Product Specification

basicCAR 3085

USB 2.0 stand-alone Device for CAN/ LIN/ K-LINE/ J1850 Interfaces

User Manual Version 1.1

GOEPEL electronic

Get the total Coverage!

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

© 2010 GOEPEL electronic GmbH. All rights reserved.

The software described in this manual as well as the manual itself are supplied under license and may be used or copied only in accordance with the terms of the license.
The customer may make one copy of the software for safety purposes.

The contents of the manual is subject to change without prior notice and is supplied for information only.
Hardware and software might be modified also without prior notice due to technical progress.

In case of inaccuracies or errors appearing in this manual, GOEPEL electronic GmbH assumes no liability or responsibility.

Without the prior written permission of GOEPEL electronic GmbH, no part of this documentation may be transmitted, reproduced or stored in a retrieval system in any form or by any means as well as translated into other languages (except as permitted by the license).

GOEPEL electronic GmbH is neither liable for direct damages nor consequential damages from the company's product applications.

printed: 09.06.2010

All product and company names appearing in this manual are trade names or registered trade names of their respective owners.

Issue: June 2010

1 Installation

1.1 Hardware Installation

Generally hardware installation for basicCAR 3085 means exchanging the transceiver modules.

Please make absolutely certain that all of the installation procedures described below are carried out with your system switched off.

If it is necessary to exchange transceiver modules, the corresponding device is to be opened according to its conditions.

Doing this, pay attention to the general rules to avoid electrostatic charging.
Transceiver modules must never be removed or mounted with the power switched on!
In addition, the right alignment is absolutely required (see Assembly).

1.2 Driver Installation

For proper installation of the GOEPEL electronic USB drivers on your system, we recommend to execute the GUSB driver setup.
To do that, start the *GUSB-Setup-*.exe* setup program (of the supplied CD, "*" stands for the version number) and follow the instructions.

At present, the available device drivers only support Windows® 2000/ XP systems.

If you want to create your own software for basicCAR 3085 devices, you possibly need additional files for user specific programming (*.LLB*, *.H*). These files are not automatically copied to the computer and have to be transferred individually from the supplied CD to your development directory.

The USB interface uses the high-speed data rate according to the USB2.0 specification (if possible, otherwise full-speed).

After driver installation, you can check whether the devices are properly embedded by the system.

The following picture shows the successful embedding of four basicCAR 3085 devices:

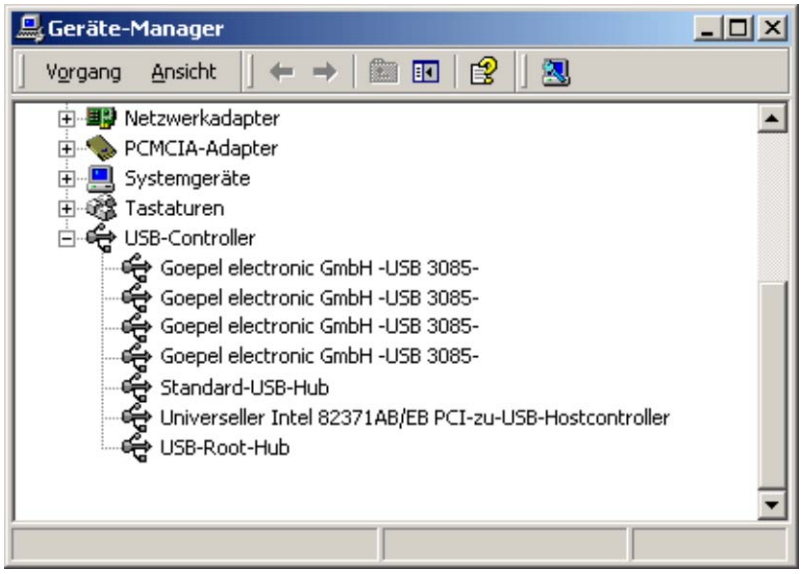

Figure 1-1: Display of Device Manager

Please note that the Device Manager shows ALL USB controllers.

2 Hardware

2.1 Definition

basicCAR 3085 is a GOEPEL electronic GmbH stand-alone device to be connected to a PC or laptop. It was in particular developed for applications out of complex test systems.
Compared with USB 3080/ basicCAR 3080, a basicCAR 3085 has connectors to distribute and control the operating voltage of peripheral devices.

The external power supply allows the use of this device for data acquisition and the inspection of signals for a multitude of applications, for example in motor vehicles.

