Produktbeschreibung

PXI 3078

LIN/ KLine Interfaces
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: 26.04.2012

Stand: April 2012

1 Installation der Boards

1.1 Hardware Installation

Stellen Sie bitte unbedingt sicher, dass alle Installationsarbeiten im **ausgeschalteten** Zustand Ihres Systems erfolgen!
Die Stromversorgung sollte abgeklemmt sein.

Vergleichen Sie bitte auch das Handbuch für Ihr PXI-System. Ggf. sind darin weitere zu beachtende Installationshinweise enthalten.

Elektrostatische Entladungen (ESD) können Ihr System schädigen und elektronische Bauelemente zerstören. Das kann zu irreparablen Schäden am PXI 6161-Board oder an dem System führen, in dem das Board betrieben wird.
Folge sind unerwartete Fehlfunktionen Ihres Prüfsystems. Berühren Sie daher niemals die Boardoberfläche, Steckverbinderanschlüsse oder elektronische Bauelemente.

Das *CompactPCI ™*- oder PXI™-System wird entsprechend seinen Gegebenheiten geöffnet. Wählen Sie einen freien Steckplatz in Ihrem System aus.
Beim ausgewählten Steckplatz entfernen Sie das vorhandene Slotblech. Dazu müssen die beiden Befestigungsschrauben gelöst werden.

(Wenn es notwendig ist, Transceivermodule zu tauschen, sind die allgemeinen Regeln zur Vermeidung von elektrostatischen Aufladungen zu beachten. Die Module dürfen nie unter Spannung gezogen oder gesteckt werden! Ein lagerichtiges Stecken der Module ist unbedingt zu realisieren.)

Das Board ist vorsichtig in den vorbereiteten Steckplatz einzuführen. Mit dem an der Frontplatte befindlichen Hebel wird es das letzte Stück eingeschoben.

Nach dem Kontaktieren wird dass Board mit den beiden Schrauben am Frontblech befestigt. Somit ist das Board ordnungsgemäß eingebaut.

Danach sind ggf. die Arbeiten am System auszuführen, die dieses wieder betriebsbereit machen.

1.2 Treiberinstallation

1.2.1 Windows Device Treiber Installation

PXI 3078-Boards können unter Windows® 2000/ XP sowie unter Windows®7/ 64 Bit betrieben werden.

Durch die Plug-and-Play Fähigkeit von Windows® wird automatisch (über den Hardwareassistenten) eine Treiberinstallation für jede neu erkannte Hardwarekomponente gestartet.
Mit der auf der beiliegenden CD im Ordner *GPxi3078* enthaltenen *inf* Datei kann der Hardwareassistent die Installation des Devicetreibers durchführen.

Bei Bedarf finden Sie die jeweils erforderliche *inf*-Datei

- *Pxi3078.inf* für Windows® 2000/ XP
  im Ordner *Driver PXI_PCI - W2K, WinXP (Version xx)*
- *Pxi3078_x64.inf* für Windows®7/ 64 Bit
  im Ordner *Driver PXI_PCI - Win7_x64 (Version xx)*

Ein Neustart des Systems ist nicht zwingend erforderlich.

Der folgende Schritt ist nur erforderlich, wenn Sie nicht mit der G-API arbeiten (siehe auch Ansteuersoftware).

Wenn Sie eigene Software für das Board 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.

1.2.2 VISA Device Treiber Installation

Der PXI 3078 VISA Gerätetreiber ist derzeit für Windows® 2000/ XP verfügbar.

Wir empfehlen für den Einsatz mit PXI 3078 VISA Version 4.3 und höher.

1. Schritt

Kopieren Sie den Ordner *VISA_Driver...* aus dem Ordner *GPxi3078* der mitgelieferten CD auf die Festplatte.

(Empfehlung: vollständigen Ordner auf *C:* \)

2. Schritt

VISA für Windows 2000/ XP :

Durch die Plug-and-Play Fähigkeit wird über den Hardwareassistenten für jede neu erkannte Hardwarekomponente eine automatische Treiberinstallation gestartet. Folgen Sie den Anweisungen und geben Sie bei der Suche nach dem Treiber das Zielverzeichnis an, in dem sich die Datei *PXI3078.inf* befindet
(nach Empfehlung: *C:* *VISA_Driver...*).

VISA für LabView RT :

Für den Einsatz des PXI 3078-Boards unter dem RT Betriebssystem muss die Datei *PXI3078.inf* aus dem Verzeichnis *C:* *VISA_Driver...* verwendet werden.

Kopieren Sie diese Datei in das Verzeichnis *ni-rt* *system* des embedded Controllers.
Nutzen Sie dafür den NI Measurement & Automation Explorer. Unter Netzwerkverbindung finden Sie den angeschlossenen RT-Controller.

Über die rechte Maustaste öffnet sich ein Popup-Menü.
Wählen Sie daraus den Menüeintrag Dateitransfer und folgen Sie den weiteren Anweisungen.

Zum ggf. späteren Erstellen einer *startup.rtexe* sollte auch Ihre Datei *cvi_lvrt.dll* in das Verzeichnis *ni-rt* *system* kopiert werden.

3. Schritt:

Nach einem Neustart des Computers ist die Installation abgeschlossen.

Nach der Treiberinstallation können Sie überprüfen, ob die installierten Boards einwandfrei vom System eingebunden worden sind:

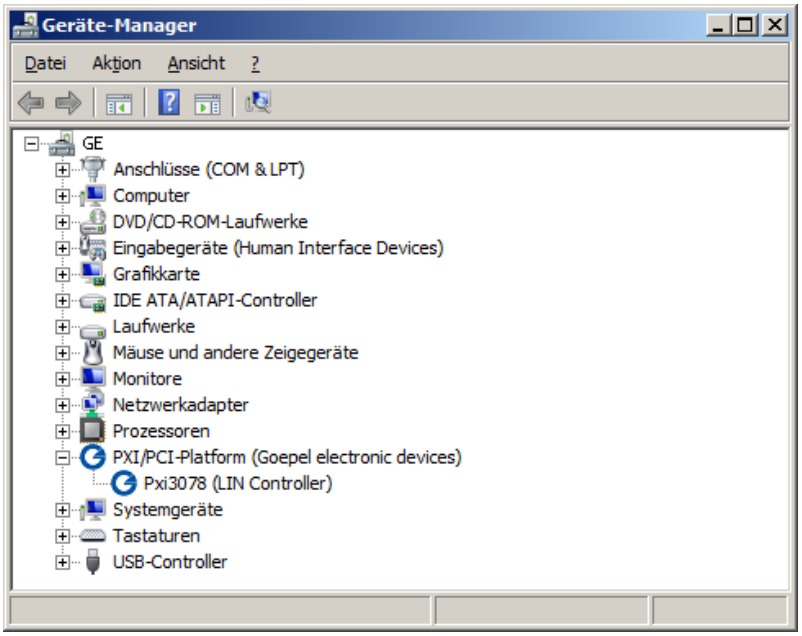

Abbildung 1-1: Windows 7 (64 Bit)

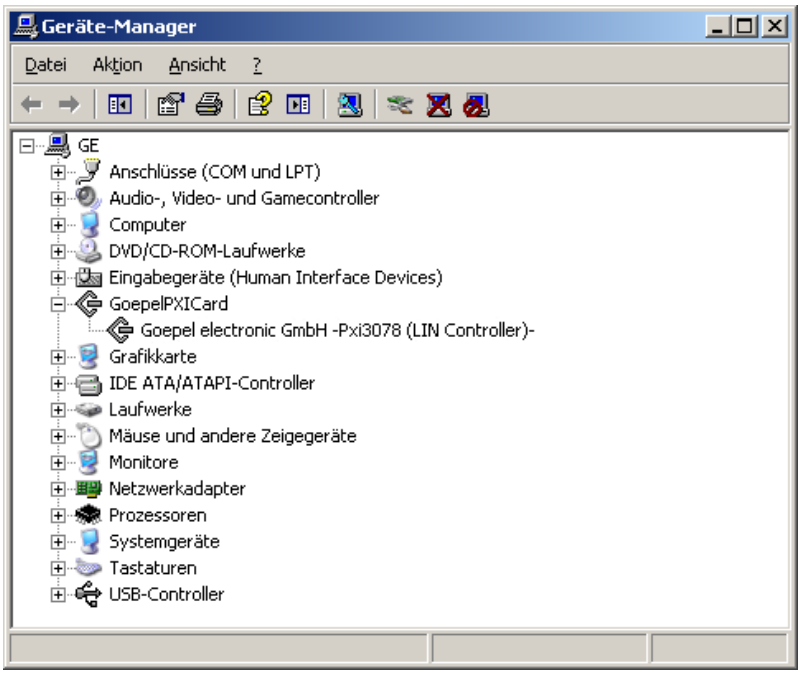

Abbildung 1-2: Windows XP

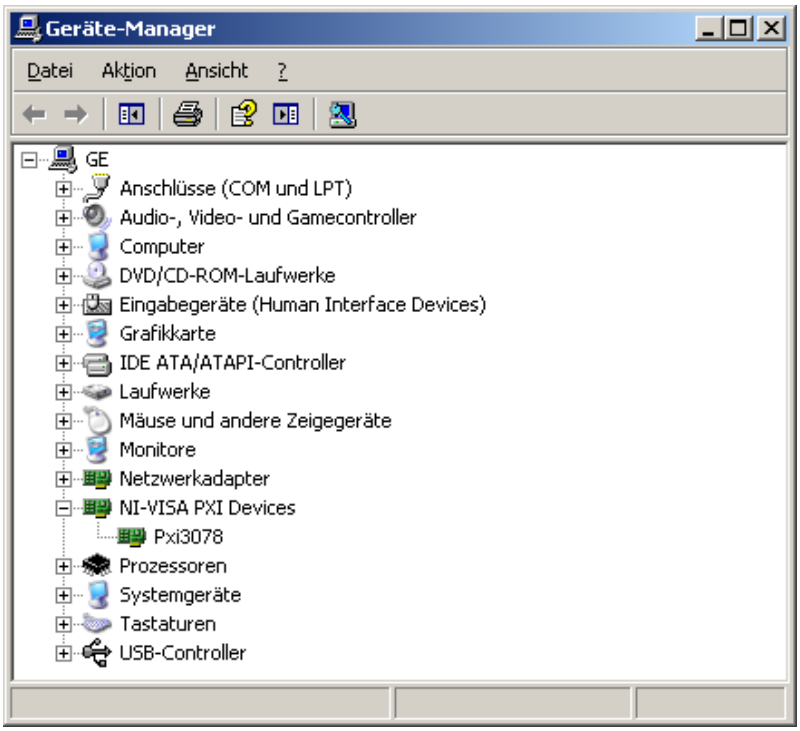

Abbildung 1-3:
VISA für Windows XP

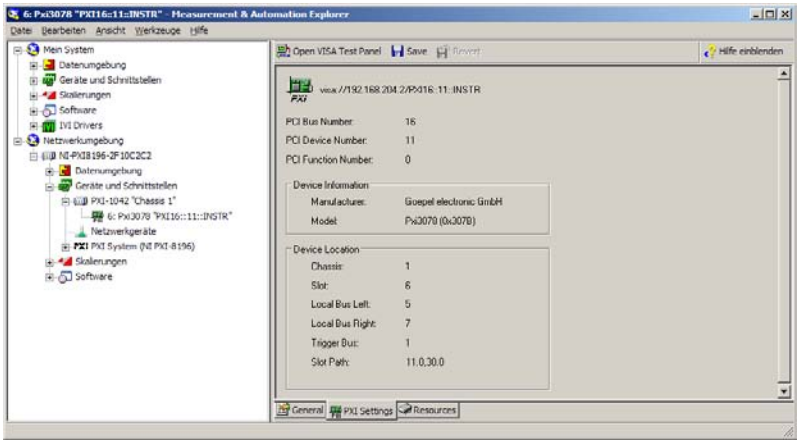

Abbildung 1-4:
VISA für LabVIEW RT

2 Hardware PXI 3078

2.1 Bestimmung

Die LIN-Karte PXI 3078 ist eine Kommunikationskarte der GÖPEL electronic GmbH. Diese Karte wurde für den PXI™-Bus (PCI eXtensions for Instrumentation) entwickelt.

Die PXI 3078 kann in der allgemeinen Steuerungstechnik, speziell in der Automobiltechnik, verwendet werden. Diese Karte dient zur Bereitstellung von LIN-Knoten für die Datenübertragung in LIN-Netzwerken.
Sie ist modular aufgebaut und realisiert 2 LIN-Knoten.
Die Knoten können auch als K-Line konfiguriert werden und sind vollkommen galvanisch getrennt.

Abbildung 2-1: PXI 3078

2.2 Allgemeine Daten

Die PXI 3078 Karte wird über den PCI Bus mit dem Host-PC verbunden und kann so von diesem gesteuert und konfiguriert werden. Die Karte verfügt über einen PCI Controller, einen FPGA und einen Mikrocontroller TC1796.

Zwei Steckplätze, auf welche beliebige Transceiver-Platinen (LIN oder K-Line) gesteckt werden können, sorgen für die Signalumsetzung auf den jeweiligen Busstandard LIN oder K-Line. Die Transceiver sind galvanisch getrennt vom LIN Controller und Host Interface.

2.2.1 PXI-Schnittstelle

Die LIN-Kommunikationskarte PXI 3078 ist ein Einsteckboard, das für den PXI™-Bus (**P**CI e**X**tensions for **I**nstrumentation) entwickelt wurde.
Basis für diesen Bus ist der *CompactPCI™* Bus.
Es ist möglich, die Karte in einem *CompactPCI ™*- oder einem *PXI ™*-System zu betreiben.

Die maximale Übertragungsrate des Busses beträgt 133 MByte/s.
Die PXI 3078 kann auf jeden beliebigen Steckplatz (ausgenommen Steckplatz 1) eines solchen Systems gesteckt werden.
Sie ist auch bei gleichzeitigem Gebrauch mehrerer Karten dieses Typs in einem Rack eindeutig identifizierbar.

2.2.2 Abmessungen

Die Karte hat die Standard-Abmessungen des zugehörigen Bussystems:

- 160 mm x 100 mm x 20,32 mm (L x B x H)

2.2.3 PXI 3078 Kennwerte

Die Kommunikationskarte PXI 3078 hat folgende Kennwerte:

Symbol	Kennwert	Min.	Typ.	Max.	Einheit
V_{CC1}	Versorgungsspannung 1	3,0	3,3	3,6	V
I_{CC1}	Eingangsstrom 1			150	mA
V_{CC2}	Versorgungsspannung 2	4,8	5,0	5,2	V
I_{CC2}	Eingangsstrom 2			300	mA
T_A	Betriebstemperatur	-20	25	60	°C
	Galvanische Isolation			1500	VDC
V_{BusInt}	Spannung interne Busversorgung		12		V
I_{BusInt}	Strom interne Busversorgung			100	mA
V_{BusExt}	Spannung externe Busversorgung			26	V
I_{BusExt}	Strom externe Busversorgung			300	mA
	Übertragungsrate LIN		19,2	22	kBaud
	Übertragungsrate K-Line		10,4	115	kBaud

2.3 Aufbau

2.3.1 Allgemeines

Die LIN-Karte PXI 3078 besitzt zwei LIN oder K-Line Knoten.
Der modulare Aufbau ermöglicht eine individuelle Transceiverkonfiguration.
Die Karte erkennt den aufgesteckten Transceiver dabei automatisch.

Der Mikrocontroller TC1796 sorgt für die deterministische Verarbeitung und Beeinflussung der Bussignale. FPGA und PCI Controller stellen die Schnittstelle zum Host-PC über den PXI™-Bus bereit.

Zwei LEDs signalisieren den Status der Karte.

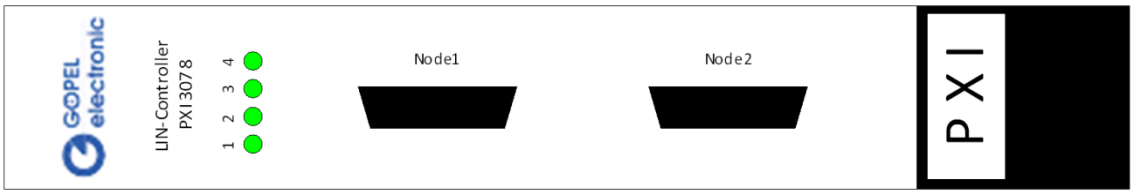

Abbildung 2-2: Frontansicht PXI 3078

2.3.2 LEDs

Die vier LEDs zeigen folgende Zustände an:

LED	Zustand	Beschreibung
1 und 2	blinken abwechselnd	Mikrocontroller befindet sich im Bootloader-Modus
1	leuchtet	Kommandoabarbeitung für Node 1
2	leuchtet	Kommandoabarbeitung für Node 2
3		Reservier
4		Reserviert

2.4 Funktion

2.4.1 Adressierung

Ein PXI-Rack besitzt eine eigene geographische Slotadressierung.

