Product Specification

# USB 3052
# basicCAN 3052

## CAN Interfaces
## User Manual Version 1.3

GOPEL 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

Printed: 09.06.2010



**Issue: June 2010**

# 1 Installation

## 1.1 Hardware Installation

Generally hardware installation for USB 3052/ basicCAN 3052 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 driver only supports Windows® 2000/ XP systems.

If you want to create your own software for USB 3052/ basicCAN 3052 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 figure shows the successful embedding of one fully equipped USB 3052/ basicCAN 3052 (USB 3052) device with four controllers:

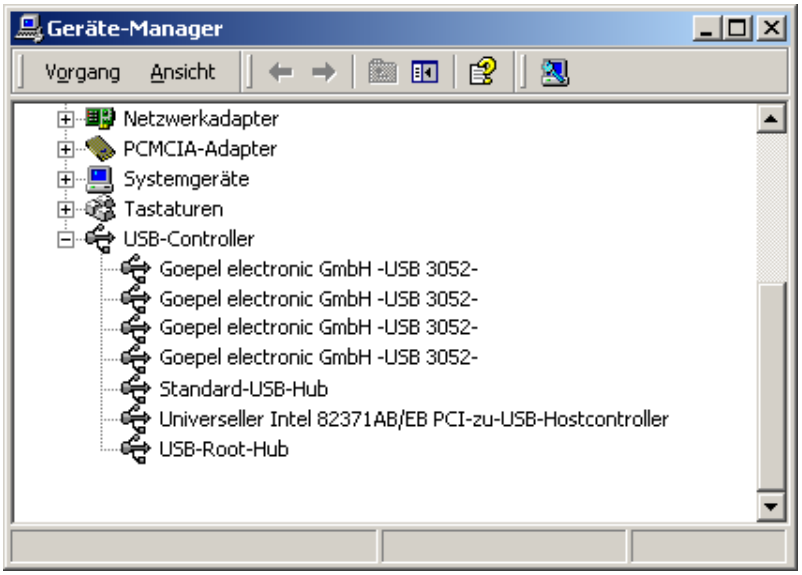


*Figure 1-1: Display of Device Manager*

Please note that the Device-Manager always shows ALL USB controllers.

# 2 Hardware

## 2.1 Definition

USB 3052 CAN boards are GOEPEL electronic GmbH communication boards with USB 2.0 interface.

These boards are used in general control technology, for example for applications in automotive technology.


*Figure 2-1: USB 3052 with 4x CAN*

Please note: Downloading the Xilinx FPGA is absolutely required for operating the USB 3052 board (see Xilinx_Download in the Windows Device Driver section)!

For operating USB 3052 boards you need the GOEPEL electronic USB rack which can cover up to 16 GOEPEL electronic USB boards. In this case, power supply comes from the built-in power supply unit.

basicCAN 3052 is a GOEPEL electronic stand-alone device based on a USB 3052 communication board to be connected to a PC or laptop.

It was in particular developed for applications out of complex test systems. The external power supply allows the use of this device for data acquisition and the inspection of signals e.g. in motor vehicles.


*Figure 2-2: basicCAN 3052*

Power supply with 8..25 VDC (and approx. 600 mA quiescent current at 12 V) is effected via the two ext. Power Supply females (red = plus/ blue= minus) at the device's rear side (opposite to the CAN interfaces connector).

These females are used to supply the internal logic. In addition, the blue female is connected with the GND connections of the USB interface.

On the other hand, all connections of the CAN interfaces are galvanically isolated from the USB interface and the internal logic.

Resources of USB 3052/ basicCAN 3052:

- 2 up to 4 CAN interfaces of Version 2.0b according to the construction stage
- Extended trigger functions with one trigger input and output to the frontal plug connector or the backplane
- Possibility to switch off the CAN sending path without losing the receiving data if no CAN acknowledge was received (see the 0x1C CAN Control Hardware transmitting Path firmware command)
- Galvanic separation of the CAN interfaces from the USB interface and the internal logic
- For each CAN interface there is a 32 bits microcontroller (TriCore TC1765, 40MHz)
- Visualisation of the controller states by LEDs arranged at the front panel (two LEDs per controller, see LED Indication)
- High flexibility through pluggable transceiver modules

In this User Manual, Controller means ALWAYS one of the microcontrollers assigned to each CAN interface
(with the exception of the “CAN Controller” designation on the front panel of a USB 3052 board).