Figure 2-1: basicCAR 3085

Please note: Downloading the Xilinx FPGA is absolutely required for operating the basicCAR 3085 device (see Xilinx_Download in the Windows Device Driver section)!

Power supply with 8..25 VDC (and approx. 350 mA at 12 V) is effected via the ext. Power basicCAR connector at the device's rear side (opposite to the communication interfaces side):

Figure 2-2: Rear side

This connector is used to supply the internal logic of the device. It's ground connection is joined to the ground connection of the USB interface.

The two input females for the ext Power Front peripheral voltage supply (red = plus/ black = minus) are arranged to the right of the ext. Power basicCAR connector.

All connections of the communication interfaces as well as the peripheral voltage supply are galvanically isolated from the USB interface and the internal logic.

The USB connector is also arranged at the device's rear side.

In the maximum construction stage, basicCAR 3085 devices offer the following resources:

- 2 x CAN
- 2 x LIN or K-Line
- 1 x J1850 VPW
- 1 x J1850 PWM
  (in this case only ONE LIN or K-Line interface is possible)
- 4 x digital input
- 4 x digital output
- 2 x analog input
- 1 x Wake line
- 1 x peripheral voltage output (2 poles)
- 3x disengageable peripheral voltage outputs (1 pole)

Communication for J1850 PWM is made via the LIN2/ K-Line interface!!!

2.2 Technical Specification

2.2.1 Dimensions

The board dimensions correspond to the standard dimensions of the accompanying bus system (width x height x depth):

- basicCAR 3085: 145 mm x 70 mm x 220 mm

2.2.2 Properties

The characteristics of basicCAR 3085 are as follows:

Symbol	Parameter	Min.	Typ.	Max.	Unit	Remarks
V_{EXT}	External power supply	8	12	25	V_{DC}	
V_{BAT}	Battery voltage		12	27	V_{DC}	Acc. to transceiver's type
V_{PER_IN}	Peripheral voltage input			30	V_{DC}	
I_{PER_IN}	Peripheral current input			10	A	
I_{PER_OUT}	Peripheral current output			5 (10)	A	10A only for the NOT disengageable output
	Transmission rate			1	MBaud	CAN
	Transmission rate			22	kBaud	LIN
R_{bus}	Terminating resistor 1		120		Ohm	CAN jumper plugged in
R_{bus}	Terminating resistors 2		10		kOhm	CAN jumper plugged in
R_{Pullup}	Pull-up resistor		680		Ohm	K-Line jumper plugged in
V_{in}	Input voltage	3.3		50	V_{DC}	Digital input
V_{out}	Output voltage			V_{BAT}	V_{DC}	Digital output, OC
V_{in}	Input voltage		20	26	V_{DC}	Analog input
V_{iso}	Galvanic separation	560			V_{DC}	USB In-/ output

The Analog input channels are designed with the LTC 1400 (analog-to-digital transducer) of Linear Technology.
This component has a Resolution of 12 Bit and an Input voltage range of 0..4.095V.

Caused by the input voltage divider (122K/22K) the following results for the measured voltage:
V_{meas} = AD transducer value * 1mV * (122K/22K).

Warning

Please make absolutely certain to supply the transceivers via the V_{BAT} battery voltage connections with the LOWEST maximum voltage of all transceivers.

2.3 Construction

2.3.1 General

An ASIC (TC1775) is used as the interface to the USB bus for basicCAR 3085. It includes all the function blocks required for the communication with the computer bus.