Die Nummerierung beginnt mit 1 und ist auf der Gehäusefrontseite sichtbar. Steckplatz 1 ist immer mit einem embedded Controller oder einer MXI-Karte zu bestücken.

Die Karte PXI 3078 kann die geographische Slotadresse auslesen.

2.4.2 Frontsteckverbinder

Typ: DSub 9-polig Buchse

Die Kommunikationsknoten stehen dem Anwender über diesen Steckverbinder an der Frontseite der Karte PXI 3078 zur Verfügung.

Die Belegung ist bei beiden Knoten identisch und in der folgenden Tabelle für LIN dargestellt:

Pin	Signalname
1	Reserviert
2	Reserviert
3	Gnd
4	n.c.
5	n.c.
6	n.c.
7	LIN
8	Reserviert
9	Externe Busversorgung (V_{BusExt})

Die zweite Tabelle zeigt die Belegung für K-Line:

Pin	Signalname
1	Reserviert
2	L-Leitung
3	Gnd
4	n.c.
5	n.c.
6	n.c.
7	K-Leitung
8	Reserviert
9	Externe Busversorgung (V_{BusExt})

2.4.3 Transceiver

Die Transceiver sind als steckbare Module ausgeführt und werden automatisch von der Firmware erkannt.

Es können beliebige Transceiver für LIN oder K-Line für die beiden Knoten aufgesteckt werden.

LIN:

Im Allgemeinen wird der TJA1020 von Philips als LIN Transceiver verwendet.

In der Standardausführung der Transceivermodule kann per Software über ein Halbleiter-Relais zwischen Master- und Slave-Konfiguration umgeschaltet werden. Die 1kOhm Pullup-Widerstände für den LIN Master werden dementsprechend zu- oder abgeschaltet.

K-Line:

Im Allgemeinen wird der L9637D der Firma ST Microelectronics als Transceiver für K-Line verwendet.

2.4.4 Transceiver Versorgung

Die Transceiver werden standardmäßig von der PXI 3078 Karte versorgt. Die interne Busversorgungsspannung V_{BusInt} beträgt 12V und ist galvanisch getrennt.

Über den Anschluss V_{BusExt} am jeweiligen Portsteckverbinder kann die Versorgungsspannung V_{BAT} des Transceivermoduls auch extern eingespeist werden. Dadurch wird es dem Anwender ermöglicht, den Bus mit Spannungen größer oder kleiner 12V zu versorgen und dementsprechend die Signalpegel anzupassen. Dazu sollte allerdings die interne Busversorgung über die Software abgeschaltet werden.

Die hier eingespeiste Spannung wird durch das Transceivermodul auf 26V begrenzt.

2.5 Lieferhinweise

PXI 3078-Boards sind in folgenden Varianten erhältlich:

- 2x LIN Schnittstelle oder
- 2x K-LINE Schnittstelle

Der LIN Softwareumfang beinhaltet die LIN-Kommunikation gemäß den Spezifikationen 1.3, 2.0 und 2.1.
einschließlich der Diagnose über LIN.

3 Ansteuersoftware

Zur Einbindung der PXI 3078-Hardware in eigene Applikationen existieren drei Möglichkeiten:

- Programmieren über G-API
- Programmieren über DLL-Funktionen
- Programmieren mit LabVIEW

3.1 Programmieren über G-API

Die G_API (GÖPEL-API) ist eine auf „C“ basierende Programmier-Umgebung für GÖPEL electronic-Hardware unter Windows® und die bevorzugte Programmier-Umgebung für diese Hardware.
Sie stellt einen umfangreichen, Hardware-unabhängigen Befehlssatz für CAN, LIN, K-Line, FlexRay, MOST, LVDS, analoge und digitale Ein-/Ausgänge sowie Diagnosedienste zur Verfügung.
Egal ob ein PXI-/ PCI-, USB- oder Ethernet-Gerät genutzt wird – die Befehle sind dieselben.

Die mit der G-API einher gehende Hardware-Abstraktion erlaubt der Testapplikation Parallelzugriff auf die Hardware. Das ermöglicht einer Applikation den Zugriff auf mehrere Hardware-Schnittstellen. Andererseits können auch mehrere Applikationen parallel auf die gleiche Hardware-Schnittstelle zugreifen.

Ein weiteres Feature der G-API ist der asynchrone Hardware-Zugriff. Das bedeutet: Keine Ausführungs-Einschränkungen für wartende Firmwarebefehle. Die Befehls-Quittierung wird über einen Callback-Mechanismus geliefert.

Mit dem Hardware Explorer stellt die GÖPEL electronic GmbH ein effizientes Hardware Konfigurations- und Management-Tool zur Verfügung, das den Anwendern die bequeme Möglichkeit bietet, ihre Hardware-Konfigurationen zu verwalten und auf die einzelnen Hardware-Schnittstellen über logische Namen zuzugreifen. Durch die Verwendung logischer Namen ist ein erneutes Compilieren der Applikation beim Wechsel auf eine andere Schnittstelle oder ein anderes Controllerboard nicht mehr erforderlich: Die Schnittstellen können im Hardware Explorer problemlos neu zugeordnet werden. Außerdem bietet der Hardware Explorer eine einfache Möglichkeit, das Zusammenwirken von Hard- und Software durch die Ausführung integrierter Selbsttests zu überprüfen.

Die folgende Abbildung zeigt den GÖPEL electronic Hardware Explorer:

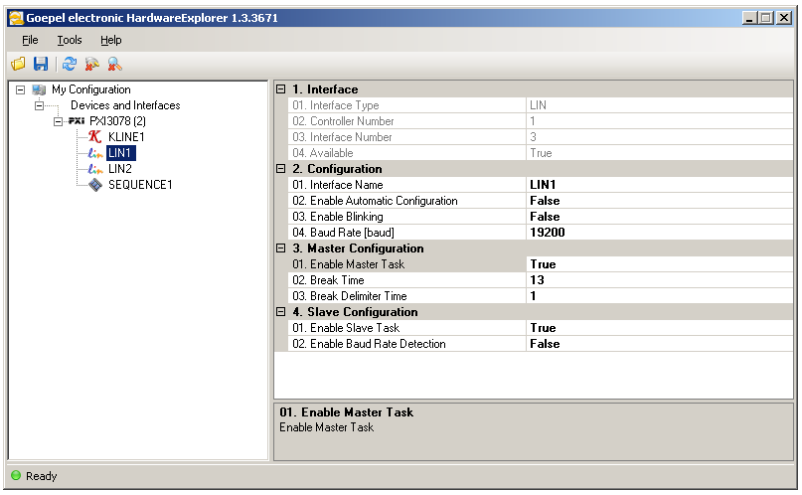

Abbildung 3-1 Hardware Explorer

Bitte vergleichen Sie die G-API Dokumentation für weitere Informationen.
Diese Dokumentation und die Installationssoftware finden Sie im Ordner *G-API* der mitgelieferten CD „Produktinformationen“.

3.2 Programmieren über DLL-Funktionen

Die Programmierung über DLL Funktionen ist auch zukünftig z.B. für existierende Projekte möglich, bei denen nicht mit dem GOEPEL electronic Programmierinterface G-API gearbeitet werden kann.

Informationen zu den Strukturen, Datentypen und Error-Codes enthalten die Header – die entsprechenden Dateien finden Sie auf der mitgelieferten CD (siehe auch Allgemeines zur Firmware).

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:

- DriverInfo
- Xilinx Read Write Register
- Write Instruction
- Read Response

Dabei werden folgende Typdefinitionen verwendet:

U8 – unsigned char

U16 – unsigned short

U32 – unsigned long

3.2.1.1 DriverInfo

Die Funktion Pxi3078__DriverInfo dient zur Status-Abfrage des Hardware-Treibers.

Format:

```
S32 Pxi3078__DriverInfo(t_DriverInfo *pDriverInfo,
                        U32  Length);
```

Parameter:

Zeiger, z.B. pDriverInfo, auf eine Datenstruktur

Zur Struktur siehe das File *Pxi3078_UserInterface.h* auf der mitgelieferten CD

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

Beschreibung:

Die Funktion Pxi3078__DriverInfo gibt Informationen über den Status des Hardware-Treibers zurück.

Dazu muss der Funktion die Adresse des Zeigers pDriverInfo übergeben werden.
Innerhalb der Funktion wird die Struktur, auf die pDriverInfo zeigt, mit verschiedenen Informationen gefüllt.

3.2.1.2 Xilinx Read Write Register

Die Funktion Pxi3078__XilinxReadWriteRegister ermöglicht den FPGA-Zugriff.

Format:

```
S32 Pxi3078__XilinxReadWriteRegister(U8 *pData,
                                     U32 *DataLength);
```

Parameter:

Zeiger, z.B. pData, auf den Bereich für Schreibdaten
bestehend aus Befehlskopf und Befehlsbytes

DataLength
Größe des Speicherbereiches, auf den pData zeigt, in Bytes

Beschreibung:
Siehe oben

3.2.1.3 Write Instruction

Die Funktion Pxi3078__WriteInstruction dient zum Senden eines Befehls zum PXI 3078-Controller.

Format:

```
S32 Pxi3078__WriteInstruction(U8 *pData,
                              U16  DataLength);
```

Parameter:

Zeiger, z.B. pData , auf den Bereich für Schreibdaten,
bestehend aus Befehlskopf und Befehlsbytes
(siehe Befehlsaufbau)

DataLength
Größe des Speicherbereiches, auf den pData zeigt, in Bytes

Beschreibung:

Die Funktion Pxi3078__WriteInstruction sendet einen Befehl zum PXI 3078-Controller.

Im Header der Struktur, auf die pData zeigt, befinden sich die Informationen zum anzusprechenden PXI 3078-Board.
Deshalb sind sie nicht gesondert als Parameter anzugeben.

Die allgemeine Struktur ist im Abschnitt Allgemeines zur Firmware beschrieben.

3.2.1.4 Read Response

Die Funktion Pxi3078__ReadResponse dient zum Lesen einer Antwort vom PXI 3078-Controller.

Format:

```
S32 Pxi3078__ReadResponse(U8  Device,
                          U8 *pData,
                         U32 *DataLength);
```

Parameter:

Device

Index des PXI 3078-Boards, links beginnend mit 1

Zeiger, z.B. pData, auf den Bereich für Lesedaten,

bestehend aus Antwortkopf und Antwortbytes
(siehe Antwortaufbau)

DataLength

Parameterwert vor Funktionsaufruf:
Größe des Puffers, auf den pData zeigt, in Bytes;

Parameterwert nach Funktionsaufruf:
Tatsächlich gelesene Byteanzahl

Beschreibung:

Die Funktion Pxi3078__ReadResponse liest die älteste vom PXI 3078-Controller geschriebene Antwort zurück.

Werden mehrere Antworten vom Controller bereitgestellt, ohne sie zu lesen, gehen diese nicht verloren, sondern werden in einer Art Liste abgelegt.

Aufrufe von Pxi3078__ReadResponse liefern dann solange Werte, bis diese Liste keine Einträge mehr enthält.

3.2.2 VISA Device Treiber

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

- Init
- Done
- Driver Info
- Write Data
- Read Data
- Xilinx Read Write Register

3.2.2.1 Init

Die Funktion PXI3078_Init dient zur Eröffnung von VISA Sessions für alle im System befindlichen PXI 3078-Boards und deren Initialisierung.

Format:

```
ViStatus PXI3078_Init(ViUInt32 *CardCount);
```

Parameter:

CardCount

Anzahl der vom VISA Treiber erkannten PXI 3078-Boards im System.

Beschreibung:

Die Funktion PXI3078_Init sucht alle im System befindlichen PXI 3078-Boards und eröffnet die erforderlichen Sessions.
Außerdem werden Board-interne Initialisierungen durchgeführt.

Deshalb muss diese Funktion als erster Schritt ausgeführt werden.

3.2.2.2 Done

Die Funktion PXI3078_Done schließt alle VISA Sessions für im System befindliche PXI 3078-Boards.

Format:

```
ViStatus PXI3078_Done(void);
```

Parameter:

keine

Beschreibung:

Die Funktion PXI3078_Done schließt alle VISA Sessions für im System befindliche PXI 3078-Boards.

Damit ist kein weiterer Boardzugriff möglich.

3.2.2.3 Driver Info

Die Funktion PXI3078_DriverInfo liefert allgemeine Treiber- und Boardinformationen.

Format:

```
ViStatus PXI3078_DriverInfo(PXI3078_StructDriverInfo *DriverData,
                                          ViChar *DeviceName);
```

Parameter:

Zeiger, z.B. DriverData, auf eine Datenstruktur

Zur Struktur siehe das File *PXI3078_API.h* der mitgelieferten CD

DeviceName
`Array[K_DEV_MAX][K_RES_NAME_LENGTH]`
(siehe *PXI3078_API.h*)

Beschreibung:

Die Funktion PXI3078_DriverInfo stellt verschiedene Informationen zum Treiber und zu den im System befindlichen PXI 3078-Boards zur Verfügung.

Der DeviceName gibt die von VISA erfassten Ressourcennamen an. Diese Informationen korrelieren mit der Anzeige im NI MAX.

3.2.2.4 Write Data

Die Funktion PXI3078_WriteData dient zum Schreiben von Daten zum PXI 3078-Controller.

Format:

```
ViStatus PXI3078_WriteData(ViUInt8 *WriteData,
                           ViUInt32  Length_In_Bytes);
```

Parameter:

Zeiger, z.B. WriteData, auf den Bereich für Schreibdaten,
bestehend aus Befehlskopf und Befehlsbytes
(siehe Befehlsaufbau)

Length_In_Bytes
Größe des Speicherbereiches, auf den WriteData zeigt, in Bytes

Beschreibung:

Die Funktion PXI3078_WriteData ermöglicht das Schreiben von Daten zum PXI 3078-Controller.
Im Header der Struktur, auf die WriteData zeigt, befinden sich die Informationen zum anzusprechenden PXI 3078-Board.
Deshalb sind diese Parameter nicht gesondert anzugeben.

Die Struktur ist im Abschnitt Allgemeines zur Firmware beschrieben.

3.2.2.5 Read Data

Die Funktion PXI3078_ReadData dient zum Lesen von Daten von der ausgewählten Schnittstelle des PXI 3078-Controllers.

Format:

```
ViStatus PXI3078_ReadData(ViUInt8  Card,
                          ViUInt8  Port,
                          ViUInt8 *ReadData,
                         ViUInt32 *DataLength);
```

Parameter:

Card

Index des PXI 3078-Boards, links beginnend mit 1

Port

Nummer der Schnittstelle (1..2)

Zeiger, z.B. ReadData, auf den Bereich für Lesedaten,

bestehend aus Antwortkopf und Antwortbytes
(siehe Antwortaufbau)

DataLength

Parameterwert vor Funktionsaufruf:
Größe des Puffers, auf den ReadData zeigt, in Bytes;

Parameterwert nach Funktionsaufruf:
Anzahl der tatsächlich gelesenen Bytes

Beschreibung:

Die Funktion PXI3078_ReadData ermöglicht das Lesen von Daten, die vom PXI 3078-Controller bereitgestellt wurden (siehe auch Funktion Read Response im Abschnitt Windows Device Treiber).

3.2.2.6 Xilinx Read Write Register

Die Funktion PXI3078_XilinxReadWriteRegister dient zum Lesen von Registerdaten des FPGAs.

Format:

```
ViStatus PXI3078_XilinxReadWriteRegister(ViUInt8 *ReadData,
                                         ViUInt32 *DataLength);
```

Parameter:

Zeiger, z.B. ReadData, auf den Bereich für Lesedaten

DataLength

Parameterwert vor Funktionsaufruf:
Größe des Puffers, auf den ReadData zeigt, in Bytes;

Parameterwert nach Funktionsaufruf:
Anzahl der tatsächlich gelesenen Einträge

Beschreibung:

Die Funktion PXI3078_XilinxReadWriteRegister ermöglicht das Lesen von Registerdaten des FPGAs.

3.3 Programmieren mit LabVIEW

3.3.1 LabVIEW über G-API

Auf der mitgelieferten CD befindet sich eine VI-Sammlung, mit deren Hilfe PXI 3078-Boards 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 PXI 3078-Boards unter LabVIEW angesprochen werden können.

Dabei werden die Funktionen genutzt, die im Abschnitt Windows Device Treiber beschrieben worden sind.