In case a basicCAN 3052 device does not provide enough resources for your applications, there is a GOEPEL electronic USB rack available to cover up to 16 GOEPEL electronic USB boards.
Then the power supply comes from a built-in power supply unit with 230V or 115V connector at the rack’s rear side.

# 2.2 Technical Specification

## 2.2.1 Dimensions

(width x height x depth):

- USB 3052: 4 HP x 130 mm x 185 mm
- basicCAN 3052: 126 mm x 51 mm x 183 mm

The dimensions stated for USB 3052 refer to a board inside the GOEPEL electronic USB Rack.

## 2.2.2 Properties

The characteristics of USB 3052/ basicCAN 3052 are as follows:

| Symbol | Parameter | Min. | Typ. | Max. | Unit | Remarks |
|---|---|---|---|---|---|---|
| Ext. Power Supply | Power supply for internal logic | 8 | 12 | 25 | V | |
| $V_{BAT}$ | Battery voltage | | 12 | 27/ 50 | V | Acc. to transceiver's type |
| | Transmission rate | | | 1 | MBaud | CAN |
| | | | | | | |
| $R_{bus}$ | Terminating resistor 1 | | 120 | | Ohm | CAN jumper plugged in |
| $R_{bus}$ | Terminating resistors 2 | | 5.1 | | kOhm | CAN jumper plugged in |
| | | | | | | |
| | External trigger input | 3.3 | | 50 | V | |
| | External trigger output | | | $V_{BAT}$ | V | Open collector output via npn transistor:<br>max. 50mA/ 200mW |
| | | | | | | |
| $V_{iso}$ | Galvanic separation | 560 | | | V | CAN interfaces/ USB interface and internal logic |

To create a voltage level difference at the external trigger output, an external pull-up resistor must be connected with this output via a voltage source, e.g. 10kΩ via the $V_{Bat}$ voltage.

The external trigger inputs can also be used to feed in an external clock signal.

# 2.3 Construction

## 2.3.1 General

In the basis version, USB 3052/ basicCAN 3052 devices have two CAN interfaces of version 2.0b.
The maximum extension of four CAN interfaces per device can be achieved by means of set-top boards (Aufsatzboards) and further transceiver modules.

Each CAN interface is supported by an own microcontroller.

On the board the USB information is distributed among the controllers.


Please use the delivered USB cables to connect USB 3052/ basicCAN 3052 devices to the PC's USB interface.
Other cables may be inapplicable.

## 2.3.2 Addressing

Addressing an individual USB 3052/ basicCAN 3052 device when operating several USB 3052/ basicCAN 3052 at the same computer takes place exclusively according to the serial numbers of the CAN controllers (see Control Software): The CAN controller with the LEAST serial number is always the device with the number 1.


To improve clarity, we recommend to arrange the individual USB 3052 devices in the USB rack in the order of ascending serial numbers of their CAN controllers (or to connect the individual basicCAN 3052 devices in the same order to the computer).

### 2.3.3 Communication Interfaces

**4 x CAN Interfaces Version 2.0b at most:**

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 USB 3052/ basicCAN 3052 device, the transceivers are designed as plug-in modules. There are several types (highspeed, lowspeed, 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 which is mounted on the board is active for each CAN interface.
These resistors can be deactivated by removing the J1.. J4 jumpers. Then the resistors can be replaced by inserting wired resistors (to be soldered!) of the desired value at the positions RP11, 12 to RP41, 42 (see Figure 2-3).

In the case of lowspeed CAN transceivers, two terminating resistors of 5.1 kOhm each for RTH and RTL per CAN interface are mounted on the transceiver module. Then a wired resistor must not be inserted, and the corresponding jumper has to be removed.