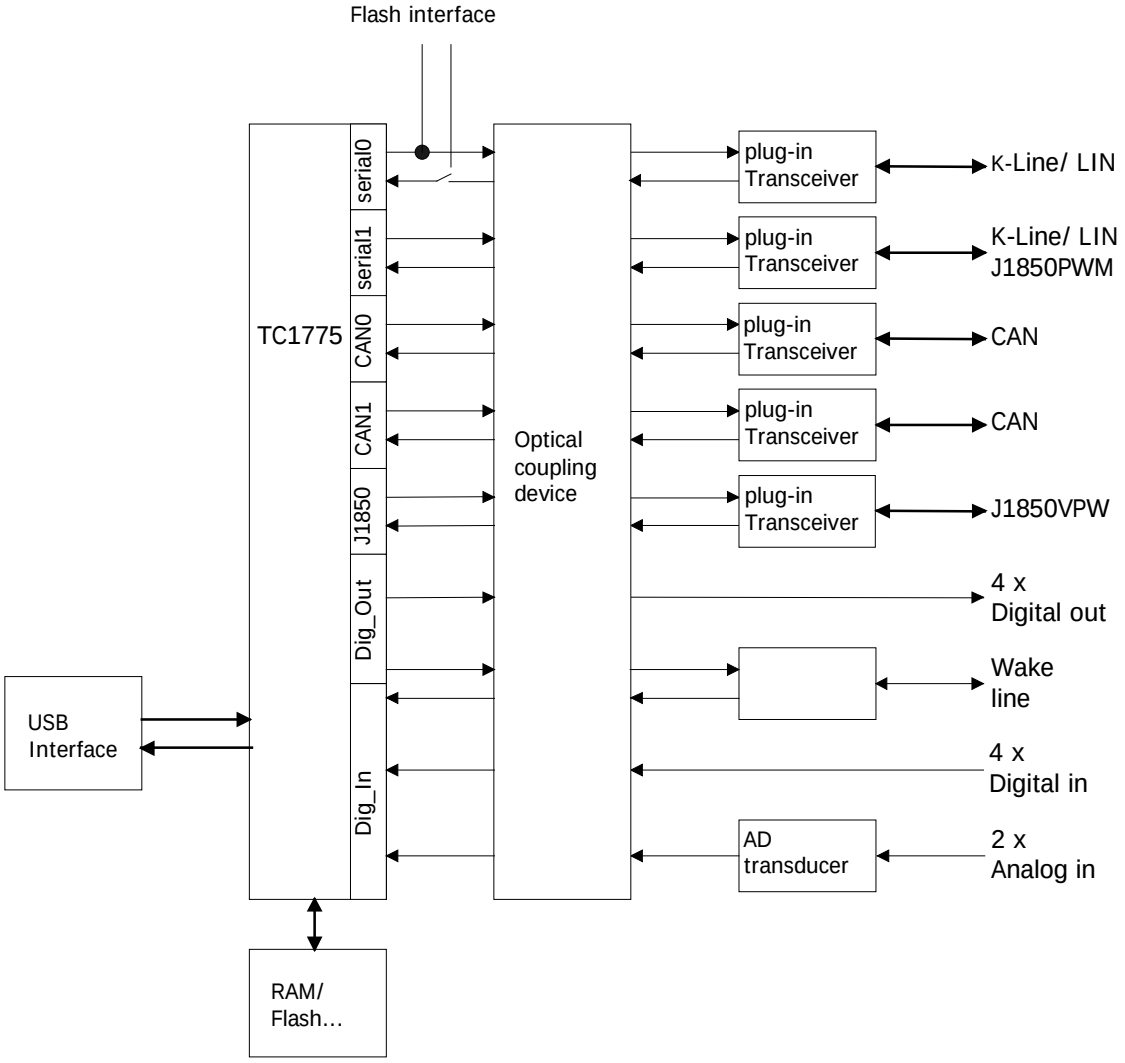

Figure 2-3: Block diagram of the communication board for basicCAR 3085

Please use the delivered USB cables to connect basicCAR 3085 devices to the PC's USB interface.
Other cables may be inapplicable.

2.3.2 Relays

Your basicCAR 3085 device is populated with five relays used for the master/ slave changeover of the LIN interfaces as well as for switching the peripheral voltages.

Take the 0x81…0x84 Firmware commands of the Digital Commands to switch the relays (see the GOEPEL Firmware documentation). Please use the relay numbers according to the following table for these commands:

Relay	Use
1	LIN1 master/ slave configuration see Communication Interfaces/ 2 x LIN-Interface
2	LIN2 master/ slave configuration see Communication Interfaces/ 2 x LIN-Interface
3	Rel1 external relay output
4	Rel2 external relay output
5	Rel3 external relay output

The 0x81…0x84 LIN Commands can also be used for the master/ slave configuration of the LIN interfaces
(see also the GOEPEL Firmware documentation).

2.3.3 Communication Interfaces

2 x CAN Interface:

The type of the mounted transceiver is decisive for proper operation of a CAN interface in a network. Often CAN networks do only operate properly in the case that all members use a compatible type of transceiver.

To offer maximal flexibility to the users of a basicCAR 3085, the transceivers are designed as plug-in modules. There are several types (high speed, low speed, single-wire etc.) that can be easily exchanged.

Not only the type of the mounted transceiver, but also the terminating resistor of the bus is very important for proper operation of a CAN network.

For the use of highspeed CAN transceivers, usually one 120 Ohm resistor is active on each CAN interface.These resistors can be deactivated by removing the J1401 or J1501 jumpers.