3.3.3 LLB unter Verwendung des VISA Device Treibers

Auf der mitgelieferten CD befindet sich eine VI-Sammlung, mit deren Hilfe PXI 3078-Boards unter LabVIEW angesprochen werden können.

Dabei werden die Funktionen genutzt, die im Abschnitt VISA Device Treiber beschrieben worden sind.

3.4 Weitere GOEPEL Software

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

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

4 Firmwarebefehle

4.1 Allgemeines zur Firmware

In diesem Abschnitt werden die Gemeinsamkeiten für alle Firmware-Befehle auf dem Microcontroller eines PXI 3078-Boards beschrieben (siehe auch Schnittstellen).

Nach den Unterabschnitten

- Schnittstellen (siehe Schnittstellen)
- Datentypen (siehe Datentypen)
- Header (siehe Header)
- Befehlsaufbau (siehe Befehlsaufbau)
- Antwortaufbau (siehe Antwortaufbau)
- Befehlsquittierung (siehe Befehlsquittierung) und
- Befehlsbeispiel (siehe Befehlsbeispiel)

finden Sie die eigentliche Befehlsbeschreibung in den Abschnitten

- Allgemeine Firmwarebefehle
- LIN Befehle
- KLine Befehle.

Bitte achten Sie darauf, für die in der Befehlsbeschreibung mit Reserviert gekennzeichneten Bits und Bytes immer den Wert 0 zu übergeben!

Werden mehr Befehlsbytes übergeben als beschrieben, müssen diese ebenfalls mit 0 übergeben werden.

4.1.1 Schnittstellen

Die Zuordnung der Firmware zu den entsprechenden Schnittstellen-Nummern (Nodes) für ein PXI 3078-Board ist aus der folgenden Tabelle ersichtlich:

Software-Schnittstelle	Controller-Nummer	Schnittstellen-Nummer	PXI 3078 Schnittstelle
LIN	1	3	LIN1
		4	LIN2
K-LINE	1	5	K-LINE1
		6	K-LINE2

4.1.2 Datentypen

Daten werden in der Regel als 8-, 16- und 32-Bit Integer im Intel-Format (little endian) übergeben.
Das bedeutet erst Low-Byte, dann High-Byte(s), siehe folgendes Beispiel für einen 32 Bit Integer-Wert:

Bsp.: Identifier **0x00A534FE** (4 Byte)
Bytearray[x] **0xFE**
Bytearray[x+1] **0x34**
Bytearray[x+2] **0xA5**
Bytearray[x+3] **0x00**

Die Übergabe von Gleitkommawerten erfolgt im
little endian IEEE-754 32-bit single precision format für float und im
little endian IEEE-754 64-bit double precision format für double.

Es folgen weitere Beispiele für unterschiedliche Datentypen:

Datentyp	Wert	Bytes	Byte Offset
16 Bit integer (short)	**0x1234**	**0x34** **0x12**	0 (Low Byte) 1 (High Byte)
32 Bit integer (long)	**0x12345678**	**0x78** **0x56** **0x34** **0x12**	0 (Low Byte) 1 2 3 (High Byte)
32 Bit Gleitkomma (float)	**123.456**	**0x79** **0xE9** **0xF6** **0x42**	0 (Low Byte) 1 2 3 (High Byte)
64 Bit Gleitkomma (double)	**123.456**	**0x77** **0xBE** **0x9F** **0x1A** **0x2F** **0xDD** **0x5E** **0x40**	0 (Low Byte) 1 2 3 4 5 6 7 (High Byte)

4.1.3 Header

Der folgende Header wird für den gesamten Datenaustausch verwendet:

Byte-nummer	Bedeutung	Inhalt
0	StartByte	0x23 (ASCII-Zeichen „#“)
1	Flags	Bit 0 = 1: Befehlsquittierung immer Bit 1 = 1: Befehlsquittierung nur bei Fehler Bits 2..7: Reserviert Somit ergibt sich für den Byte-Wert: 0: Keine Befehlsquittierung 1: Immer Befehlsquittierung 2: Befehlsquittierung nur bei Fehler
2, 3	Length	12..4096 (siehe Befehlsaufbau) Gesamtlänge des Befehls (Header + Parameter)
4	TargetAddress	**Adresse des Controllers** (**1**, **Befehl**) bzw. Adresse der Host-Applikation (0, Antwort)
5	TargetPort	**Schnittstelle auf dem Controller** (**3**..**6** = **LIN**/ **K-LINE**, **Befehl**) bzw. Portadresse der Host-Applikation (0, Antwort)
6	SourceAddress	**Adresse der Host-Applikation** (**0**, **Befehl**) bzw. Adresse des Controllers (1, Antwort)
7	SourcePort	**Portadresse der Host-Applikation** (**0**, **Befehl**) bzw. Schnittstelle auf dem Controller (3..6 = LIN/ K-LINE, Antwort)
8	Type	0: Befehl 1: Antwort 2: Befehlsquittierung
9	ApplicationHandle	Der Inhalt von ApplicationHandle wird bei Antworten unverändert vom Controller an den Host zurückgesendet
10	ModuleAddress	Adresse (Nummer) des PXI 3078-Boards im System, 1..12 Boards anderen Typs werden NICHT mit einbezogen!!!
11	Command	Befehlscode (0x00..) Verwendete Codes entsprechend Firmware, siehe Allgemeine Firmwarebefehle, LIN Befehle und KLine Befehle

Die Befehlsquittierung (Flags/ Bit 0 bzw. Bit 1 = 1) sollte möglichst immer verwendet werden, um nach Befehlsausführung eine Hardware-Rückmeldung zu erhalten.

Die Adresse der Host-Applikation (SourceAddress bei Befehl, TargetAddress bei Antwort) sollte immer 0 sein.
Der Port der Host-Applikation (SourcePort bei Befehl, TargetPort bei Antwort) kann dabei frei vergeben werden (i. Allg. 0).

4.1.4 Befehlsaufbau

Ein Befehl besteht i. Allg. aus Befehlskopf und den Befehlsbytes (aber nicht alle Befehle benötigen Befehlsbytes!).

Der Befehlskopf besteht aus dem Header mit Type = 0 und dem entsprechenden Befehlscode Command. Der Befehlscode ist in dieser Beschreibung am Anfang der Überschrift für den jeweiligen Befehl zu finden.

Die Befehlsbytes sind in ihrer Anzahl durch die maximale Größe eines Befehls (MESSAGE_SIZE) und die Größe des Befehlskopfes (HEADER_SIZE) begrenzt.

Die maximale Anzahl von Befehlsbytes (PARAM_SIZE) ergibt sich somit aus folgender Gleichung:

PARAM_SIZE = MESSAGE_SIZE - HEADER_SIZE

Symbolische Konstante	Wert für PXI 3078	Bemerkung
MESSAGE_SIZE	4096	Maximale Größe des Befehls bzw. der Antwort inklusive Header in Bytes
HEADER_SIZE	12	Größe des Headers in Bytes
PARAM_SIZE	4084	Maximale Größe der Parameter (Befehlsbytes bzw. Antwortbytes) in Bytes

4.1.5 Antwortaufbau

Eine Antwort besteht aus dem Antwortkopf und den Antwortbytes. Der Antwortkopf besteht aus dem Header mit Type = 1 und dem Befehlscode Command entsprechend dem verarbeiteten Befehl.

Die Werte für TargetAddress, TargetPort, SourceAddress und SourcePort werden durch den Richtungswechsel (Befehl/ Antwort) getauscht.

Die Antwort für Firmwarebefehle mit einer Antwort finden Sie direkt nach der eigentlichen Befehlsbeschreibung.

Fehlt diese Antwortbeschreibung, erzeugt der entsprechende Befehl auch keine Antwort.

Andererseits existieren auch Befehle ohne Befehlsbytes. Dann wird nur die entsprechende Antwort beschrieben.

4.1.6 Befehls-quittierung

Befehle, bei denen im Header Bit 0 in Flags auf 1 gesetzt ist, werden nach ihrer Ausführung im Controller durch eine besondere Rückantwort an den Host quittiert, die Befehls-Quittierungs-Antwort.

Diese Rückantwort besteht aus Header und Antwortbytes.

Im Header ist Type = 2 gesetzt, um Befehls-Quittierungs-Antworten von normalen Antworten zu unterscheiden.

Antwort:

Byte	Bezeichnung	Bedeutung
0..3	ErrorNumber	0: Kein Fehler Sonst: Fehler
4.. 3+N	ErrorDescription	Nullterminierter Fehler-String der Länge N (inklusive abschließendem Null-Zeichen); 1 ≤ N ≤ (PARAM_SIZE – 4) (PARAM_SIZE siehe Befehlsaufbau)

Bei Befehlen mit einer Rückantwort (z.B. 0xF0 Firmwareversion abfragen) sendet der Controller bei gesetztem Bit 0 in Flags des 12-Byte-Headers zuerst die eigentliche Antwort und danach die Befehls-Quittierungs-Antwort an den Host. Sind in diesem Fall bestimmte Parameter des Befehls ungültig, sendet der Controller nur eine Befehls-Quittierungs-Antwort (mit entsprechend gesetzter Fehlernummer und Fehlerbeschreibung). Deshalb ist das Auswerten des Types im 12-Byte-Header zwingend notwendig.

Um effizienter mit Befehlen zu arbeiten, die eine Rückantwort erzeugen, sollte nicht das Bit 0, sondern das Bit 1 in Flags des 12-Byte-Headers gesetzt werden. Dadurch kommt immer nur eine Antwort vom Controller: Die gewünschte, wenn kein Fehler aufgetreten ist, oder im Fehlerfall die Befehls-Quittierungs-Antwort.

4.1.7 Befehls-beispiel

Das folgende Beispiel zeigt die einzelnen Bytes inklusive Header des Befehls 0x30 LIN Botschafts-Antwort definieren und der dazugehörigen Befehlsquittierung.

Der Befehl wird zur Software-Schnittstelle 4 (LIN2) des Controllers des dritten PXI 3078-Boards geschrieben (durch Funktionsaufruf Write Instruction, siehe Programmieren über DLL-Funktionen.
Anschließend wird vom selben Controller desselben Boards die Befehlsquittierung gelesen.

Befehl (inklusive Header):

Byte-Index	Byte-Wert	Bezeichnung	Erläuterung
0	**0x23**	StartByte	0x23 (ASCII-Zeichen „#“)
1	**0x01**	Flags	Bit 0 = 1: Befehlsquittierung (immer) aktiviert
2	**0x1C**	Length	Gesamtlänge des Befehls = 12 + 16 = 28, LowByte
3	**0x00**		Gesamtlänge des Befehls = 12 + 16 = 28, HighByte
4	**0x01**	TargetAddress	Adresse des Controllers = 1
5	**0x04**	TargetPort	Schnittstelle auf dem Controller = 4
6	**0x00**	SourceAddress	Adresse der Host-Applikation = 0
7	**0x00**	SourcePort	Portadresse der Host-Applikation = 0
8	**0x00**	Type	0: Befehl
9	**0x00**	ApplicationHandle	Frei wählbar, wird unverändert in Befehlsquittierung übernommen
10	**0x03**	ModuleAddress	Adresse des Boards = 3 (drittes PXI 3078-Board)
11	**0x30**	Command	Befehlscode für 0x30 LIN Botschafts-Antwort definieren
12	**0x12**	Id	Identifier = 0x12
13	**0x01**	Mode	1: LIN Botschafts-Antwort ausgeben
14	**0x00**	PrepareMode	0: LIN Botschafts-Antwort nicht vorbereiten
15	**0x03**	MessageCount	LIN Botschafts-Antwort 3-mal ausgeben
16	**0x06**	DLC	Datenlänge = 6
17	**0x00**	reserved	Reserviert
18	**0x00**		Reserviert
19	**0x00**		Reserviert
20	**0x11**	Data	Datenbyte 0 = 0x11
21	**0x22**		Datenbyte 1 = 0x22
22	**0x33**		Datenbyte 2 = 0x33
23	**0x44**		Datenbyte 3 = 0x44
24	**0x55**		Datenbyte 4 = 0x55
25	**0x66**		Datenbyte 5 = 0x66
26	**0x77**		Datenbyte 6 = 0x77 (Wert ist uninteressant, da DLC = 6)
27	**0x88**		Datenbyte 7 = 0x88 (Wert ist uninteressant, da DLC = 6)

Befehlsquittierung (inklusive Header):

Byte-Index	Byte-Wert	Bezeichnung	Erläuterung
0	**0x23**	StartByte	0x23 (ASCII-Zeichen „#“)
1	**0x00**	Flags	Keine Flags gesetzt
2	**0x11**	Length	Gesamtlänge des Befehls = 12 + 5 = 17, LowByte
3	**0x00**		Gesamtlänge des Befehls = 12 + 5 = 17, HighByte
4	**0x00**	TargetAddress	Adresse der Host-Applikation = 0
5	**0x00**	TargetPort	Portadresse der Host-Applikation = 0
6	**0x01**	SourceAddress	Adresse des Controllers = 1
7	**0x04**	SourcePort	Schnittstelle auf dem Controller = 4
8	**0x02**	Type	2: Befehlsquittierung
9	**0x00**	ApplicationHandle	Der Inhalt wird unverändert zurückgesendet
10	**0x03**	ModuleAddress	Adresse des Boards = 3 (drittes PXI 3078-Board)
11	**0x30**	Command	Befehlscode für 0x30 LIN Botschafts-Antwort definieren
12	**0x00**	ErrorNumber	Fehlernummer (0: Kein Fehler), LowByte
13	**0x00**		Fehlernummer (0: Kein Fehler), MidByte1
14	**0x00**		Fehlernummer (0: Kein Fehler), MidByte2
15	**0x00**		Fehlernummer (0: Kein Fehler), HighByte
16	**0x00**	ErrorDescription	Leerer String (nur die abschließende „Null“)

4.2 Allgemeine Firmwarebefehle

4.2.1 0x03 Funktionalitäten freischalten

Dieser Befehl schaltet (falls vorhanden bzw. möglich) die folgenden Firmware-Elemente für den ausgewählten Controller frei:

- Schnittstellen
- Funktionalitäten

Die Auswahl des Controllers erfolgt durch den Parameter TargetAddress im Header des Befehls.

Welche Firmware-Elemente aktuell freigeschaltet sind, kann über den Befehl 0xF0 Firmareversion abfragen ermittelt werden.

Dieser Befehl besitzt keine Befehlsbytes.

4.2.2 0x10 Software Reset

Dieser Befehl setzt den ausgewählten Controller in den Initialzustand zurück, wobei ein Software-Reset durchlaufen wird.
Die Auswahl des Controllers erfolgt durch den Parameter TargetAddress im Header des Befehls.

Der Befehl besitzt keine Befehlsbytes.

4.2.3 0xF0 Firmareversion abfragen

Mit diesem Befehl wird die Firmware-Version des ausgewählten Controllers abgefragt.
Die Auswahl des Controllers erfolgt durch den Parameter TargetAddress im Header des Befehls.

Dieser Befehl besitzt keine Befehlsbytes.

Antwort:

Byte	Bezeichnung	Bedeutung
0..	Version	Firmwareversion als 0-terminierter String

Im Antwortstring sind vier Informationsteile hinterlegt:

- Firmwareversion (version)
- Erstellungsdatum (date)
- Erstellungszeit (time)
- Freigeschaltete Funktionalitäten (code)
  über 0x03 Funktionalitäten freischalten freigeschaltet

Code für mögliche K-Line Funktionalitäten:

code: `00000000-00010000-00000000-00000000` --> Diagnose KWP2000

code: `00000000-00020000-00000000-00000000` --> Diagnose KWP1281

code: `00000000-00040000-00000000-00000000` --> Diagnose ISO-9141-Ford

Bei mehreren freigeschalteten Funktionalitäten wird der resultierende code bitorientiert über die ODER-Funktion verknüpft aus den einzelnen Codes gebildet.

4.3 LIN Befehle

In diesem Kapitel werden die LIN Befehle für Ihre GÖPEL-Hardware beschrieben.

Die für alle Firmware-Befehle geltenden allgemeinen Angaben finden Sie unter Allgemeines zur Firmware.

Initialzustand:

Nach dem Einschalten oder einem Software-Reset ist die Master-Task deaktiviert und die Slave-Task aktiviert (für das Monitoring).
Die Slave-Task arbeitet mit einer automatischen Baudraten-Erkennung, einer Break-Erkennungs-Schwelle von 9,5 Bitzeiten und dem klassischen Checksummen-Modell (siehe Erläuterung). Dadurch ist es möglich, den LIN Bus ohne Bekanntmachung der Baudrate mit dem LIN Bus Monitor zu beobachten (Befehl 0x54 LIN Monitor – aktivieren/ deaktivieren).

