schnittstelle des PCs nur das im Lieferumfang enthaltene USB-Kabel. Andere Kabel sind u. U. nicht geeignet!

2.3.2 Adressierung

Bei Verwendung mehrerer smartCAR an einem Rechner erfolgt die Adressierung der smartCAR-Baugruppen ausschließlich über deren Seriennummern (siehe Ansteuersoftware):
Die Baugruppe mit der KLEINSTEN Seriennummer ist immer das Gerät Nummer 1.

Tip

Um die Übersichtlichkeit zu verbessern empfehlen wir, die einzelnen smartCAR Geräte in aufsteigender Reihenfolge ihrer Seriennummern am PC anzuschließen.

2.3.3 Wechsel des Transceivermoduls

Abbildung 2-3 verdeutlicht die mechanische Verbindung zwischen Haupt- und Transceivermodul des smartCAR.

Zum Wechseln ist das Transceivermodul durch vertikales Ziehen nach unten vom Hauptmodul zu trennen.

Abbildung 2-3 Wechsel des Transceivermoduls

2.3.4 Kommunikationsschnittstellen

CAN-Interface Version 2.0b:

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 smartCAR-Nutzer 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 (siehe Abbildung 2-3).

U_{bat} ist der interne Anschluss für die Versorgungsspannung der Transceivermodule.

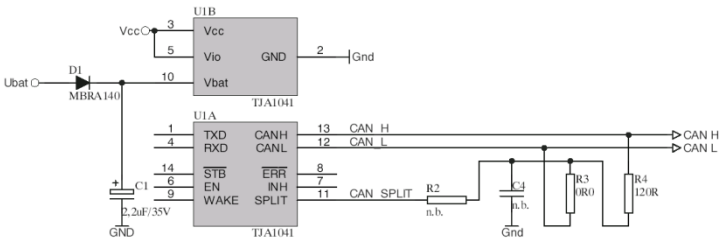

Abbildung 2-4: CAN Schnittstelle

K-Line Interface (ISO 9141):

Der Transceiver ist als steckbares Modul ausgeführt. I. Allg. wird der L9637 der Fa. ST für diesen Transceiver verwendet.

U_{bat} ist der interne Anschluss für die Versorgungsspannung des Transceivermoduls.

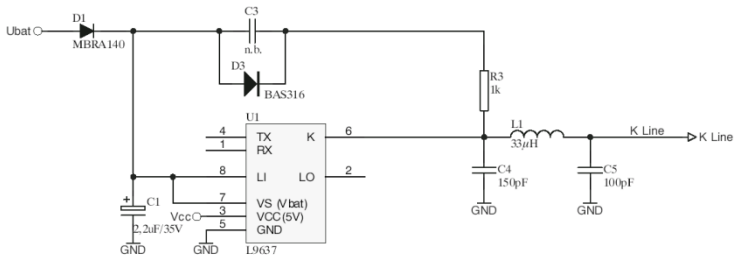

Abbildung 2-5
K-Line Schnittstelle

LIN-Interface Version 2.0:

Der Transceiver ist als steckbares Modul ausgeführt. I. Allg. wird der TJA1020 der Fa. Philips für diesen Transceiver verwendet.

Das Transceivermodul kann per Software über das Relais mit der Nummer 2 zwischen Master- und Slave-Konfiguration umgeschaltet werden.

U_{bat} ist der interne Anschluss für die Versorgungsspannung des Transceivermoduls.

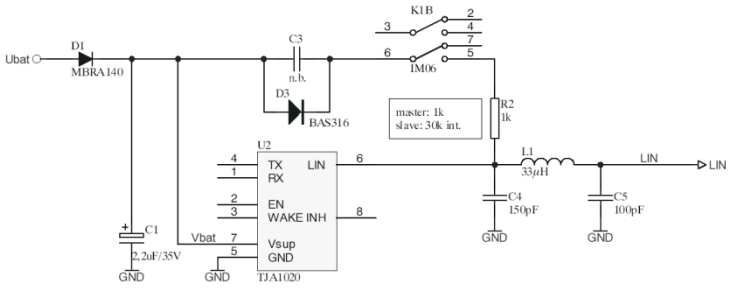

Abbildung 2-6:
LIN Schnittstelle

2.3.5 Belegung der Steckverbinder

Auf die Signale der Kommunikationsschnittstelle können Sie direkt über die RJ45-Buchse des smartCAR oder über den DSub-Steckverbinder des mitgelieferten Kabels zugreifen.