In the case of lowspeed CAN transceivers, usually two resistors with a resistance value of 10 kOhm for RTH and RTL are active for each CAN interface (by inserting the J1402/ J1403 or J1502/ J1503 jumpers).
Then the J1401 or J1501 jumpers must NOT be plugged-in.

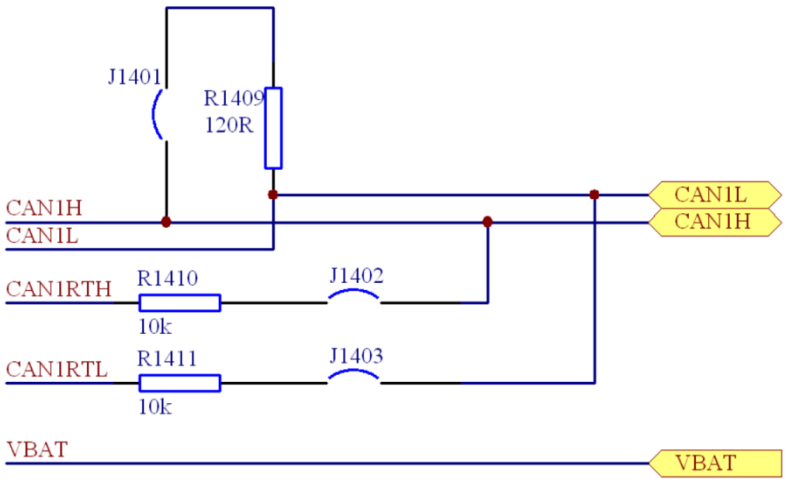

Figure 2-4: CAN Interfacee

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

LIN:

The transceivers are designed as plug-in modules. Generally, the TJA1020 is used for this type of transceicer.

For the standard design of the transceiver modules it is possible to change over between Master and Slave configuration per software. For a Master configuration set the corresponding relay, and for Slave configuration reset it (see Relays).

The pull-up resistors for LIN are located on the transceiver module. Therefore the J1601 or J1701 jumpers must NOT be plugged-in.

Via the V_{Bat} contacts the power supply of the transceiver modules is connected. According to the LIN specification, this power supply is to be carried out via a reverse polarity diode. Therefore the J1602 or J1703 jumpers must NOT be plugged-in.

K-Line:

The transceivers are designed as plug-in modules. Generally, the L9637 is used for this type of transceicer.

Via the V_{Bat} contacts the power supply of the transceiver modules is connected. To bridge the reverse polarity diode for V_{Bat} for LIN, the J1602 or J1703 jumpers must be plugged-in.

In the case the pull-up resistor to V_{Bat} is to be activated, the J1601 or J1701 jumpers must be plugged-in.

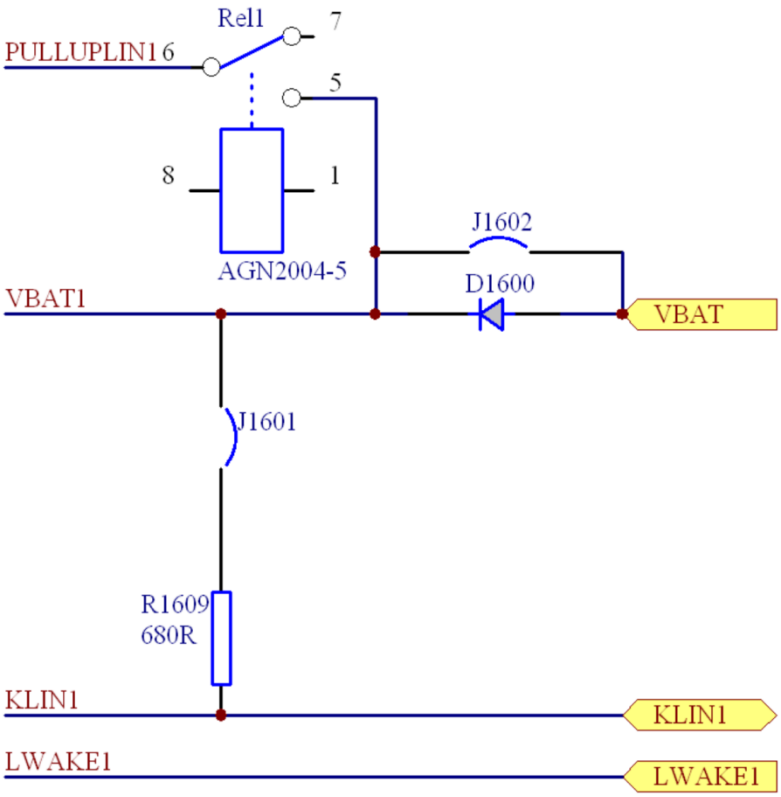

Figure 2-5: LIN/ K-Line Interface

J1850 Interfaces:

The transceivers are designed as plug-in modules.