Die Firmware-internen Variablen ResponseSpace und InterByteSpace werden mit 0 initialisiert. Jedoch erscheint auf dem LIN-Bus ein ResponseSpace und InterByteSpace von 0..1 Bitzeit, da für das Senden der UART genutzt wird.

Erläuterung:

Classic checksum = Checksumme über Datenbytes

Enhanced checksum = Checksumme über Identifier-Byte und Datenbytes

In der Firmware wird die Break-Erkennungs-Schwelle (BreakDetectionThreshold) von 9,5 Bitzeiten entgegen der LIN-Spezifikation (11 Bitzeiten) für das Bus-Monitoring genutzt, um auch kürzere Breaks als 11 Bitzeiten zu erkennen.

Die Break-Erkennungs-Schwelle kann aber zu jeder Zeit mit dem Befehl 0x46 LIN BreakDetectionThreshold setzen verändert werden.

Reihenfolge der Befehle

Nach dem Einschalten oder einem Software-Reset müssen vorhandene Optionale Funktionalitäten über den Befehl 0x03 Funktionalitäten freischalten freigeschaltet werden.

Anschließend sollten die folgenden Firmware-Befehle in der angegebenen Reihenfolge ausgeführt werden:

- 0x12 LIN Init Interface
- 0x14 LIN Interface Eigenschaften setzen
- 0x15 LIN ChecksummenModell setzen
  (im Falle von LIN2.0 für die einzelnen Identifier)

für LIN Master:

- 0x22 LIN Schedule-Tabelle füllen
- 0x23 LIN Botschafts-Antwort Tabelle füllen
- 0x28 LIN Master – Senden starten

für LIN Slave:

- 0x23 LIN Botschafts-Antwort Tabelle füllen

Aufbau eines LIN-Clusters

Die folgenden Informationen wurden der LIN-Spezifikation 2.0 entnommen.

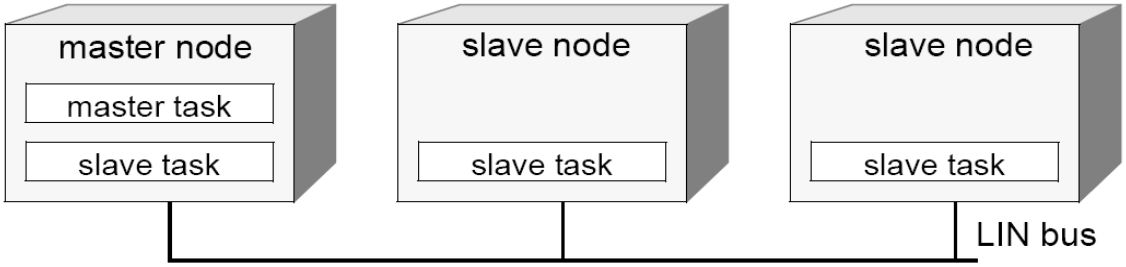

Abbildung 4-1: Aufbau eines LIN-Clusters

Ein LIN-Cluster besteht aus einer Master-Task und mehreren Slave-Tasks.

Der LIN-Master (master node) beinhaltet eine Master-Task und eine Slave-Task.

Die LIN-Slaves (slave node) besitzen jeweils nur eine Slave-Task.

Aufbau einer LIN-Botschaft

Die folgenden Informationen wurden der LIN-Spezifikation 2.0 entnommen.

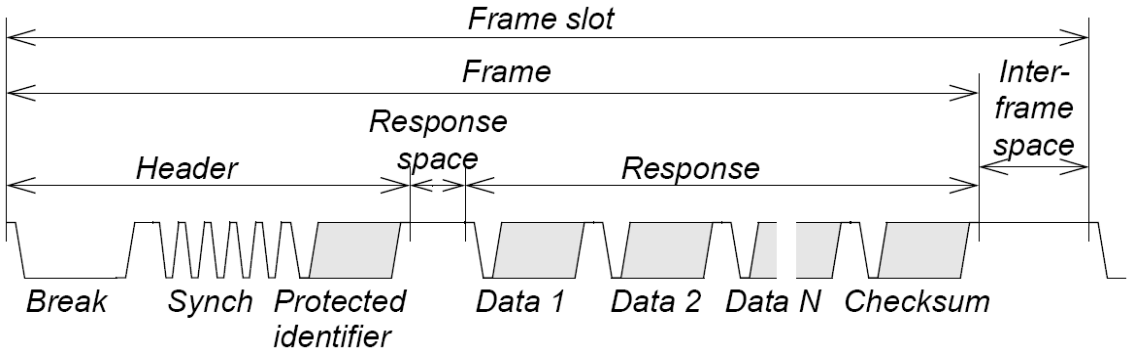

Abbildung 4-2: Aufbau einer LIN-Botschaft

Eine LIN-Botschaft (Frame) besteht im Wesentlichen aus dem LIN-Botschafts-Header (Header) und der LIN-Botschafts-Antwort (Response).

Der LIN-Botschafts-Header wird ausschließlich von der Master-Task des LIN-Masters gesendet.

Die LIN-Botschafts-Antwort wird von der Slave-Task des LIN-Masters bzw. eines LIN-Slaves gesendet.

Die Pause zwischen dem LIN-Botschafts-Header und der LIN-Botschafts-Antwort wird als Response space bezeichnet.

Das folgende Beispiel zeigt eine LIN-Botschaft mit zwei Datenbytes, bestehend aus Header und Response.

Alle mit Firmware-Befehlen änderbaren Zeiten innerhalb einer LIN-Botschaft sind eingezeichnet (BreakTime, BreakDelimiterTime, ArbitrationTime, ResponseSpace und InterByteSpace):

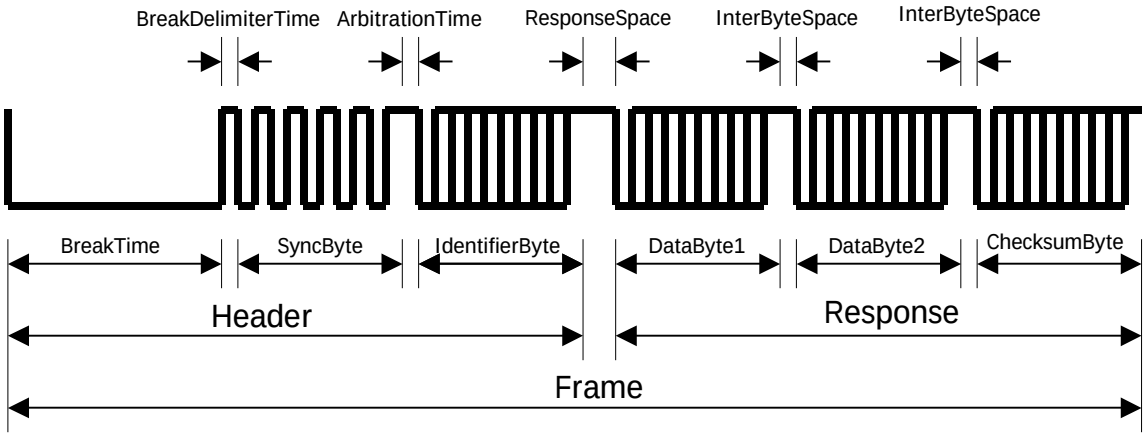

Abbildung 4-3: LIN-Botschaft

Die durchgezogenen Linien oberhalb und unterhalb des LIN-Signals sollen verdeutlichen, dass das Signal zu diesen Zeiten sowohl den Pegel von V_{Bat} (high) als auch den von Gnd (low) annehmen kann.

4.3.1 0x12 LIN Init Interface

Dieser Befehl setzt die ausgewählte LIN Schnittstelle ohne Software-Reset in den Initialzustand zurück. Zusätzlich bietet er weitere optionale Konfigurationsmöglichkeiten.
Die Auswahl der Schnittstelle erfolgt durch die Parameter TargetAddress und TargetPort im Header des Befehls.

Die Befehlsbytes sind optional. Werden keine Befehlsbytes übergeben, arbeitet die Firmware so, als wären die optionalen Befehlsbytes Null.

Befehl:

Byte	Bezeichnung	Bedeutung
0	reserved	Reserviert
1	DontUseUartForTx	0: Benutzung des UART für das Senden (default) 1: Keine Benutzung des UART für das Senden
2	reserved	Reserviert
3	BlinkMode	0: Blinken der LEDs deaktiviert (default) 1: Blinken der LEDs aktiviert

Für das Senden ohne Jitter (Response-Space-Jitter) und die Veränderung der ArbitrationTime, des ResponseSpace und des InterByteSpace ist der Parameter DontUseUartForTx = 1 zu setzen, da beim Senden mit dem UART Verzögerungen (Jitter) bis zu einer Bitzeit entstehen können.

Im anderen Fall bedeutet das Senden mit dem UART innerhalb der Firmware eine geringere CPU Belastung.

4.3.2 0x14 LIN Interface Eigenschaften setzen

Mit diesem Befehl werden die Eigenschaften der ausgewählten LIN-Schnittstelle festgelegt.

Befehl:

Byte	Bezeichnung	Bedeutung
0	EnableMasterTask	0: Deaktivieren der Master-Task 1: Aktivieren der Master-Task (zur Struktur siehe unten)
1	EnableSlaveTask	0: Deaktivieren der Slave-Task 1: Aktivieren der Slave-Task (zur Struktur siehe unten) **Hinweis:** Für das Monitoren ist die Slave-Task zu aktivieren, d.h., der Parameter EnableSlaveTask ist auf 1 zu setzen.
2, 3	reserved	Reserviert

Parameter der Master-Task:

Byte	Bezeichnung	Bedeutung
4..7	BaudRate	BaudRate in Hz
8..11	BreakTime	BreakTime in Vielfachen von 25ns (i. Allg. 13 Bitzeiten, z.B. 27 083 bei 19 200 Baud) (Zeit, die den Start einer neuen LIN-Botschaft signalisiert, Dauer des Low-Pegels des Breaks, siehe Abbildung 4-3)
12..15	BreakDelimiterTime	BreakDelimiterTime in Vielfachen von 25ns (i. Allg. 1 Bitzeit, z.B. 2 083 bei 19 200 Baud) (Zeit zwischen Ende des Low-Pegels des Breaks und Anfang des StartBits des SyncBytes bzw. Dauer des High-Pegels des BreakDelimiters, siehe Abbildung 4-3)
16..19	ArbitrationTime	NUR für Advanced library, sonst mit 0 zu initialisieren ArbitrationTime in Vielfachen von 25ns (i. Allg. 0) (Zeit zwischen Ende des StopBits des SyncBytes und Anfang des StartBits des IdentifierBytes, siehe Abbildung 4-3)

Parameter der Slave-Task:

Byte	Bezeichnung	Bedeutung
20	Enable-BaudRateDetection	0: BaudRate Erkennung deaktiviert 1: BaudRate Erkennung aktiviert
21..23	reserved	Reserviert
24..27	BaudRate	BaudRate in Hz
28..31	ResponseSpace	NUR für Advanced library, sonst mit 0 zu initialisieren ResponseSpace in Vielfachen von 25ns (i. Allg. 0) (Zeit zwischen Ende des StopBits des IdentifierBytes und Anfang des StartBits der Response bzw. Zeit zwischen Header und Response, siehe Abbildung 4-3)
32..35	InterByteSpace	NUR für Advanced library, sonst mit 0 zu initialisieren InterByteSpace in Vielfachen von 25ns (i. Allg. 0) (Zeit zwischen Ende des StopBits eines Response DataBytes und Anfang des StartBits des nächsten Response DataBytes, siehe Abbildung 4-3)

Wenn die Slave-Task nicht aktiviert ist, liefern die Monitor-Befehle kein Ergebnis!!!

Beachten Sie bitte, dass zur Realisierung der Zeiten ArbitrationTime, ResponseSpace und InterByteSpace der UART für das Senden NICHT genutzt werden sollte (siehe Befehl 0x12 LIN Init Interface).

Beachten Sie bitte ebenfalls, dass der Sendezeitpunkt für die LIN-Botschafts-Antwort, angegeben durch ResponseSpace, durch die Transceiver-Durchlaufzeiten (Empfangs- und Sende-Durchlaufzeit) verfälscht wird.
Die Gesamtdurchlaufzeit eines Transceivers ist Typ-abhängig und kann zwischen 5µs und 15µs liegen (i. Allg. etwa 8µs bis 9µs).

4.3.3 0x15 LIN Checksummen-Modell setzen

Dieser Befehl wird zum Setzen des Checksummen-Modells verwendet.

Befehl:

Byte	Bezeichnung	Bedeutung
0	CheckSumModel	0: Classic (Checksumme nur über die Datenbytes) 1: Enhanced (Checksumme über Identifier und Datenbytes)
1	NumberOfIds	Anzahl der Identifier (N)
2	SelectAll	0: Das Setzen gilt nur für die unter Ids angegebenen Identifier 1: Das Setzen gilt für alle Identifier
3	reserved	Reserviert
4.. (3+N)	Ids	Identifier-Liste (pro Identifier ein Byte)

4.3.4 0x1A LIN Node

Dieser Befehl dient zur Konfiguration und Steuerung der LIN-Schnittstelle als LIN-Knoten. Der Befehl unterteilt sich in mehrere Unterbefehle, welche durch den Parameter Mode unterschieden werden.

Alle Unterbefehle besitzen bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	Mode	4: PROPERTY__SET_BY_ID 5: PROPERTY__GET_BY_ID
1..3	reserved	Reserviert

PropertyId:

Wert	Bedeutung
18	**BAUD_RATE** Baudrate in Hertz
19	**BAUD_RATE_DETECTION__IS_ENABLED** Baudratendetektion (0 = deaktiviert, 1 = aktiviert)
20	**UART__USE_FOR_TRANSMIT** Legt fest, ob der UART für das Senden genutzt wird, ohne die Schnittstelle neu zu initialisieren (0 = Der UART wird NICHT für das Senden genutzt 1 = Der UART wird für das Senden genutzt). Siehe auch Befehl 0x12 LIN Init Interface
49	**ENABLE_BUS_TERMINATION** 0 = Deaktivieren des Busabschlusses (Pull-Up-Resistor) 1 = Aktivieren des Busabschlusses (Pull-Up-Resistor)
50	**ENABLE_INTERNAL_VBAT** 0 = Deaktivieren der internen Spannungsversorgung für den Transceiver 1 = Aktivieren der internen Spannungsversorgung für den Transceiver (dadurch kann das Anlegen einer externen Spannung entfallen.)

4.3.4.1 Property SetById

Das SubCmd = PROPERTY__SET_BY_ID dient zum Setzen des Wertes einer Eigenschaft, die durch PropertyId spezifiziert ist.

Befehl:

Byte	Bezeichnung	Bedeutung
4, 5	Id	ProperyId: Eigenschafts-Identifier (siehe PropertyId in Abschnitt 0x1A LIN Node)
6, 7	Reserved	Reserviert
8..11	Value	Wert der Eigenschaft

4.3.4.2 Property GetById

Das SubCmd = GET_FLAG_BY_ID dient zur Abfrage des Wertes einer Eigenschaft, die durch PropertyId spezifiziert ist.

Befehl:

Byte	Bezeichnung	Bedeutung
4, 5	Id	ProperyId: Eigenschafts-Identifier (siehe PropertyId im Abschnitt 0x1A LIN Node)
6, 7	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
4, 5	Id	ProperyId: Eigenschafts-Identifier (siehe PropertyId im Abschnitt 0x1A LIN Node)
6, 7	Reserved	Reserviert
8..11	Value	Wert der Eigenschaft

4.3.5 0x22 LIN Schedule-Tabelle füllen

Dieser Befehl dient zum Füllen einer LIN Schedule-Tabelle.

Insgesamt existieren in der Firmware 256 Schedule-Tabellen; die Einträge hängen vom verfügbaren Speicher ab.

Wenn die zu füllende Tabelle mehr Einträge besitzt als mit einem Befehl geschrieben werden können, ist dieser Befehl mehrfach auszuführen.
Dazu ist der Parameter Concatenate entsprechend zu setzen.