CAN transceivers of the following types require a connection of the battery voltage with the pins 15, 18, 21 or 24 of the XS1 plug connector (V_Bat1..V_Bat4, see Connector Assignments) for the corresponding CAN interface:

- TJA1041A
- TJA1054
- PCA82C252
- B10011S

## 2.3.4 Assembly

Figure 2-3 shows schematically the component side of a USB 3052 board.
The positions of the optional set-top boards (Aufsatzboard) and the transceiver modules can be seen on this illustration as well as the positions of the J1..J4 jumpers for activating/ deactivating the terminating resistors. A plugged-in jumper means that the 120 Ohms terminating resistor is active.

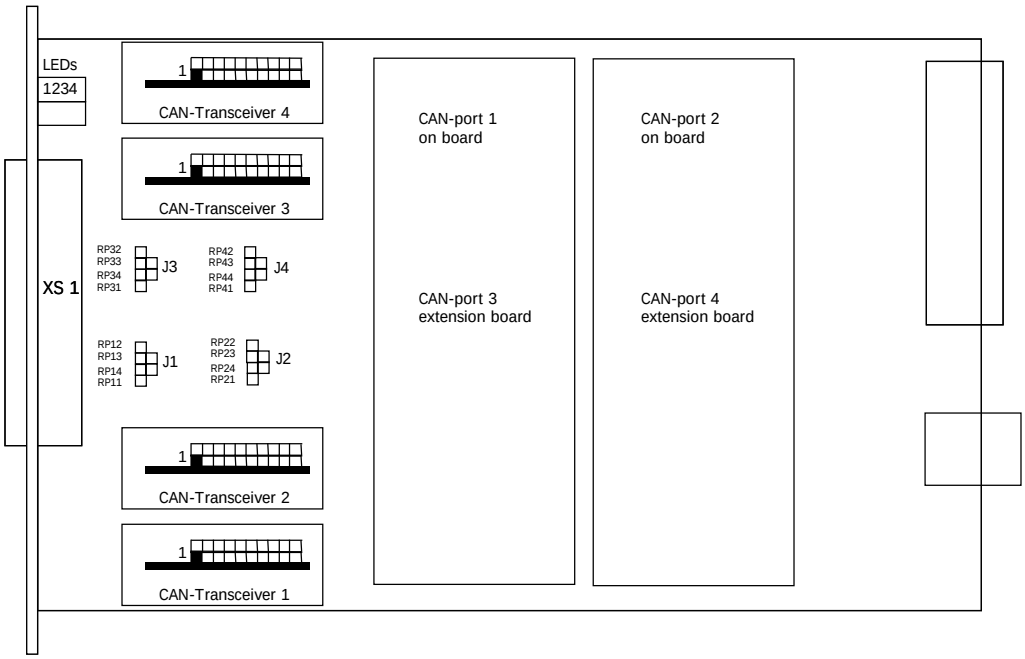


*Figure 2-3: Component side of USB 3052*

The configuration elements of Figure 2-3 are explained in the following table:

| | |
|---|---|
| **CAN Transc.1** | Transceiver module for CAN1 |
| **CAN Transc.2** | Transceiver module for CAN2 |
| **CAN Transc.3** | Transceiver module for CAN3 |
| **CAN Transc.4** | Transceiver module for CAN4 |
| **J1** | Jumper to activate the **120Ω** bus terminating resistor (onboard) for CAN1 |
| **J2** | Jumper to activate the **120Ω** bus terminating resistor (onboard) for CAN2 |
| **J3** | Jumper to activate the **120Ω** bus terminating resistor (onboard) for CAN3 |
| **J4** | Jumper to activate the **120Ω** bus terminating resistor (onboard) for CAN4 |
| **RP 11, 12** | Position for the optional wired terminating resistor – CAN1 |
| **RP 21, 22** | Position for the optional wired terminating resistor – CAN2 |
| **RP 31, 32** | Position for the optional wired terminating resistor – CAN3 |
| **RP 41, 42** | Position for the optional wired terminating resistor – CAN4 |