Generally, the AU5780 is used for J1850 VPW transceivers.

The output circuitry of a J1850 PWM transceiver is realized by discrete components.

The transceiver for a J1850 VPW interface has to be inserted at the position for the J1850 transceiver.
On the other hand, the transceiver for a J1850 PWM interface must be inserted at the position for the LIN/ K-Line 2 transceiver (see also Figure 2-6).

J1701 must NOT be mounted in the case of a J1850 PWM interface!

2.3.4 Peripheral Voltage Supply

The power supply of peripheral devices can be effected via the five voltage females.
The red (+) and the black (–) females are for distributing the not switched peripheral voltage of the input (rear side).

The Rel 1...Rel 3 yellow females offer the plus connection of the switchable peripheral voltages.
After switching on your basicCAR 3085, these voltages are switched off. Activate them by the 0x81…084 Digital Commands (see the GOEPEL Firmware documentation).

The black female (-) can be used as ground connection for the switchable voltages.
The LEDs nearby show the relay status.

2.3.5 Addressing

The individual GOEPEL electronic basicCAR 3085 USB devices are exclusively addressed according to their serial numbers (see Control Software):
The device with the LEAST serial number is always the device with the number 1.

2.3.6 Assembly

Figure 2-6 shows schematically the component side of a board in a basicCAR 3085 device. You can see the positions of the transceiver modules, plug connectors, DIP switches and jumpers.

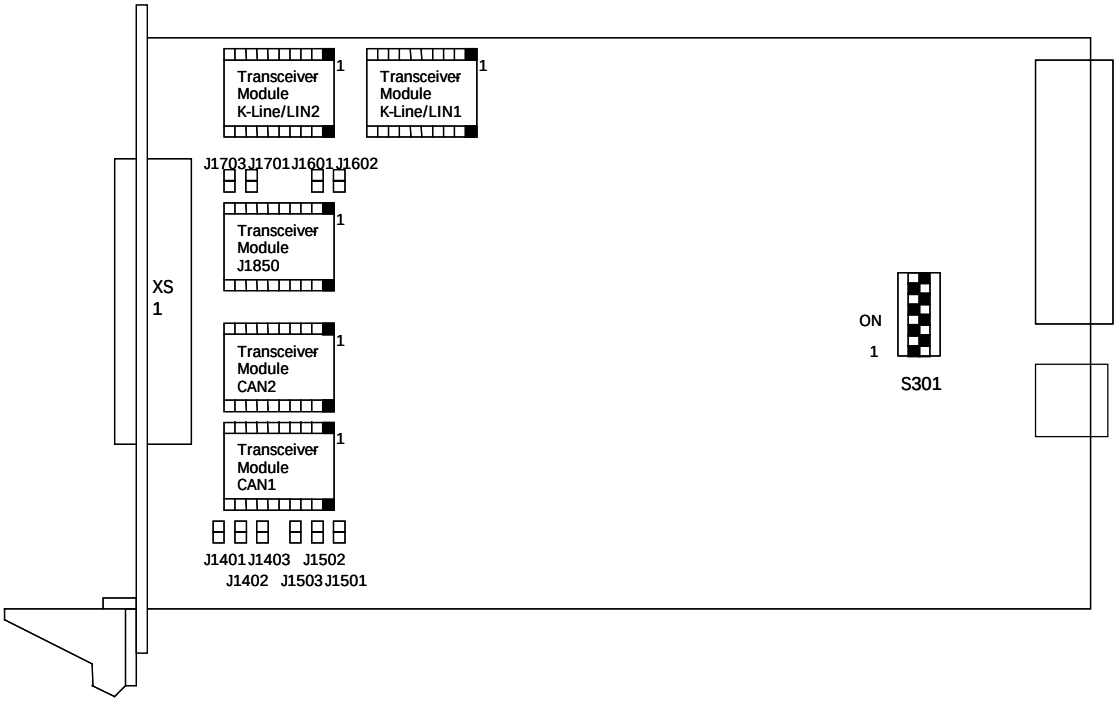

Figure 2-6: Component side of the board for a basicCAR 3085 (schematically)

The configuration elements of Figure 2-6 as well as the indications of the plug connectors for the transceivers are explained in the table:

XS1401	Transceiver module for CAN1
J1401	Jumper to activate the **120Ω** terminating resistor for CAN1
J1402	Jumper to activate the RTH **10kΩ** terminating resistor for CAN1
J1403	Jumper to activate the RTL **10kΩ** terminating resistor for CAN1
XS1501	Transceiver module for CAN2
J1501	Jumper to activate the **120Ω** terminating resistor for CAN2
J1502	Jumper to activate the RTH **10kΩ** terminating resistor for CAN2
J1503	Jumper to activate the RTL **10kΩ** terminating resistor for CAN2
XS1601	Transceiver module for LIN1/ K-Line1
J1601	Jumper to activate the **680Ω** pull-up resistor to V_{BAT} for K-Line1
J1602	Jumper to bridge the reverse polarity diode for V_{Bat} for LIN1
XS1701	Transceiver module for LIN2/ K-Line2/ J1850 PWM
J1701	Jumper to activate the **680Ω** pull-up resistor to V_{BAT} for K-Line2
J1703	Jumper to bridge the reverse polarity diode for V_{Bat} for LIN2
XS1801	Transceiver module for J1850 VPW
S301	DIP switch of the basicCAR 3085 board to configurate the microcontroller. **Do NOT change the settings!**

2.3.7 Connector Assignments

Type: DSub 25 poles socket

For the access to the communication interfaces there is this connector at the front side of the basicCAR 3085 device.

The assignments are shown in the following table:

No.	XS1 pin	Signal name	Remarks
1	1	CAN1_High	CAN bus high
2	14	CAN1_Low	CAN bus low
3	2	CAN2_High	CAN bus high
4	15	CAN2_Low	CAN bus low
5	3	VBAT	Power supply input for transceiver (see Properties)
6	16	GND	Ground potential communication interfaces
7	4	LIN1/ K-Line1	depending on transceiver
8	17	L-Line1/ WAKE1	depending on transceiver
9	5	LIN2/ K-Line2/ J1850 PWM+	depending on transceiver
10	18	L-Line2/ WAKE2/ J1850 PWM-	depending on transceiver
11	6	J1850 VPW	
12	19	-	Please do not assign
13	7	VBAT	Power supply input for transceiver (see Properties)
14	20	GND	Ground potential communication interfaces
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	Wake line	

For K-Line the connections for the L-Line are wired to PIN 17/ 18 if necessary (depending on the output circuitry).

For LIN the connections for the Wake-Line are wired to PIN 17/ 18 if necessary (depending on the selection of the transceiver).

The pins 3 and 7 (V_{BAT}) as well as 16 and 20 (GND) are bridged on every basicCAR 3085 device.

USB Interface

At the device's rear side there is the USB-B-Socket (with USB standard assignment) for the USB 2.0 interface.

2.3.8 LED Display

The LEDs indicate the following states:

- Green LED Rel1: Relay 1 (peripheral supply) on
- Green LED Rel2: Relay 2 (peripheral supply) on
- Green LED Rel3: Relay 3 (peripheral supply) on
- Not labeled green LED: no function

- Red LED Reset: /HDRST
  hardware reset indication output of the microcontroller
- Green LED Ub1: Status indication internal 5V voltage
- Green LED Ub2: Status indication internal 3.3V voltage
- Green LED Ub3: Status indication internal 2.5V voltage

- Yellow LED 1: Status indication CAN 1
  Yellow LED 2: Status indication CAN 2
- Yellow LED 3: Status indication K-Line/ LIN1
- Yellow LED 4: Status indication K-Line/ LIN2

The LEDs are arranged as follows on basiCAR 3085's front side:

Rel1
Rel2
Rel3
Not labeled

Reset	4
Ub1	3
Ub2	2
Ub3	1

2.4 Delivery Notes

basicCAR 3085 devices are delivered in the following basic variants:

- 1x CAN interface and 1x LIN interface or
- 1x CAN interface and 1x K-Line interface

These basic variants can be extended by the following options:

- 1x additional CAN interface
- 1x additional LIN interface or K-Line interface
- 1x additional J1850 VPW interface
- 1x additional J1850 PWM interface

If you select the 1x Additional J1850 PWM interface option, the 1x Additional LIN interface or K-Line interface option is NOT possible.

In addition to selecting an interface, the type of the corresponding CAN/ LIN/ K-Line/ J1850 transceiver as well as the required Functionalities for each CAN/ K-Line/ J1850 interface must be selected.

3 Control Software

There are three ways to integrate the basicCAR 3085 hardware in your own applications:

- Programming via G-API
- Programming via DLL Functions
- Programming with LabVIEW

3.1 Programming via G-API

The G_API (GOEPEL-API) is the favored user interface for this GOEPEL hardware.