Befehl:

Byte	Bezeichnung	Bedeutung
0	ScheduleTable-Number	Nummer der Schedule-Tabelle
1	Concatenate	0: Schreiben ab Anfang der Tabelle 1: Tabellen-Einträge anhängen
2	IdAsIdCode	0: Der Protected-Identifier wird entsprechend der Spezifikation von der Firmware berechnet 1: Der Protected-Identifier wird wie übergeben übernommen (Dadurch ist das Senden eines ungültigen Identifiers möglich!)
3..5	reserved	Reserviert
6, 7	NumberOfItems	Anzahl der Tabellen-Einträge (N)
8.. 7+(N*8)	Items	Tabellen-Einträge (zur Struktur siehe unten)

Ein Schedule-Tabellen-Eintrag besteht aus den folgenden acht Bytes:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier
1	FrameType	0: UnconditionalFrame – „Normale“ Botschaften 1: EventTriggeredFrame – Ereignis-getriggerte Botschaften 2: SporadicFrame – Sporadische Botschaften 3: DiagnosticFrame – Diagnose-Botschaften (0x3C und 0x3D)
2	FrameListIndex	Listenindex für Event Triggered Frames und Sporadic Frames (beginnend mit 0)
3	reserved	Reserviert
4..7	Delay	Verzögerung zur nächsten LIN-Botschaft in Vielfachen von 25ns (z.B. 400 000 für ein Delay von 10ms)

So, wie für die Unconditional Frames in einem LIN Description File (LDF-File) eine Liste Frames existiert, existiert ggf. eine Liste Event_triggered_frames für die Event Triggered Frames oder auch eine Liste Sporadic_frames für die Sporadic Frames.

FrameListIndex ist der Index in diesen Listen.

4.3.6 0x23 LIN Botschafts-Antwort Tabelle füllen

Mit diesem Befehl wird die LIN Botschafts-Antwort Tabelle gefüllt.

Wenn die zu füllende Tabelle mehr Einträge besitzt als mit einem Befehl geschrieben werden können, ist dieser Befehl mehrfach auszuführen.

Mit dem hier beschriebenen Befehl werden die mit Id in jedem einzelnen Tabellen-Eintrag (Items) angegebenen LIN Botschafts-Antworten automatisch definiert.

Befehl:

Byte	Bezeichnung	Bedeutung
0, 1	NumberOfItems	Anzahl der Tabelleneinträge (N)
2, 3	reserved	Reserviert
4.. 3+(N*12)	Items	Tabelleneinträge (zur Struktur siehe unten)

Ein Tabellen-Eintrag einer LIN-Botschafts-Antwort besteht aus den folgenden 12 Bytes:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1	Length	Datenlänge
2, 3	reserved	Reserviert
4..11	Data[0..7]	Datenbytes 0..7

Als Alternative zu diesem Befehl existiert der Befehl 0x30 LIN Botschafts-Antwort definieren, der mehr Möglichkeiten bietet.

4.3.7 0x24 LIN WakeUp Request senden

Mit diesem Befehl wird ein Wake-Up-Request gesendet.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	WakeUpTime	Dauer des dominanten Bus-Pegels in Vielfachen von 25ns, (z.B. 10 000 für eine WakeUp Time von 250µs) (0 = Default WakeUpTime)

Wird als WakeUpTime eine Null übergeben, wird mit einer Default-WakeUpTime von acht Bit-Zeiten geweckt.
Gemäß der LIN 2.0 Specification sollte die WakeUpTime zwischen 250µs und 5ms liegen.

4.3.8 0x25 LIN Slave-Task Zustand setzen

Dieser Befehl dient dazu, den Slave-Controller in den Sleep-Zustand zu setzen bzw. ihn aus diesem Zustand zu „wecken“.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Awake	0: Sleep-Zustand 1: Awake-Zustand
1..3	reserved	Reserviert

4.3.9 0x26 LIN GotoSleep Befehl senden

Der Befehl aktiviert die Möglichkeit, ein Go to Sleep Command zu senden. Unmittelbar nach der Aktivierung wird die Ausführung dieses Befehls beendet.

Der Befehl besitzt keine Befehlsbytes.

Das Senden des Go to Sleep Commands selbst erfolgt, wenn der Master einen LIN-Header mit dem Identifier 0x3C (Diagnostic Master Request Frame) auf den Bus legt. Als erstes Datenbyte wird dazu eine Null gesendet. D.h., der Zeitpunkt zum Senden des Go to Sleep Commands wird ALLEIN vom LIN Master bestimmt.

Ein Go to Sleep Command kann von jedem Controller gesendet werden (auch vom Slave).

Das Senden eines Go to Sleep Commands ist während der aktiven Diagnose (Type ≠ 0 bei 0xA0 LIN Diagnose – Konfiguration) NICHT möglich.

4.3.10 0x28 LIN Master – Senden starten

Durch diesen Befehl beginnt die Master-Task, die angegebene Schedule-Tabelle abzuarbeiten und somit die entsprechenden LIN-Botschafts-Header zu senden.

Befehl:

Byte	Bezeichnung	Bedeutung
0	ScheduleTableNumber	Nummer der Schedule-Tabelle
1..3	reserved	Reserviert

Die angegebene Schedule-Tabelle wird immer ab dem ERSTEN Eintrag abgearbeitet.

4.3.11 0x29 LIN Master – Senden stoppen

Durch diesen Befehl stoppt die Master-Task, LIN-Botschafts-Header zu senden.

Der Befehl besitzt keine Befehlsbytes.

4.3.12 0x2A LIN Schedule-Tabelle leeren

Durch diesen Befehl wird die angegebene LIN Schedule-Tabelle geleert.

Befehl:

Byte	Bezeichnung	Bedeutung
0	ScheduleTableNumber	Nummer der Schedule-Tabelle
1..3	reserved	Reserviert

4.3.13 0x2B LIN Botschafts-Antwort Tabellen-Einträge löschen

Mit diesem Befehl können Sie alle oder nur die durch Ids bestimmten Einträge aus der LIN Botschafts-Antwort Tabelle löschen.

Befehl:

Byte	Bezeichnung	Bedeutung
0, 1	NumberOfIds	Anzahl der Identifier (N)
2	SelectAll	0: Nur die unter Ids angegebenen Tabellen-Einträge leeren 1: Alle Tabellen-Einträge leeren
3	reserved	Reserviert
4.. (3+N)	Ids	Identifier-Liste (pro Identifier ein Byte)

4.3.14 0x2C LIN Schedule-Tabelle wechseln

Durch diesen Befehl kann in eine andere Schedule-Tabelle gewechselt werden (auch während die aktuelle noch abgearbeitet wird).

Im Normalfall findet der Wechsel in die angegebene Schedule-Tabelle immer nach Abarbeitung der aktuellen Schedule-Tabelle statt. Durch Setzen des Flags ChangeImmediately kann ein sofortiger Wechsel erzwungen werden.

Befehl:

Byte	Bezeichnung	Bedeutung
0	ScheduleTableNumber	Nummer der Schedule-Tabelle
1	Flags	Im Normalfall sind alle Flags 0. Bit 0: StartSchedulerBefore (Der Scheduler wird vor dem Wechseln der Schedule-Tabelle gestartet) Bit 1: StopSchedulerAfter (Der Scheduler wird nach NumberOfCycles-facher Abarbeitung der Schedule-Tabelle gestoppt) Bit 2: ChangeToPreviousTableAfter (Der Scheduler wechselt nach NumberOfCycles-facher Abarbeitung der Schedule-Tabelle zur vorherigen Schedule-Tabelle zurück) Bit 3: ChangeImmediately (Der Scheduler wechselt sofort in die angegebene Schedule-Tabelle) Bit 4: StartSchedulerOnWakeUp (Der Scheduler startet nach einem WakeUp automatisch) Bits 5..7: Reserviert
2, 3	reserved	Reserviert
4..7	NumberOfCycles	Anzahl von Schedule-Tabellen-Durchläufen, bevor in die vorherige Schedule-Tabelle zurück gewechselt wird (falls das Flag ChangeToPreviousTableAfter gesetzt ist) bzw. der Scheduler gestoppt wird (falls das Flag StopSchedulerAfter gesetzt ist). Ist keines der beiden Flags gesetzt, ist NumberOfCycles auf 0 zu setzen.

Die angegebene Schedule-Tabelle wird immer ab dem ERSTEN Eintrag abgearbeitet.

Der Scheduler ist für die Abarbeitung der Schedule-Tabelle in der Master-Task zuständig. Deshalb wird die Schedule-Tabelle bei angehaltenem Scheduler NICHT abgearbeitet. Dementsprechend werden auch keine LIN-Botschafts-Header von der Master-Task gesendet.

4.3.15 0x2E LIN Master Task

Dieser Befehl dient zur Steuerung der LIN-Master-Task.

Der Befehl unterteilt sich in mehrere Unterbefehle, die durch den Parameter SubCmd unterschieden werden.

Alle Unterbefehle besitzen bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	SubCmd	0: SEND_HEADER
1..3	reserved	Reserviert

4.3.15.1 Send Header

Das SubCmd = SEND_HEADER dient zum sofortigen Senden eines LIN-Frame-Headers, ohne eine Schedule-Tabelle zu füllen und zu starten.

Befehl:

Byte	Bezeichnung	Bedeutung
4	Flags	Bit 0 = 0: Der protected Identifier (PID) wird automatisch berechnet Bit 0 = 1: Der protected Identifier (PID) ist direkt der Wert von Id Bits 1..7: Reserviert
5	Id	LIN-Identifier
6, 7	Reserved	Reserviert

4.3.16 0x30 LIN Botschafts-Antwort definieren

Dieser Befehl dient zum Definieren der durch Id festgelegten LIN-Botschafts-Antwort.

Das Definieren einer LIN Botschafts-Antwort mit Hilfe dieses Befehls bildet eine Alternative zu 0x23 LIN Botschafts-Antwort Tabelle füllen, um LIN Botschafts-Antworten zu senden. Dieser Befehl bietet jedoch mehr Möglichkeiten, wie z.B. eine begrenzte Sendeanzahl (MessageCount ≠ 0) und das Bilden einer Gruppe von LIN Botschafts-Antworten (PrepareMode = 1).

Beachten Sie bitte, dass eine LIN Botschafts-Antwort immer nur dann gesendet wird, wenn die Master-Task (auf diesem oder einem anderen LIN-Knoten) den entsprechenden LIN Botschafts-Header gesendet hat.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1	Mode	0: LIN Botschafts-Antwort nicht senden 1: LIN Botschafts-Antwort senden
2	PrepareMode	0: LIN Botschafts-Antwort nicht vorbereiten 1: LIN Botschafts-Antwort vorbereiten
3	MessageCount	0: LIN Botschafts-Antwort immer senden 1 ≤ N ≤ 255: LIN Botschafts-Antwort N-mal senden
4	Dlc	Datenlänge (0..8)
5..7	reserved	Reserviert
8..15	Data[0..7]	Datenbytes 0..7

Das „Vorbereiten“ dient zum parallelen Starten und Stoppen mehrerer LIN Botschafts-Antworten (siehe Befehle 0x34 LIN Vorbereitete Botschafts-Antworten starten und 0x35 LIN Vorbereitete Botschafts-Antworten stoppen).

Das Senden der LIN Botschafts-Header wird NICHT durch diesen Befehl ausgelöst oder beeinflusst.

4.3.17 0x31 LIN Botschafts-Antwort löschen

Nach Aufruf dieser Funktion wird nicht nur das Senden der durch Id festgelegten LIN Botschafts-Antwort beendet, sondern diese LIN Botschafts-Antwort wird auch aus der internen Verwaltung entfernt.

Ein erneutes Senden ist nur nach Ausführung des Befehls 0x30 LIN Botschafts-Antwort definieren möglich.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1..3	reserved	Reserviert

4.3.18 0x32 LIN Botschafts-Antwort Modus ändern

Dieser Befehl dient zum Ändern der Parameter Mode und PrepareMode der durch Id festgelegten LIN Botschafts-Antwort, die anschließend MessageCount-mal gesendet werden kann.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1	Mode	0: LIN Botschafts-Antwort nicht senden 1: LIN Botschafts-Antwort senden
2	PrepareMode	0: LIN Botschafts-Antwort nicht vorbereiten 1: LIN Botschafts-Antwort vorbereiten
3	MessageCount	0: LIN Botschafts-Antwort immer senden 1 ≤ N ≤ 255: LIN Botschafts-Antwort N-mal senden

4.3.19 0x33 LIN Botschafts-Antwort Daten ändern

Dieser Befehl dient zum Ändern der Parameter Dlc und Data der durch Id festgelegten LIN-Botschafts-Antwort, die anschließend MessageCount-mal gesendet werden kann.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1	reserved	Reserviert
2	MessageCount	0: LIN Botschafts-Antwort immer senden 1 ≤ N ≤ 255: LIN Botschafts-Antwort N-mal senden
3	Dlc	Datenlänge (0..8)
4..11	Mask[0..7]	Maskenbytes 0..7
12..19	Data[0..7]	Datenbytes 0..7 Daten werden an entsprechend gesetzten Maskenbyte-Bit-Positionen übernommen (Bei nicht gesetzten Masken-Byte-Bits die ursprünglichen Daten)

4.3.20 0x34 LIN Vorbereitete Botschafts-Antworten starten

Nach Aufruf dieses Befehls werden alle LIN Botschafts-Antworten gesendet, bei denen der Parameter PrepareMode auf 1 steht.

Der Befehl besitzt keine Befehlsbytes.

4.3.21 0x35 LIN Vorbereitete Botschafts-Antworten stoppen

Nach Aufruf dieses Befehls wird das Senden aller LIN Botschafts-Antworten beendet, bei denen der Parameter PrepareMode auf 1 steht.

Der Befehl besitzt keine Befehlsbytes.

4.3.22 0x3A LIN Botschaftszähler definieren

Dieser Befehl dient zum Definieren eines Botschaftszählers innerhalb eines bestimmten Datenbytes der Daten einer LIN-Botschafts-Antwort. Der Zähler wird bei jedem Senden der LIN-Botschafts-Antwort um den Wert 1 erhöht.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1	Mode	0: Zähler deaktivieren 1: Zähler aktivieren
2, 3	reserved	Reserviert
4	Byte	Byteposition, an der der Zähler innerhalb der LIN Botschafts-Antwort beginnt (0..7)
5	Bit	Bitposition, an der der Zähler innerhalb des Byte beginnt (0..7)
6	BitLength	Bitlänge des Zählers (1..8)
7	StartValue	Startwert des Zählers

Die Befehlsbytes 4 bis 7 sind im Mode = 0 irrelevant und können entfallen.

4.3.23 0x40 LIN Bus BaudRate setzen

Mit diesem Befehl kann der Parameter BaudRate gesetzt werden, ohne den kompletten Befehl 0x14 LIN Interface Eigenschaften setzen ausführen zu müssen.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	BaudRate	BaudRate in Baud (i. Allg. 19 200); Wird als BaudRate eine 0 übergeben, wird die automatische Baudraten-Erkennung aktiviert

Der Wertebereich für die BaudRate ist 700..12.5000.
Das entspricht einer minimalen Baudrate von 700Baud
und einer maximalen Baudrate von 125KBaud.

4.3.24 0x46 LIN BreakDetection-Threshold setzen

Mit diesem Befehl kann der Parameter BreakDetectionThreshold gesetzt werden.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	BreakDetection-Threshold	BreakDetectionThreshold in Prozent der Bitzeit (i. Allg. 950) Break-Erkennungs-Schwelle für GÖPEL electronic Firmware: 9,5 Bit-Zeiten (entgegen der LIN-Spezifikation mit 11 Bit-Zeiten) für Bus-Monitoring, um auch kürzere Breaks als 11 Bitzeiten zu erkennen

4.3.25 0x47 LIN WakeUpDelimiter-Time setzen

Mit diesem Befehl kann der Parameter WakeUpDelimiterTime gesetzt werden.

Die WakeUpDelimiterTime bestimmt, wann die Master-Task (falls sie aktiviert ist) nach Ende des dominanten Pegels eines WakeUps wieder mit dem Abarbeiten der Schedule-Tabelle und somit mit dem Senden beginnt.

Laut LIN 2.0 Spezifikation sollen alle Slaves 100ms nach einem WakeUp bereit sein, LIN Botschaften zu empfangen. Demzufolge sollte die Master-Task 100ms nach Ende des WakeUps die Kommunikation wieder aufnehmen.

Befehl:

Byte	Bezeichnung	Bedeutung
0..3	WakeUpDelimiterTime	WakeUpDelimiterTime in Vielfachen von 25ns, (z.B. 4 000 000 für eine WakeUpDelimiterTime von 100ms) 0: Default wake-up-time

Wird als WakeUpDelimiterTime eine Null übergeben, nutzt die Firmware die Default WakeUpDelimiterTime von vier Bit-Zeiten.

4.3.26 0x50 LIN Monitor – Steuerung

Dieser Befehl dient zur Steuerung des Monitors. Der Befehl unterteilt sich in mehrere Unterbefehle, welche durch den Parameter Mode unterschieden werden.