### 2.3.5 Connector Assignments

Type: DSub 29 poles socket

The signals of the CAN interfaces can be accessed via this connector with the following assignment:

| No. | XS1 pin | Signals name | Remarks |
|---|---|---|---|
| 1 | 14 | CAN1_High | CAN high bus connection |
| 2 | 2 | CAN1_Low | CAN low bus connection |
| 3 | 15 | V_Bat1 | Supply voltage input for Transceiver 1 |
| 4 | 1 | GND | Transceiver/ Trigger ground potential |
| 5 | 17 | CAN2_High | CAN high bus connection |
| 6 | 5 | CAN2_Low | CAN low bus connection |
| 7 | 18 | V_Bat2 | Supply voltage input for Transceiver 2 |
| 8 | 4 | GND | Transceiver/ Trigger ground potential |
| 9 | 20 | CAN3_High | CAN high bus connection |
| 10 | 8 | CAN3_Low | CAN low bus connection |
| 11 | 21 | V_Bat3 | Supply voltage input for Transceiver 3 |
| 12 | 7 | GND | Transceiver/ Trigger ground potential |
| 13 | 23 | CAN4_High | CAN high bus connection |
| 14 | 11 | CAN4_Low | CAN low bus connection |
| 15 | 24 | V_Bat4 | Supply voltage input for Transceiver 4 |
| 16 | 10 | GND | Transceiver/ Trigger ground potential |
| 17 | 3 | INPUT1 | Trigger input 1 |
| 18 | 16 | OUTPUT1 | Trigger output 1 |
| 19 | 6 | INPUT2 | Trigger input 2 |
| 20 | 19 | OUTPUT2 | Trigger output 2 |
| 21 | 9 | INPUT3 | Trigger input 3 |
| 22 | 22 | OUTPUT3 | Trigger output 3 |
| 23 | 12 | INPUT4 | Trigger input 4 |
| 24 | 25 | OUTPUT4 | Trigger output 4 |
| 25 | 13 | GND | Transceiver/ Trigger ground potential |

**USB Interface**

You find the USB-B-Socket (with USB standard assignment) for the USB 2.0 interface opposite to the CAN interfaces side of USB 3052.

### 2.3.6 LED Indication

The LEDs arranged at the front panel of a USB 3052 board indicate the current operating state of the controllers assigned to the CAN interfaces (also called “CAN Ports”).
One green LED and one red LED belong to each CAN interface.
The arrangement is shown in the following figure:

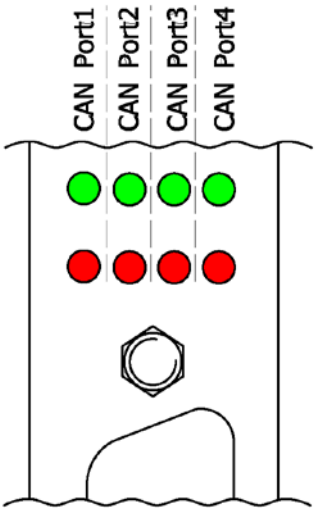


*Figure 2-4: LED Indication*

The LED states are explained in the table:

| green LED | red LED | Remarks |
|---|---|---|
| Permanently ON | | Controller not running<br>Error cause (probably):<br>Xilinx download not executed |
| Alternately blinking | | Bootloader software runs<br>Error cause (probably):<br>Software reset not executed |
| OFF | | Firmware runs |
| ON (shortly) | OFF | Firmware runs executing firmware commands |

## 2.4 Delivery Notes

An individual USB 3052/ basicCAN 3052 device is delivered in the following variants:

- 2x CAN interface (basic variant)
- 3x CAN interface
- 4x CAN interface

In addition to the interface, the type of the corresponding CAN transceiver as well as the required Functionalities for each interface must be selected.

For operating USB 3052 boards you need the GOEPEL electronic USB rack which can cover up to 16 GOEPEL electronic USB boards. In this case, power supply comes from the built-in power supply unit.

# 3 Control Software

There are three ways to integrate USB 3052/ basicCAN 3052 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 the name of a GOEPEL electronic USB driver.

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

In this User Manual, Controller means always the microcontroller assigned to the corresponding CAN interface of a USB 3052/ basicCAN 3052 device. An own USB Controller providing the USB 2.0 interface is assigned to each of these Controllers.

On the other hand, USB Controller means ALWAYS the controller providing the USB 2.0 interface of the USB 3052/ basicCAN 3052 device.

### 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

### Assignment of the USB controllers to a USB 3052/ basicCAN 3052 device

A USB 3052/ basicCAN 3052 device appears with two up to four USB devices in the Windows Device Manager, as each Controller has an own USB Controller (that means each CAN node, too).
See Figure 1-1 in the Driver Installation chapter).