You can find all necessary information in the *G-API* folder of the delivered CD.

3.2 Programming via DLL Functions

Programming via DLL Functions is possible also in future for existing projects which can not be processed with the GOEPEL electronic programming interface G-API.

We would be pleased to send the GOEPEL Firmware documentation to you on your request. Please get in touch with our sales department in case you need that.

The GUSB_Platform expression used in the following function description stands for one individual basicCAR 3085 device.

For the used structures, data types and error codes refer to the headers – you find the corresponding files on the supplied CD.

3.2.1 Windows Device Driver

The DLL functions for programming using the Windows device driver are described in the following sections:

- 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*

The GUSB_Platform_Driver_Info function is for the status query of the hardware driver and for the internal initialization of the required handles.

Executing this function at least once is obligatory before calling any other function of the GUSB_Platform driver.

Format:

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

Parameters:

Pointer, for example pDriverInfo
to a data structure
For the structure, see the *GUSB_Platform.h* file on the delivered CD

LengthInByte

Size of the storage area pDriverInfo is pointing to, in bytes

Description:

The GUSB_Platform_Driver_Info function returns information regarding the status of the hardware driver.

For this reason, the address of the pDriverInfo pointer has to be transferred to the function. By means of the LengthInByte parameter the function checks internally if the user memory is initialized correctly.

The function fills the structure pDriverInfo is pointing to with statements regarding the driver version, the number of all involved USB controllers (supported by this driver) and additional information, e.g. the serial number(s).

Making the hardware information available
as well as initializing the belonging handles is obligatory for the further use of the USB hardware.

3.2.1.2 DLL_Info

The GUSB_Platform_DLL_Info function is for the version number query of the DLL.

Format:

```
int GUSB_Platform_DLL_Info(GUSB_Platform_DLLinfo *DLLinformation)
```

Parameters

Pointer, for example DLLinformation
to a data structure
For the structure, see the *GUSB_Platform.h* file on the delivered CD

Description:

The GUSB_Platform_DLL_Info function returns the DLLinfo structure. The first integer value contains the version number of the *GUSB_Platform.dll*.

Examples:

Version number **1.23** is returned as **123**,
and version number **1.60** as **160**.

3.2.1.3 *Write_FIFO*

With the GUSB_Platform_Write_FIFO function a command is sent to the Controller.

Format:

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

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for basicCAR 3085 = 28)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pWrite
to the write data area

DataLength

Size of the storage area pWrite is pointing to, in bytes
Data is consisting of Command Header and Command Bytes
(Currently max. 1024 bytes per command)

Description:

The GUSB_Platform_Write_FIFO function sends a command to the Controller.

For the general structure, see the General Firmware Notes section of the GOEPEL Firmware document.

3.2.1.4 Read_FIFO

The GUSB_Platform_Read_FIFO function is for reading a response from the Controller.

Format:

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

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for basicCAR 3085 = 28)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pRead
to the reading buffer
After successful execution of the function, there is the data in this reading buffer, consisting of Response Header and Response Bytes (Currently max. 1024 bytes per response)

DataLength

Prior to function call: Size of the reading buffer in bytes (to be given)

After function execution: Number of bytes actually read

Description:

The GUSB_Platform_Read_FIFO function reads back the oldest response written by the Controller. In the case no response was received within the fixed Timeout of 100 ms, the function returns NO error, but the Number of bytes actually read is 0 !!!

3.2.1.5 Read_FIFO_Timeout

The GUSB_Platform_Read_FIFO_Timeout function is for reading a response from the Controller within the Timeout to be given.