Kommunikationsschnittstelle

Typ: RJ45 Buchse

Die Belegung ist in der folgenden Tabelle dargestellt:

Pin	CAN	K-Line/LIN
1	U_{bat}	U_{bat}
2	n.c.	n.c.
3	CAN-High	n.c.
4	CAN-Low	n.c.
5	n.c.	K-Line/ LIN
6	n.c.	n.c.
7	n.c.	n.c.
8	GND	GND

Typ: DSub 9-polig Stecker (am Kabel)

Die Belegung ist in der folgenden Tabelle dargestellt:

Pin	CAN	K-Line/LIN
1	n.c.	n.c.
2	CAN-Low	n.c.
3	GND	GND
4	n.c.	n.c.
5	n.c.	n.c.
6	n.c.	n.c.
7	CAN-High	K-Line/ LIN
8	n.c.	n.c.
9	U_{bat}	U_{bat}

USB-Schnittstelle

An der Rückseite des smartCAR befindet sich die miniUSB-Buchse (USB-Standardbelegung) für das USB 2.0 Interface.

2.4 Lieferhinweise

Eine smartCAR Lieferung enthält mindestens

- 1x smartCAR Hauptmodul und 1x smartCAR Transceivermodul

Folgende smartCAR Transceivermodule sind derzeit verfügbar:

- 1x TJA1041 CAN Highspeed
- 1x TJA1054 CAN Lowspeed
- 1x AU5790 CAN Single Wire
- 1x L9637 K-Line
- 1x TJA 1020 LIN

Bitte geben Sie bei der smartCAR-Bestellung auch den Typ des erforderlichen Transceivers an.

Lediglich durch den Austausch des Transceivermoduls (Abbildung 2-3) können Sie bestimmen, ob die smartCAR-Hardwareschnittstelle als CAN, LIN oder K-Line Schnittstelle arbeitet.

3 Ansteuersoftware

Zur Einbindung der smartCAR-Hardware in Ihre eigenen 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 setzten Sie sich bei Bedarf mit unserem Vertrieb in Verbindung.

Der Begriff GUSB_Platform in der folgenden Funktionsbeschreibung steht jeweils für eine smartCAR-Baugruppe.

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

In diesem Nutzerhandbuch ist unter Controller IMMER der der CAN, LIN oder K-Line-Schnittstelle zugeordnete Microcontroller einer smartCAR-Baugruppe zu verstehen.

Unter USB-Controller wird in diesem Nutzerhandbuch IMMER der Controller verstanden, der das USB 2.0-Interface auf der smartCAR-Baugruppe bereitstellt.

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

3.2.1.1 *Driver_Info*

Die Funktion GUSB_Platform_Driver_Info dient zur Status-Abfrage des Hardware-Treibers und zur internen Initalisierung 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 smartCAR = 13)

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 smartCAR = 13)

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 Lesepuffer
Nach erfolgreicher Funktionsausführung befinden sich die Daten im Lesepuffer, bestehend aus Antwortkopf und Antwortbytes
(z. Zt. max. 1024 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_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 smartCAR = 13)

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

DataLength

Vor Funktionsaufruf: Anzugebende Größe des Lesepuffers 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 smartCAR = 13)

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 smartCAR = 13)

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 Lesepuffer
Nach erfolgreicher Funktionsausführung befinden sich die Daten im Lesepuffer, 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!

3.3 Programmieren mit LabVIEW

3.3.1 LabVIEW über G-API

Auf der mitgelieferten CD befindet sich eine VI-Sammlung, mit deren Hilfe smartCAR-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 smartCAR-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 smartCAR 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