Alle Unterbefehle besitzen bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	Mode	0: Steuern des Monitor-Einschaltverhaltens 1: Abfragen der aktuellen Monitor-Zeit
1..3	reserved	Reserviert

Die folgenden Befehls-Parameter gelten nur für Mode = 0:

Byte	Bezeichnung	Bedeutung
4..7	Flags	Bit 0: DisableStartingOfMonitorTimeWithZero (Der Monitor startet nicht mit Zeitstempel = 0 für den Einschaltzeitpunkt, die interne Monitor-Zeit wird also NICHT rückgesetzt) Bit 1: DontDiscardTooEarlyMonitorTimeStamps (Monitor-Einträge mit einem Zeitstempel, der vor dem Einschaltzeitpunkt des Monitors liegt, werden NICHT verworfen, Beispiel: Eine LIN-Botschaft wird gerade gesendet, und genau während des Sendens wird der Monitor eingeschaltet) Bits 2..31: Reserviert
8..11	reserved	Reserviert

Für Mode = 0 existieren keine weiteren Antwort-Parameter.

Die folgenden Befehls-Parameter gelten nur für Mode = 1:

Byte	Bezeichnung	Bedeutung
4..7	reserved	Reserviert

Die folgenden Antwort-Parameter gelten nur für Mode = 1:

Byte	Bezeichnung	Bedeutung
4..7	Flags	Bit 0: MonitorIsStarted (Der Monitor läuft) Bits 1..31: Reserviert
8..11	MonitorTimeOfStart	Monitor-Zeit während des Monitor-Einschaltzeitpunktes
12..15	CurrentMonitorTime	Aktuelle Monitor-Zeit
16..19	reserved	Reserviert

4.3.27 0x52 LIN Monitor – Empfangsfilter definieren

Mit diesem Befehl können bestimmte Identifier mit dem LIN Monitor erfasst werden.

Wenn der Filter aktiv ist, werden
alle Identifier zwischen StartId und EndId (Mode = 1) bzw.
alle durch Ids angegebenen Identifier (Mode = 2) gefiltert.

Ist der Filter inaktiv, werden alle Identifier „durchgelassen“.

Soll nur ein Identifier gefiltert werden, ist bei Mode = 1 StartId gleich EndId zu setzen.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Mode	0: Kein Filter 1: Einen Bereich filtern (zur Struktur siehe unten) 2: Einzelne Identifier filtern (festgelegt durch Ids, zur Struktur siehe unten)
1..3	reserved	Reserviert

Parameter für Mode = 1:

Byte	Bezeichnung	Bedeutung
4	StartId	Identifier, ab dem gefiltert wird
5	EndId	Identifier, bis zu dem gefiltert wird
6, 7	reserved	Reserviert

Parameter für Mode = 2:

Byte	Bezeichnung	Bedeutung
4	NumberOfIds	Anzahl der Identifier (N)
5..(4*N)	Ids	Identifier-Liste (pro Identifier ein Byte)

4.3.28 0x54 LIN Monitor – Aktivieren/ deaktivieren

Bei Aktivierung dieses Befehls wird unter Mode zwischen Pufferempfang und Listenempfang unterschieden.
Im Falle von Pufferempfang sind weitere Parameter einzustellen.

Bei Pufferempfang laufen die Botschaften, wie sie auf dem Bus gesendet/ vom Bus empfangen werden, nach Passieren des Monitor-Filters nacheinander in einem Ring-Puffer ein. Dieser interne Ring-Puffer kann bis etwa 1024 LIN-Botschaften zwischenspeichern.

Bei Listenempfang existiert für jeden Identifier ein Listeneintrag, der bei Empfang/ Senden des Identifiers aktualisiert wird und zu jeder Zeit gezielt durch Angabe des Identifiers abgefragt werden kann.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Mode	0: Monitor deaktivieren 1: Pufferempfang aktivieren (zur Struktur siehe unten) 2: Listenempfang aktivieren

Die folgenden Parameter sind für den Pufferempfang (Mode = 1) zusätzlich notwendig:

Byte	Bezeichnung	Bedeutung
1	BufferMode	1: Rx (empfangene LIN-Botschaften) 2: Tx (gesendete LIN-Botschaften) 3: Tx+Rx (empfangene und gesendete LIN-Botschaften) 4: WakeUp 5: WakeUp + Rx 6: WakeUp + Tx 7: WakeUp + Tx + Rx
2	AutomaticEmpty	0: Leeren des Puffers auf Anfrage 1: Automatisches Leeren des Puffers
3	Type	1: Kleine Monitoreinträge

Für Mode = 0 oder 2 sind die Bytes 1..3 reserviert (deshalb sollten sie mit 0 übergeben werden).

Wenn die Slave-Task nicht aktiviert ist, liefern die Monitor-Befehle kein Ergebnis!!!

Nach Aktivierung des Pufferempfangs mit AutomaticEmpty = 1 sendet der ausgewählte Controller automatisch empfangene LIN-Botschaften an den Host.
Daher muss der Host den Controller zyklisch auslesen. Dabei haben die Monitor-Antworten die gleiche Struktur wie die Antworten auf den Befehl

0xF2 LIN Monitor – Kleine Puffereinträge abfragen (Type = 1)

Durch das Aktivieren des Monitors mit Mode = 1 oder 2 wird der Timer zur Erzeugung des Zeitstempels StartTime auf 0 gesetzt (siehe z.B. 0xF2 LIN Monitor – Kleine Puffereinträge abfragen).

Die folgenden Befehle können zum Auslesen der Monitor-Daten genutzt werden:

Im Falle von Pufferempfang mit AutomaticEmpty = 0

0xF2 LIN Monitor – Kleine Puffereinträge abfragen.

Im Falle von Listenempfang

0xF4 LIN Monitor – Kleinen Listeneintrag abfragen.

4.3.29 0x64 LIN Sporadic Frame definieren

Mit diesem Befehl wird ein Sporadic Frame mit den zugehörigen Unconditional Frames definiert.

Sporadic Frames können nur von einem Master gesendet werden, da dieser die jeweiligen LIN Botschafts-Header sendet.

Befehl:

Byte	Bezeichnung	Bedeutung
0	SporadicFrame-ListIndex	Listenindex (beginnend mit 0), Siehe auch Befehl 0x22 LIN Schedule-Tabelle füllen
1..5	reserved	Reserviert
6, 7	NumberOf-Unconditional-Frames	Anzahl der zu einem Sporadic Frame gehörenden Unconditional Frames (N)
8.. 7+(N*4)	Unconditional-Frames	Die zu einem Sporadic Frame gehörenden Unconditional Frames

Ein Unconditional Frame ist wie folgt strukturiert (vier Bytes):

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1..3	reserved	Reserviert

So wie für die Unconditional Frames in einem LIN Description File (LDF-File) eine Liste Frames existiert, existiert ggf. eine Liste Sporadic_frames für die Sporadic Frames.

SporadicFrameListIndex ist der Index in der Liste Sporadic_frames.

4.3.30 0x65 LIN Sporadic Frame löschen

Mit diesem Befehl wird ein mit dem Befehl 0x64 LIN Sporadic Frame definieren definiertes Sporadic Frame mit den dazugehörigen Unconditional Frames wieder gelöscht.

Befehl:

Byte	Bezeichnung	Bedeutung
0	SporadicFrame-ListIndex	Listenindex (beginnend mit 0), Siehe auch Befehl 0x22 LIN Schedule-Tabelle füllen
1..3	reserved	Reserviert

4.3.31 0x66 LIN Sporadic Frame senden

Dieser nur auf einem LIN-Master mögliche Befehl erlaubt das Senden im Sporadic Frame Slot (zu Frame Slot siehe auch Abbildung 4-2).

Der Befehl ist NUR für den LIN Master verfügbar.

Befehl:

Byte	Bezeichnung	Bedeutung
0	SporadicFrame-ListIndex	Listenindex (beginnend mit 0), Siehe auch Befehl 0x22 LIN Schedule-Tabelle füllen
1	Mode	0: Deaktivieren des Sendens 1: Aktivieren des Sendens (Der Header mit UnconditionalFrameId wird nur einmalig gesendet)
2	Unconditional-FrameId	Identifier (0x00..0x3F) des zu diesem Sporadic Frame gehörenden Unconditional Frames
3	reserved	Reserviert

Das mit UnconditionalFrameId angegebene Unconditional Frame muss wie jedes normale Unconditional Frame mit dem Befehl 0x23 LIN Botschafts-Antwort Tabelle füllen oder dem Befehl 0x30 LIN Botschafts-Antwort definieren definiert werden.

4.3.32 0x67 LIN Steuern von Sporadic Frames

Dieser Befehl dient zur Steuerung eines LIN Sporadic Frames. Der Befehl unterteilt sich in mehrere Unterbefehle, welche durch den Parameter Mode unterschieden werden.

Alle Unterbefehle besitzen bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	SporadicFrame-ListIndex	Listenindex (beginnend mit 0), Siehe auch Befehl 0x22 LIN Schedule-Tabelle füllen
1	Mode	0: Abfrage des Zustandes eines Unconditional Frames innerhalb eines Sporadic Frames
2, 3	reserved	Reserviert

Die folgenden Befehls-Parameter gelten nur für Mode = 0:

Byte	Bezeichnung	Bedeutung
4	Unconditional-FrameId	Identifier des zu diesem Sporadic Frame gehörenden Unconditional Frames (0x00..0x3F)
5..7	reserved	Reserviert

Die folgenden Antwort-Parameter gelten nur für Mode = 0:

Byte	Bezeichnung	Bedeutung
4..7	Flags	Bit 0: TxPending (Der Unconditional Frame wird noch gesendet) Bits 1..31: Reserviert
8	Unconditional-FrameId	Identifier des zu diesem Sporadic Frame gehörenden Unconditional Frames (0x00..0x3F)
9..11	reserved	Reserviert

4.3.33 0x68 LIN Event Triggered Frame definieren

Mit diesem Befehl wird ein LIN Event Triggered Frame mit den zugehörigen Unconditional Frames definiert.

Dieser Befehl ist auf dem Master sowie auf allen Slaves auszuführen, die ein Event Triggered Frame senden sollen.

Befehl:

Byte	Bezeichnung	Bedeutung
0	EventTriggered-FrameListIndex	Listenindex (beginnend mit 0), Siehe auch Befehl 0x22 LIN Schedule-Tabelle füllen
1..3	reserved	Reserviert
4	EventTriggered-Frame.Id	Identifier des Event Triggered Frames (0x00..0x3F)
5	EventTriggered-Frame.Dlc	Datenlänge des Event Triggered Frames (0..8)
6	EventTriggered-Frame.Flags	Bit 0: IgnoreDlc – Ignorieren der Datenlänge Bits 1..7: Reserviert
7	EventTriggered-Frame.reserved	Reserviert
8, 9	reserved	Reserviert
10, 11	NumberOf-Unconditional-Frames	Anzahl der zu einem Event Triggered Frame gehörenden Unconditional Frames (N)
12.. 11+(N*4)	Unconditional-Frames	Die zu einem Event Triggered Frame gehörenden Unconditional Frames

Ein Unconditional Frame ist wie folgt strukturiert (vier Bytes):

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1..3	reserved	Reserviert

So wie für die Unconditional Frames in einem LIN Description File (LDF-File) eine Liste Frames existiert, existiert ggf. eine Liste Event_triggered_frames für die Event Triggered Frames.

EventTriggeredFrameListIndex ist der Index in der Liste Event_triggered_frames.

Der Master überprüft bei jedem Empfang einer LIN-Botschafts-Antwort deren Inhalt und beginnt das Senden der zu diesem Event Triggered Frame gehörenden Unconditional Frames, falls mindestens eine der folgenden Bedingungen zutrifft:

- Ein Übertragungs-Fehler ist aufgetreten (z.B. falsche Checksumme)
- Im ersten Datenbyte ist keiner der durch diesen Befehl bekannt gemachten Unconditional Frames Identifier enthalten
- Das Flag IgnoreDlc ist nicht gesetzt, und die Datenlänge der empfangenen LIN Botschafts-Antwort stimmt nicht mit der durch EventTriggeredFrame.Dlc konfigurierten Datenlänge überein

4.3.34 0x69 LIN Event Triggered Frame löschen

Mit diesem Befehl wird ein mit dem Befehl 0x68 LIN Event Triggered Frame definieren definiertes Event Triggered Frame mit den zugehörigen Unconditional Frames wieder gelöscht.

Befehl:

Byte	Bezeichnung	Bedeutung
0	EventTriggered-FrameListIndex	Listenindex (beginnend mit 0), Siehe auch Befehl 0x22 LIN Schedule-Tabelle füllen
1..3	reserved	Reserviert

4.3.35 0x6A LIN Event Triggered Frame senden

Dieser Befehl erlaubt das Senden im Event Triggered Frame Slot (zu Frame-Slot siehe auch Abbildung 4-2).

Befehl:

Byte	Bezeichnung	Bedeutung
0	EventTriggered-FrameListIndex	Listenindex (beginnend mit 0), Siehe auch Befehl 0x22 LIN Schedule-Tabelle füllen
1	Mode	0: Deaktivieren des Sendens 1: Aktivieren des Sendens, bis die mit UnconditionalFrameId angegebene LIN-Botschafts-Antwort erfolgreich und ohne Fehler auf dem Bus gesendet wurde
2	Unconditional-FrameId	Identifier des zu diesem Event Triggered Frame gehörenden Unconditional Frames (0x00..0x3F)
3	reserved	Reserviert

Das mit UnconditionalFrameId angegebene Unconditional Frame muss wie jedes normale Unconditional Frame mit dem Befehl 0x23 LIN Botschafts-Antwort Tabelle füllen oder dem Befehl 0x30 LIN Botschafts-Antwort definieren definiert werden.

4.3.36 0x6B LIN Steuern von Event Triggered Frames

Dieser Befehl dient zur Steuerung eines LIN Event Triggered Frames. Der Befehl unterteilt sich in mehrere Unterbefehle, welche durch den Parameter Mode unterschieden werden.

Alle Unterbefehle besitzen bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	EventTriggered-FrameListIndex	Listenindex (beginnend mit 0), Siehe auch Befehl 0x22 LIN Schedule-Tabelle füllen
1	Mode	0: Abfrage des Zustandes eines Unconditional Frames innerhalb eines Event Triggered Frames 1: Setzen der CollisionResolvingScheduleTable. Diese Schedule Table wird bei einer Kollision abgefahren, um die Kollision wieder aufzulösen.
2, 3	reserved	Reserviert

Die folgenden Befehls-Parameter gelten nur für Mode = 0:

Byte	Bezeichnung	Bedeutung
4	Unconditional-FrameId	Identifier des zu diesem Event Triggered Frame gehörenden Unconditional Frames (0x00..0x3F)
5..7	reserved	reserviert

Die folgenden Antwort-Parameter gelten nur für Mode = 0:

Byte	Bezeichnung	Bedeutung
4..7	Flags	Bit 0: TxPending (Der Unconditional Frame wird noch gesendet, egal ob im Event Triggered Frame Slot oder im Unconditional Frame Slot) Bits 1..31: Reserviert
8	Unconditional-FrameId	Identifier des zu diesem Event Triggered Frame gehörenden Unconditional Frames (0x00..0x3F)
9..11	reserved	Reserviert

Die folgenden Befehls-Parameter gelten nur für Mode = 1:

Byte	Bezeichnung	Bedeutung
4	ScheduleTableNumber	Nummer der Schedule Tabelle, die als CollisionResolvingScheduleTable verwendet wird
5..7	reserved	Reserviert

4.3.37 0xA0 LIN Diagnose – Konfiguration

Dieser Befehl wird zum Konfigurieren des Diagnoseprotokolls für den mit Channel festgelegten Multisession-Kanal genutzt.

Mit Type = 0 dient er zum Deaktivieren der gesamten Diagnose.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Type	Diagnose Typ: 0: Keine Diagnose 1: Diagnose im RAW Mode 2: Diagnose entsprechend LIN 2.0 (zu den erforderlichen Strukturen siehe Folgeseiten)
2	AutomaticEmpty	0: Kein automatisches Leeren des Response-Puffers 1: Automatisches Leeren des Response-Puffers (Steuergeräte-Diagnose-Antwort wird automatisch an den Host gesendet)
3	TxMethod	Sende- bzw. Scheduling-Methode für MasterRequest- und SlaveResponse-Ids 0: Diagnose-Identifier (MasterRequest-Id und SlaveResponse-Id) sind in der Schedule-Tabelle enthalten 1: Senden der MasterRequest-Id oder der SlaveResponse-Id in einem Sporadic Frame Slot, falls zu diesem Zeitpunkt nicht bereits ein Sporadic Frame gesendet wird (zu Frame-Slot siehe auch Abbildung 4-2) 2: Einmaliges Senden einer MasterRequest- oder SlaveResponse-Id am Ende der Schedule-Tabelle 3: Senden ALLER MasterRequest-Ids und SlaveResponse-Ids am Ende der Schedule-Tabelle, bis das Diagnose-Request gesendet und die Diagnose-Response komplett empfangen wurde 4: Die normale Schedule-Tabelle wird für ein Diagnose-Request und dessen zugehörige Diagnose-Response solange unterbrochen, bis alle zugehörigen MasterRequest- und SlaveResponse-Ids gesendet wurden

Für TxMethod = 2, 3, 4 wird standardmäßig ein Schedule-Delay von 192 Bit-Zeiten genutzt.