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)
```

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for basicCAR 3085 = 28)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pRead
to the reading buffer
After successful execution of the function, there is the data in this reading buffer, consisting of Response Header and Response Bytes (Currently max. 1024 bytes per response)

DataLength

Prior to function call: Size of the reading buffer in bytes (to be given)

After function execution: Number of bytes actually read

Timeout

To be given in milliseconds (500 as a standard value)

Description:

The GUSB_Platform_Read_FIFO_Timeout function reads back the oldest response written by the Controller. In the case no response was received within the Timeout to be given, the function returns NO error, but the Number of bytes actually read is 0 !!!

3.2.1.6 Write_COMMAND

With the GUSB_Platform_Write_COMMAND a configuration command is sent to the USB Controller.

Format:

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

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for basicCAR 3085 = 28)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pWrite
to the write data area

DataLength

Size of the storage area pWrite is pointing to, in bytes
See also USB Controller Control Commands
(Currently max. 64 bytes per command)

Description:

The GUSB_Platform_Write_COMMAND function sends a command to the USB Controller.

For the general structure, see the USB Controller Control Commands section.

3.2.1.7 Read_COMMAND

The GUSB_Platform_Read_COMMAND function is for reading a response from the USB Controller.

Format:

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

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for basicCAR 3085 = 28)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Pointer, for example pRead
to the reading buffer
After successful execution of the function, there is the data in this reading buffer, consisting of Response Header and Response Bytes
See also USB Controller Control Commands
(Currently min. 64 bytes per response)

DataLength

Prior to function call: Size of the reading buffer in bytes (to be given)

After function execution: Number of bytes actually read

Description:

The GUSB_Platform_Read_COMMAND function reads back the oldest response written by the USB Controller.

If several responses were provided by the USB Controller, up to two of these responses are written into the buffer of the USB Controller. More possibly provided responses get lost!

3.2.1.8 Xilinx_Download

The GUSB_Platform_Xilinx_Download function is to load an FPGA file to the XILINX.

Format:

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

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for basicCAR 3085 = 28)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

pFileName

Path of the FPGA file to be loaded

pFirmwareErrorCode

Error code occurring during executing this DLL function
(error code 0 means no error occurred)
(error codes -> card firmware see *GUSB_Platform_def.h*)

Description:

The GUSB_Platform_Xilinx_Download function allows to load an FPGA file to the XILINX ((extension **.cfd*).
The loaded data is volatile. Therefore the function has to be executed again after switching off power.

After Xilinx_Download, a delay of about 500 ms is required (as the controller executes a power-on reset).
Then, carry out the 0x10 Software Reset firmware command to come into the normal operating mode from bootloader mode.

3.2.1.9 *Xilinx_Version*

The GUSB_Platform_Xilinx_Version function allows reading out the version of the loaded XILINX firmware.

Format:

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

Parameters:

DeviceName

Type of the addressed device (number declared in *GUSB_Platform_def.h*, for basicCAR 3085 = 28)

DeviceNumber

Number of the addressed device. In the case several devices of the same type are connected, numbering is carried out according to their serial numbers in ascending order (the device with the LEAST serial number has always the DeviceNumber 1).

Version

XILINX software version

Description:

The GUSB_Platform_Xilinx_Version function can be used to read out the version number of the software loaded to the FPGA.

Example:

Version number **2.34** is returned as **234**, version **2.60** as **260**.

3.3 Programming with LabVIEW

3.3.1 LabVIEW via G-API

On the delivered CD there is a folder with VIs to call basicCAR 3085 devices under LabVIEW.

The LabVIEW VIs use the functions of the GOEPEL G-API for this.

3.3.2 LLB using the Windows Device Driver

On the delivered CD there is a folder with VIs to call basicCAR 3085 devices under LabVIEW.

The functions described in the Windows Device Driver section are used for this.

3.4 Further GOEPEL Software

PROGRESS, Program Generator and myCAR of GOEPEL electronic are comfortable programs for testing with GOEPEL hardware.

Please refer to the corresponding Software Manuals to get more information regarding these programs.

3.5 USB Controller Control Commands

The USB Controller is responsible for connecting the basicCAR 3085 device to the PC via USB 2.0.

Messages (generally USB commands) required for configuration can be sent to this USB Controller.

3.5.1 USB Command Structure

A USB command consists of four bytes Header and the Data (but Data is NOT required for all USB commands!).

The header of a USB command has the following structure:

Byte number	Indication	Contents
0	StartByte	0x23 ("#" ASCII character)
1	Command	(0x..) used codes according to USB Commands
2	reserved	0x00
3	reserved	0x00

3.5.2 USB Response Structure

Same as a USB command, also the USB response consists of four bytes Header and the Data (but Data is NOT returned by all USB commands!).

The header of a USB response has the following structure:

Byte number	Indication	Contents
0	StartByte	0x24
1	Command	(0x..) used codes according to USB Commands
2	Length	Length depending on the command
3	ErrorCode	Returns the error code of the command

3.5.3 USB Commands

At present there is only the READ_SW_VERSION USB command available.

Command	Indication	Description
0x04	READ_SW_VERSION	Provides the firmware version of the USB Controller Response: Byte 4: low byte of generic software version Byte 5: high byte of generic software version Byte 6: low byte of software version of functional part Byte 7: high byte of software version of functional part

B

C

G

I

L

T

U

W