To be able to assign these USB devices to the USB 3052/ basicCAN 3052 device(s), their Controllers and also to the DeviceNumbers, first find out the serial numbers by the Driver_Info command.

The assignment is defined by the remainder of the integer division of the serial number by the number 4. That means, the corresponding serial number must be devided by 4, but without decimal places (modulo, mathematic formula symbol mod).

The following rule is valid:

| Serial number mod 4 | Controller |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |
| 3 | 4 |

Example 1: One USB 3052/ basicCAN 3052 device with 4 Controllers:

| Serial number | Controller | Device | DeviceNumber |
|---|---|---|---|
| 20070040 | 1 | 1 | 1 |
| 20070041 | 2 | 1 | 2 |
| 20070042 | 3 | 1 | 3 |
| 20070043 | 4 | 1 | 4 |

Example 2: Two USB 3052/ basicCAN 3052 devices with 2 Controllers each:

| Serial number | Controller | Device | DeviceNumber |
|---|---|---|---|
| 20070080 | 1 | 1 | 1 |
| 20070081 | 2 | 1 | 2 |
| 20070084 | 1 | 2 | 3 |
| 20070085 | 2 | 2 | 4 |

In the case further Controllers are mounted at Device 1 according to Example 2, the DeviceNumbers change as follows:

Example 3: two USB 3052/ basicCAN 3052 devices with four/ two Controllers:

| Serial number | Controller | Device | DeviceNumber |
|---|---|---|---|
| 20070080 | 1 | 1 | 1 |
| 20070081 | 2 | 1 | 2 |
| 20070082 | 3 | 1 | 3 |
| 20070083 | 4 | 1 | 4 |
| 20070084 | 1 | 2 | 5 |
| 20070085 | 2 | 2 | 6 |

### 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 USB 3052/ basicCAN 3052 = 4)

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).

Please respect the Assignment of the USB controllers ... notes given in the Windows Device Driver chapter: The quantity of "devices" and so of DeviceNumbers corresponds to the quantity of available CAN Interfaces.

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 USB 3052/ basicCAN 3052 = 4)

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).

Please respect the Assignment of the USB controllers ... notes given in the Windows Device Driver chapter: The quantity of "devices" and so of DeviceNumbers corresponds to the quantity of available CAN Interfaces.

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 USB 3052/ basicCAN 3052 = 4)

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).

Please respect the Assignment of the USB controllers ... notes given in the Windows Device Driver chapter: The quantity of "devices" and so of DeviceNumbers corresponds to the quantity of available CAN Interfaces.

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 USB 3052/ basicCAN 3052 = 4)

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).

Please respect the Assignment of the USB controllers ... notes given in the Windows Device Driver chapter: The quantity of "devices" and so of DeviceNumbers corresponds to the quantity of available CAN Interfaces.

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 USB 3052/ basicCAN 3052 = 4)

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).

Please respect the Assignment of the USB controllers ... notes given in the Windows Device Driver chapter: The quantity of "devices" and so of DeviceNumbers corresponds to the quantity of available CAN Interfaces.

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.

This function can only be executed on the USB Controller of the FIRST Controller of a USB 3052/ basicCAN 3052 device.

**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 USB 3052/ basicCAN 3052 = 4)

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).

Please respect the Assignment of the USB controllers ... notes given in the Windows Device Driver chapter: The quantity of "devices" and so of DeviceNumbers corresponds to the quantity of available CAN Interfaces.

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 controllers execute 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 USB 3052/ basicCAN 3052 = 4)

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 USB 3052/ basicCAN 3052 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 USB 3052/ basicCAN 3052 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 Controllers are responsible for connecting the USB 3052/ basicCAN 3052 device to the PC via USB 2.0.

Messages (generally USB commands) required for configuration can be sent to these USB Controllers.

## 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..)<br>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..)<br>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<br><br>Response:<br>Byte 4: low byte of generic software version<br>Byte 5: high byte of generic software version<br>Byte 6: low byte of software version of functional part<br>Byte 7: high byte of software version of functional part |

## C



## G



## L



## U



## W