Das entspricht zum Beispiel bei einer Baudrate von 19 200 Baud einer Delay-Zeit von 10ms.
Dieses Delay ist mit dem Befehl 0xA8 LIN Diagnose – Timing ändern änderbar.

Die folgenden Parameter gelten nur für Diagnose im RAW Mode:

Byte	Bezeichnung	Bedeutung
4, 5	reserved	Reserviert
6, 7	P2max	Timeout P2max in Millisekunden (Maximale Zeit zwischen Ende des Requests und Anfang der Response, z.B. 200ms)
8, 9	P3max	Timeout P3max in Millisekunden (Maximale Zeit zwischen Ende des Requests und Anfang der Response während ResponsePending, z.B. 5 100ms)
10, 11	Repetitions	Anzahl der Wiederholungen des Requests, wenn das Steuergerät innerhalb der Timeout-Zeiten P2max bzw. P3max nicht reagiert, z.B. 2
12	DefaultMaster-Data.Enabled	0: DefaultMasterRequestFrame deaktiviert 1: DefaultMasterRequestFrame aktiviert
13..15	DefaultMasterData. reserved	Reserviert
16..23	DefaultMasterData. Data	Daten des DefaultMasterRequestFrame
24	DefaultSlaveData.-Enabled	0: DefaultSlaveResponseFrame deaktiviert 1: DefaultSlaveResponseFrame aktiviert
25..27	DefaultSlaveData.-reserved	Reserviert
28..35	DefaultSlaveData.-Data	Daten des DefaultSlaveResponseFrame
36	TesterPresent.-Enabled	0: TesterPresent deaktiviert 1: TesterPresent aktiviert
37	TesterPresent.-ResponseRequired	Response für TesterPresent wird 0: Nicht erwartet 1: Erwartet
38, 39	TesterPresent.-Cycle	Zyklus für TesterPresent in Millisekunden, z.B. 1 000ms
40..47	TesterPresent.Data	RAW-Daten des TesterPresent Dienstes
48	RxEndCondition	Ende-Erkennung von Diagnose-Antworten bei Multi-Frames: 0: Nur Single-Frames (keine Multi-Frames) 1: Leerer Slot (keine Antwort vom Slave) 2: Default Slave Response Frame 3: Gleiches Frame
49, 50	reserved	Reserviert
51	NumberOf SpecialResponses	Anzahl von speziellen Diagnose-Antworten (N)
52.. 51+(N*20)	SpecialResponses	Spezielle Diagnose-Antworten (Z.B. 0x21 - busy-RepeatRequest oder 0x23 - routineNotComplete) zur Struktur siehe Folgeseite

Ein „Eintrag“ in SpecialResponses besteht aus folgenden 20 Bytes:

Byte	Bezeichnung	Bedeutung
0..7	Mask[0..7]	Maskenbytes 0..7
8..15	Data[0..7]	Datenbytes 0..7 (Daten werden entsprechend den gesetzten Masken-Byte-Bits mit den empfangenen Daten verglichen)
16	Flags	Bit 0: Request wiederholen Bit 1: Timing ändern (P2max auf P3max) Bit 2: Default Frame Bit 3: Letzter Frame Bit 4: Ignorieren des empfangenen Frames, wenn die Daten entsprechend Mask und Data nicht übereinstimmen Bits 5..7: Reserviert
17..19	reserved	Reserviert

Jeder empfangene Diagnose-Response-Frame wird mit den SpecialResponses verglichen. Der Vergleich erfolgt durch ein logisches UND der empfangenen Daten mit Mask und anschließenden binären Vergleich des Ergebnisses mit Data.

Die folgenden Parameter gelten nur für Diagnose entsprechend LIN2.0:

Byte	Bezeichnung	Bedeutung
4	NAD	Adresse des Steuergeräts (**N**ODE **AD**DRESS)
5	reserved	Reserviert
6, 7	P2max	Timeout P2max in Millisekunden (Maximale Zeit zwischen Ende des Requests und Anfang der Response, z.B. 200ms)
8, 9	P3max	Timeout P3max in Millisekunden (Maximale Zeit zwischen Ende des Requests und Anfang der Response während ResponsePending, z.B. 5 100ms)
10, 11	Repetitions	Anzahl der Wiederholungen des Requests, wenn das Steuergerät innerhalb der Timeout-Zeiten P2max bzw. P3max nicht reagiert (z.B. 2)
12	TesterPresent. Enabled	0: TesterPresent ist deaktiviert 1: TesterPresent ist aktiviert
13	TesterPresent. ResponseRequired	Response für TesterPresent wird 0: Nicht erwartet 1: Erwartet
14, 15	TesterPresent.Cycle	Zyklus für TesterPresent in Millisekunden, z.B. 1 000ms
16..18	TesterPresent.- reserved	Reserviert
19	TesterPresent.Length	Datenlänge des TesterPresent Dienstes (1..8)
20..27	TesterPresent.Data	Daten des TesterPresent-Dienstes (beginnend mit Service ID, i. Allg. 0x3E)

Nach Auswahl eines gültigen Diagnose-Typs (Type) startet die entsprechende Diagnose-Task mit Ausführung des 0xA0 LIN Diagnose – Konfiguration Befehls.

Wird die Diagnose nicht mehr benötigt, wird nochmals der Befehl 0xA0 LIN Diagnose – Konfiguration mit Type = 0 aufgerufen.

Dadurch stoppt die entsprechende Diagnose-Task, und in Anspruch genommene Ressourcen werden wieder frei.

Insgesamt ergibt sich für die Nutzung der Diagnose folgender Befehlsablauf:

- Diagnose Type mit 0xA0 LIN Diagnose – Konfiguration auswählen,
- Diagnose mit den Diagnose-Befehlen durchführen,
- Diagnose mit 0xA0 LIN Diagnose – Konfiguration/ Type = 0 deaktivieren.

Adressierungsarten:

physical: Kommunikation mit einem einzelnen Steuergerät (Punkt-zu-Punkt-Verbindung, Unicast)

functional: Kommunikation mit einer Gruppe von Steuergeräten (Punkt-zu-Mehrpunkt-Verbindung, Broadcast)

4.3.38 0xA1 LIN Diagnose – Sitzung starten

Dieser Befehl dient zum Starten einer Diagnose-Sitzung für den durch Channel festgelegten Multisession-Kanal. Dabei wird die Diagnose-Verbindung aufgebaut.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2, 3	Length	Länge des Requests (0..(PARAM_SIZE – 4)) (Bei Länge gleich Null wird kein Request gesendet)
4.. (3+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

4.3.39 0xA2 LIN Diagnose – Request senden

Mit diesem Befehl wird ein Diagnose-Request für den durch Channel festgelegten Multisession-Kanal gesendet.
Voraussetzung ist die vorherige erfolgreiche Ausführung von 0xA1 LIN Diagnose – Sitzung starten, wobei die Diagnose-Verbindung später nicht wieder getrennt worden sein darf.

Um größere Diagnose-Requests (z.B. 1 100 Bytes) zu versenden, ist durch die begrenzte Befehlsgröße (bestimmt durch MESSAGE_SIZE) eine mehrmalige Ausführung des Befehls notwendig. Dabei sind die Parameter Concatenate und Send entsprechend zu setzen.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2	Send	0: Nicht senden (nur Puffer füllen) 1: Senden
3	Concatenate	0: Von Pufferanfang schreiben 1: Anhängen
4	Segmentation	Segmentierungs-Flag für Segmentierung auf Diagnose-Ebene 0: Request nicht segmentiert 1: Request segmentiert
5	reserved	Reserviert
6, 7	Length	Länge des Requests (1..(PARAM_SIZE – 8))
8.. (7+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

Das Flag Segmentation ist auf das Diagnose-Protokoll bezogen und darf in der Regel von einem Diagnose-Tester NICHT gesetzt werden.

4.3.40 0xA3 LIN Diagnose – Response-Puffer abfragen

Der Befehl dient zum Abfragen des Diagnose-Response-Puffers für den durch Channel festgelegten Multisession-Kanal.

Wenn eine Diagnose-Response nicht in eine einzige Host-Antwort passt, muss der Host mehrere Antworten abholen.

Die letzte dieser Antworten enthält im Parameter RemainingLength eine Null.

Außerdem sollte der Puffer solange gelesen werden, wie das Segmentation-Bit, das Busy-Bit oder das BufferNotEmpty-Bit von Flags gesetzt sind.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1..3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	Flags	Bit 0 = 0: Keine Segmentierung auf Diagnose-Ebene Bit 0 = 1: Segmentation (Segmentierung auf Diagnose-Ebene) Bit 1 = 0: Idle Bit 1 = 1: Busy (Ein Request wurde noch nicht beantwortet/ erfolgreich abgesetzt) Bit 2 = 0: Invalid (Dieser Puffereintrag ist ungültig) Bit 2 = 1: Valid (Dieser Puffereintrag ist gültig) Bit 3 = 0: BufferEmpty (Der Diagnose-Response-Puffer ist leer) Bit 3 = 1: BufferNotEmpty (Der Puffer ist noch nicht leer) Bits 4..7: Reserviert
3	State	Diagnose-Zustand 0: Nicht initialisiert 1: Keine Verbindung 2: Verbindung wird aufgebaut 3: Verbindung steht 4: Verbindung wird abgebaut
4, 5	Length	Anzahl der Response-Bytes (0..(PARAM_SIZE – 8))
6, 7	RemainingLength	Anzahl der verbleibenden Response-Bytes
8.. (7+Length)	Response	Response, bestehend aus SID (Service-Identifier) und Daten

4.3.41 0xA4 LIN Diagnose – Sitzung stoppen

Dieser Befehl stoppt eine laufende Diagnose-Sitzung für den durch Channel festgelegten Multisession-Kanal. Dabei wird die Diagnose-Verbindung abgebaut.

Zum Deaktivieren der gesamten Diagnose muss der Befehl 0xA0 LIN Diagnose – Konfiguration mit Type = 0 aufgerufen werden.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2, 3	Length	Länge des Requests (0..(PARAM_SIZE – 4)) (Bei Length = 0 wird kein Request gesendet)
4.. (3+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

4.3.42 0xA5 LIN Diagnose – Zustand abfragen

Mit diesem Befehl wird der Diagnose-Zustand des durch Channel festgelegten Multisession-Kanals abgefragt. Zusätzlich kann der Firmware-interne LastErrorCode zurückgesetzt werden.

Der Wert von LastErrorCode in der Antwort entspricht dem Wert des Firmware-internen LastErrorCode vor dessen Rücksetzen.

Der Firmware-interne LastErrorCode wird I. Allg. ohne Aufruf von 0xA5 LIN Diagnose – Zustand abfragen nach Start einer Diagnose-Sitzung mit 0xA1 LIN Diagnose – Sitzung starten sowie nach Stop einer Diagnose-Sitzung mit 0xA4 LIN Diagnose – Sitzung stoppen und Length ≠ 0 automatisch zurückgesetzt.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	ResetLastError	0: LastErrorCode nicht zurücksetzen 1: LastErrorCode zurücksetzen
2, 3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	DiagType	Diagnose Typ: 0: Keine Diagnose 1: Diagnose im RAW Mode 2: Diagnose entsprechend LIN 2.0
3	State	Diagnose-Zustand 0: Nicht initialisiert 1: Keine Verbindung 2: Verbindung wird aufgebaut 3: Verbindung steht 4: Verbindung wird abgebaut
4	Flags	Bit 0 = 0: Idle Bit 0 = 1: Busy (Request wurde noch nicht beantwortet/ erfolgreich abgesetzt) Bit 1 = 0: RxBufferEmpty (Der Diagnose-Response-Puffer ist leer) Bit 1 = 1: RxBufferNotEmpty (Der Puffer ist noch nicht leer) Bits 2..7: Reserviert
5..7	reserved	Reserviert

4.3.43 0xA8 LIN Diagnose – Timing ändern

Dieser Befehl wird zum Ändern bestimmter Diagnose Timing-Parameter für den mit Channel festgelegten Multisession-Kanal genutzt.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Schedule-Delay ändern (Relevant für TxMethod = 2, 3, 4 im Befehl 0xA0 LIN Diagnose – Konfiguration) 1: Sende-Timeout ändern (Das Sende-Timeout ist standardmäßig 1 000ms)
2, 3	reserved	Reserviert

Die folgenden Parameter sind für Mode = 0 zusätzlich notwendig:

Byte	Bezeichnung	Bedeutung
4..7	MasterRequest	Schedule-Delay für ein Master Request in Vielfachen von 25ns
8..11	SlaveResponse	Schedule-Delay für eine Slave Response in Vielfachen von 25ns

Die folgenden Parameter sind für Mode = 1 zusätzlich notwendig:

Byte	Bezeichnung	Bedeutung
4, 5	TxTimeout	Sende-Timeout in Millisekunden
6, 7	reserved	Reserviert

4.3.44 0xA9 LIN Diagnose – Steuern des Protokolls

Dieser Befehl dient zur Steuerung des LIN Diagnose-Protokolls.
Der Befehl unterteilt sich in mehrere Unterbefehle, welche durch den Parameter Mode unterschieden werden.

Alle Unterbefehle besitzen bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Ändern des Protokoll-Verhaltens
2, 3	reserved	Reserviert

Die folgenden Befehls-Parameter gelten nur für Mode = 0:

Byte	Bezeichnung	Bedeutung
4..7	Flags	Im Normalfall sind ALLE Bits = 0 Bit 0: Disable21Handling (Die negative Antwort BusyRepeatRequest wird nicht behandelt) Bit 1: Disable23Handling (Die negative Antwort RoutineNotComplete wird nicht behandelt) Bit 2: Disable78Handling (Die negative Antwort RequestCorrectlyReceived_ResponsePending wird nicht behandelt) Bit 3: Treat21As78Handling (Die negative Antwort BusyRepeatRequest wird als negative Antwort RequestCorrectlyReceived_ResponsePending behandelt) Bits 4..31: Reserviert
8..11	reserved	Reserviert

Die folgenden Antwort-Parameter gelten nur für Mode = 0:

Byte	Bezeichnung	Bedeutung
4..7	reserved	Reserviert

4.3.45 0xF2 LIN Monitor – Kleine Puffereinträge abfragen

Dieser Befehl dient zum Abfragen von kleinen LIN Monitor-Puffereinträgen.

Der Befehl besitzt keine Befehlsbytes.

Antwort:

Byte	Bezeichnung	Bedeutung
0..3	NumberOfItems	Anzahl der Monitorpuffereinträge
4..	Items	Monitorpuffereinträge (zur Struktur siehe unten)

Ein kleiner LIN Monitor-Puffereintrag besteht aus den folgenden 20 Bytes:

Byte	Bezeichnung	Bedeutung
0	Flags	Bit 0: Identifier-Paritäts-Fehler Bit 1: Checksummen-Fehler Bit 2: Inkonsistentes SyncByte Bit 3: Bit-Fehler Bit 4: Event (siehe IdCode) Bit 5: WakeUp Bit 6: Gesendete LIN Botschafts-Antwort (TX) Bit 7: Puffer-Überlauf (Buffer overrun)
1	Length	Länge der Daten inklusive Checksumme (0..9)
2	IdCode	I. Allg. das Identifier-Byte, bestehend aus Identifier + Paritätsbits; ABER, Wenn Bit 4 von Flags gesetzt ist, spezifiziert IdCode das Event: IdCode = 0 - steigende Flanke am Trigger-Eingang, IdCode = 1 - fallende Flanke am Trigger-Eingang
3..11	Data	Datenbytes und Checksumme
12..15	StartTime	Startzeitstempel als Vielfaches von 400ns
16..19	BitTimeX8	Über acht Bitzeiten gemessene Bitzeit als Vielfaches von 25ns

Data enthält die Datenbytes und die Checksumme, die unmittelbar dem letzten Datenbyte folgt.
Length = 3 zeigt zwei Bytes Daten (Data[0] und Data[1]) und ein Byte Checksumme (Data[2]) an.

4.3.46 0xF4 LIN Monitor – Kleinen Listeneintrag abfragen

Dieser Befehl dient zum Abfragen eines kleinen Monitor-Listeneintrags der durch Id festgelegten LIN-Botschaft.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1..3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Id	Identifier (0x00..0x3F)
1..3	reserved	Reserviert
4..7	FrameCounter	Frame-Zähler
8	Flags	Bit 0: Identifier-Paritäts-Fehler Bit 1: Checksummen-Fehler Bit 2: Inkonsistentes SyncByte Bit 3: Bit-Fehler Bits 4..5: Reserviert Bit 6: Gesendete LIN Botschafts-Antwort (TX) Bit 7: Reserviert
9	Length	Länge der Daten inklusive Checksumme (0..9)
10	IdCode	Identifier-Byte (bestehend aus Identifier + Paritätsbits)
11..19	Data	Datenbytes und Checksumme
20..23	StartTime	Startzeitstempel als Vielfaches von 400ns
24..27	BitTimeX8	Über acht Bitzeiten gemessene Bitzeit als Vielfaches von 25ns

Data enthält die Datenbytes und die Checksumme, die unmittelbar dem letzten Datenbyte folgt.
Length = 3 zeigt zwei Bytes Daten (Data[0] und Data[1]) und ein Byte Checksumme (Data[2]) an.

4.4 KLine Befehle

In diesem Kapitel werden die KLine Befehle für Ihre GÖPEL-Hardware beschrieben.

Die für alle Firmware-Befehle geltenden allgemeinen Angaben finden Sie unter Allgemeines zur Firmware.

Initialzustand:

Nach dem Einschalten oder einem Software-Reset befinden sich alle KLine Schnittstellen im inaktiven Zustand (HIGH-Pegel).

Die in dieser Befehlsbeschreibung für die K-Leitung benutzten Begriffe Reizung und Initialisierung haben folgende Bedeutung: Der Tester sendet ein Initialisierungsmuster, um die Kommunikation aufzubauen.

Die Grundlage für den Protokolltreiber bilden folgende Dokumente:

KWP2000:

ISO 14230-2:1999 Keyword Protocol 2000 - Part 2: Data link layer

ISO 14230-3:1999 Keyword Protocol 2000 - Part 3: Application layer

KWP1281:

Robert Bosch GmbH: Funktionsbeschreibung der Diagnose VW/Audi (Y 265 K15 383 Ausgabe 04)

ISO-9141-Ford:

Ford Automotive Operations: Global Diagnostic Specification: Part One (DS-3L5T-1A294-AA)

Protokollspezifische Bezeichnungen und Abkürzungen in der folgenden KLine Beschreibung sind diesen Dokumenten entnommen und durch Verwendung der Zeichenattribute ***fett*** und ***kursiv*** kenntlich gemacht.

Sofern Parameter als „reserviert“ gekennzeichnet sind, wird der Inhalt des Feldes ignoriert. Dennoch muss der Parameter (unter anderem aus Kompatibilitätsgründen) übergeben werden. In der Praxis müssen diese Werte mit 0 initialisiert werden.

Generell gilt bei der Kommunikation auf der K-Leitung der Grundsatz des ständigen Wechsels von „Request“ und „Response“. D.h., jeder „Request“ des Testers (hier KLine Protokolltreiber) hat eine „Response“ des Steuergerätes zur Folge. Sofern diese „Responses“ nicht der Steuerung des Protokolls dienen, werden sie an das Hostinterface weitergereicht.

Protokollspezifische Ausnahmen bezüglich dieses „Request“-„Response“-Wechsels werden durch den Protokolltreiber abgefangen. D.h., bei laufender K-Leitungs-spezifischer Kommunikation über den Protokolltreiber gilt IMMER das Prinzip des „Frage-Antwort-Spiels“!

Fehlerverhalten:

Sollten während der Bearbeitung von Befehlen kritische Fehler auftreten, die zum Beispiel zum Abbruch der Kommunikation führen, wird der Protokolltreiber in einen „sauberen“ Initialzustand versetzt. Dabei wird intern eine Fehlernummer gesetzt, die z.B. mittels 0xA5 KLine Diagnose – Zustand abfragen abgefragt werden kann.

4.4.1 0x12 KLine Init Interface

Dieser Befehl setzt die ausgewählte KLine Schnittstelle ohne Software-Reset in den Initialzustand zurück.
Zusätzlich bietet er weitere optionale Konfigurationsmöglichkeiten.
Die Auswahl der Schnittstelle erfolgt durch die Parameter TargetAddress und TargetPort im Header des Befehls.

Die Befehlsbytes sind optional. Werden keine Befehlsbytes übergeben, arbeitet die Firmware so, als wären die optionalen Befehlsbytes Null.

Befehl:

Byte	Bezeichnung	Bedeutung
0..2	reserved	Reserviert
3	BlinkMode	0: Blinken der LEDs deaktiviert (default) 1: Blinken der LEDs aktiviert

4.4.2 0x20 KLine FIFO

Dieser Befehl dient zur Konfiguration und Steuerung der KLine im FIFO-Betrieb (FIFO = **F**irst **I**n **F**irst **O**ut). Dabei existiert für das Senden der Tx-FIFO und für den Empfang der Rx-FIFO.

Im FIFO-Betrieb können keine Diagnose-Befehle (wie z.B. 0xA0 KLine Diagnose – Konfiguration) ausgeführt werden.

Der Befehl unterteilt sich in mehrere Unterbefehle, die durch den Parameter SubCmd unterschieden werden.

Alle Unterbefehle besitzen bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	SubCmd	0: RESET 1: INIT 2: WRITE_TO_TX_FIFO 3: READ_FROM_RX_FIFO 4: GET_TX_FIFO_STATE 5: GET_RX_FIFO_STATE
1..3	reserved	Reserviert

4.4.2.1 Reset

Das SubCmd = RESET dient zum kompletten Rücksetzen der FIFO-Funktionalität.
Danach können wieder Diagnose-Befehle wie z.B. 0xA0 KLine Diagnose – Konfiguration ausgeführt werden.

Der Unterbefehl besitzt keine weiteren Befehls- und Antwort-Bytes.

4.4.2.2 Init

Das SubCmd = INIT dient zur Initialisierung der FIFO-Funktionalität.

Befehl:

Byte	Bezeichnung	Bedeutung
4..7	Flags	Bits 0.. 31: Reserviert
8..11	BaudRate	Baudrate in Hertz (5 bis 1000000)
12..15	TxBufferSize	Sende-Puffer-Größe in Bytes (0 = Benutzen der Default Sendepuffer-Größe)
16..19	RxBufferSize	Empfangspuffer-Größe in Bytes (0 = Benutzen der Default Empfangspuffer-Größe)
20	UartMode	UART Mode (UART = **U**niversal **A**synchronous **R**eceiver **T**ransmitter) 0: 8-Bit Daten 1: 7-Bit Daten, 1-Bit Parität 2: 8-Bit Daten, 1-Bit Parität
21	Parity	Parität 0: gerade Parität (even) 1: ungerade Parität (odd)
22	StopBits	Stop-Bit Mode 0: 1 Stop Bit 1: 2 Stop Bits
23	Reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
4..7	DesiredBaudRate	Gewünschte Baudrate in Hertz (entspricht BaudRate des Befehls)
8..11	ActualBaudRate	tatsächlich genutzte Baudrate in Hertz

4.4.2.3 Write to Tx FIFO

Das SubCmd = WRITE_TO_TX_FIFO dient zum Schreiben von Daten-Bytes in den Tx-FIFO.

Befehl:

Byte	Bezeichnung	Bedeutung
4..7	NumberOfBytes	Anzahl von Daten-Bytes
8.. (7+NumberOfBytes)	Data	Daten-Bytes

4.4.2.4 Read from Rx FIFO

Das SubCmd = READ_FROM_RX_FIFO dient zum Lesen von Daten-Bytes aus dem Rx-FIFO.

Antwort:

Byte	Bezeichnung	Bedeutung
4..7	Flags	Bit 0: FifoNotEmpty = Es sind immer noch (also nach Auslesen des Rx-FIFOs durch diesen Befehl) Daten-Bytes im Rx-FIFO Bit 1: FifoOverrun = Zwischen dem letzten und dem aktuellen Auslesen des Rx-FIFOs sind ein oder mehrere Daten-Bytes verlorengegangen, da der Rx-FIFO voll war Bits 2.. 31: Reserviert
8..11	NumberOfBytes	Anzahl von Daten-Bytes
12.. (11+NumberOfBytes)	Data	Daten-Bytes

4.4.2.5 Get Tx FIFO State

Das SubCmd = GET_TX_FIFO_STATE dient zur Abfrage des Zustands des Tx-FIFOs.

Antwort:

Byte	Bezeichnung	Bedeutung
4..7	Flags	Bit 0: FifoIsEmpty = Der Tx-FIFO ist leer Bit 1: FifoIsFull = Der Tx-FIFO ist voll
8..11	FifoSize	Größe des Tx-FIFOs in Bytes
12..15	FifoFillingLevel	Füllstand des Tx-FIFOs in Bytes

4.4.2.6 Get Rx FIFO State

Das SubCmd = GET_RX_FIFO_STATE dient zur Abfrage des Zustands des Rx-FIFOs.

Antwort:

Byte	Bezeichnung	Bedeutung
4..7	Flags	Bit 0: FifoIsEmpty = Der Rx-FIFO ist leer Bit 1: FifoIsFull = Der Rx-FIFO ist voll Bit 2: FifoOverrun = Ein oder mehrere empfangene Daten-Bytes sind verlorengegangen, da der Rx-FIFO voll war
8..11	FifoSize	Größe des Rx-FIFOs in Bytes
12..15	FifoFillingLevel	Füllstand des Rx-FIFOs in Bytes

4.4.3 0x21 KLine Node

Dieser Befehl dient zur Konfiguration und Steuerung der KLine-Schnittstelle.

Der Befehl unterteilt sich in mehrere Unterbefehle, die durch den Parameter SubCmd unterschieden werden.

Alle Unterbefehle besitzen bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	SubCmd	0: PROPERTY__SET_BY_ID 1: PROPERTY__GET_BY_ID
1..3	reserved	Reserviert

PropertyId:

Wert	Bedeutung
0	**ENABLE_RX_TX_SPLIT** 0 = Deaktivieren der Trennung von Rx und Tx 1 = Aktivieren der Trennung von Rx und Tx Dies ermöglicht bei entsprechendem Transceiver einen Voll-Duplexbetrieb (wie z.B. bei RS232) mit jeweils einer Rx- und einer Tx-Leitung. Achtung: der „normale“ KLine-Transceiver unterstützt KEINEN Voll-Duplexbetrieb.
1	**ENABLE_BUS_TERMINATION** 0 = Deaktivieren des Busabschlusses (Pull-Up-Resistor) 1 = Aktivieren des Busabschlusses (Pull-Up-Resistor)
2	**ENABLE_INTERNAL_VBAT** 0 = Deaktivieren der internen Spannungsversorgung für den Transceiver 1 = Aktivieren der internen Spannungsversorgung für den Transceiver Dadurch kann das Anlegen einer externen Spannung entfallen.

4.4.3.1 Property SetById

Das SubCmd = PROPERTY__SET_BY_ID dient zum Setzen des Wertes einer Eigenschaft, die durch PropertyId spezifiziert ist.

Befehl:

Byte	Bezeichnung	Bedeutung
4, 5	Id	ProperyId: Eigenschafts-Identifier (siehe PropertyId im Abschnitt 0x21 KLine Node)
6, 7	Reserved	Reserviert
8..11	Value	Wert der Eigenschaft

4.4.3.2 Property GetByID

Das SubCmd = GET_FLAG_BY_ID dient zur Abfrage des Wertes einer Eigenschaft, die durch PropertyId spezifiziert ist.

Befehl:

Byte	Bezeichnung	Bedeutung
4, 5	Id	ProperyId: Eigenschafts-Identifier (siehe PropertyId im Abschnitt 0x21 KLine Node)
6, 7	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
4, 5	Id	ProperyId: Eigenschafts-Identifier (siehe PropertyId im Abschnitt 0x21 KLine Node)
6, 7	Reserved	Reserviert
8..11	Value	Wert der Eigenschaft

4.4.4 0x54 KLine Monitor

Dieser Befehl dient zur Konfiguration und Steuerung des KLine Monitors.

Zur Benutzung des Monitors muss die FIFO-Funktionalität (siehe Befehl 0x20 KLine FIFO) aktiviert oder ein Diagnose-Protokoll (siehe Befehl 0xA0 KLine Diagnose – Konfiguration) ausgewählt sein. Dadurch ist unter anderem z.B. die richtige Baudrate eingestellt.

Der Befehl unterteilt sich in mehrere Unterbefehle, die durch den Parameter SubCmd unterschieden werden.

Alle Unterbefehle besitzen bis Byte 3 den gleichen Befehls- und Antwortaufbau und unterscheiden sich erst ab Byte 4 (soweit mehr als vier Bytes vorhanden sind).

Befehl und Antwort:

Byte	Bezeichnung	Bedeutung
0	SubCmd	0: RESET 1: CONFIG__NORMAL 3: START 4: STOP 5: READ__NORMAL
1..3	reserved	Reserviert

4.4.4.1 Reset

Das SubCmd = RESET dient zum kompletten Rücksetzen und Deaktivieren des Monitors.

Der Unterbefehl besitzt keine weiteren Befehls- und Antwort-Bytes.

4.4.4.2 Config Normal

Das SubCmd = CONFIG__NORMAL dient zur Konfiguration des Monitors mit normalen Monitor Einträgen.

Befehl:

Byte	Bezeichnung	Bedeutung
4..7	Flags	Bit 0: AutomaticEmpty = Empfangene Monitor-Einträge werden automatisch an den Host gesendet, die Antworten besitzen den gleichen Aufbau wie die Antworten des Unterbefehls Read Normal Bits 1.. 31: Reserviert
8..11	BufferSize	Monitor-Puffergröße als Anzahl von Monitor-Einträgen

4.4.4.3 Start

Das SubCmd = START dient zum Starten des Monitors. Dazu muss der Monitor bereits konfiguriert sein (siehe Config Normal).

Der Unterbefehl besitzt keine weiteren Befehls- und Antwortbytes.

4.4.4.4 Stop

Das SubCmd = STOP dient zum Stoppen des Monitors.

Der Unterbefehl besitzt keine weiteren Befehls- und Antwortbytes.

4.4.4.5 Read Normal

Das SubCmd = READ_NORMAL dient zum Lesen von normalen Monitor Einträgen.

Antwort:

Byte	Bezeichnung	Bedeutung
4..7	NumberOfItems	Anzahl von normalen Monitor Einträgen
8..	Items	Normale Monitor Einträge

Aufbau eines einzelnen normalen Monitor Eintrages:

Byte	Bezeichnung	Bedeutung
0..3	Timestamp	Zeitstempel mit der Auflösung TimestampResolution
4	TimestampResolution	Zeitstempel-Auflösung 0: 400 Nanosekunden
5	Flags	Bit 0: Reserviert Bit 1: Tx = gesendetes Datenbyte Bit 2: Collision = Kollision durch gleichzeitiges Senden und Empfangen eines Datenbytes Bits 3..6: Reserviert Bit 7: BufferOverrun = Pufferüberlauf
6	Data	Daten-Byte
7	reserved	Reserviert

4.4.5 0xA0 KLine Diagnose – Konfiguration

Der Befehl wird zum Konfigurieren des Diagnoseprotokolls für den mit Channel festgelegten Multisession-Kanal genutzt.

Mit Type = 0 dient er zum Deaktivieren der gesamten Diagnose.

Dieser Firmware-Befehl kann nur genutzt werden, wenn das zu konfigurierende Diagnoseprotokoll mit 0x03 Funktionalitäten freischalten freigeschaltet worden ist.

Alle für den Aufbau und den Ablauf einer Diagnosesitzung nötigen Einstellungen werden über diesen Befehl vorgegeben. Die gesetzten Einstellungen bleiben aktiv, bis explizit neue Einstellungen gesetzt werden.

Die Gesamtlänge des Befehls beträgt 84 Bytes.
Die zu Grunde liegende Datenstruktur ist für alle Protokolle gleich.
Die Interpretation der einzelnen Felder variiert je nach gewähltem Diagnoseprotokoll und nach Typ der Initialisierung.

Allgemeingültige Parameter:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Type	Diagnose Typ: 0: Keine Diagnose 1: Diagnose KWP2000 2: Diagnose KWP1281 3: Diagnose ISO-9141-Ford Zu den erforderlichen Strukturen siehe Folgeseiten
2, 3	reserved	Reserviert
4, 5	Flags	Automatisches Senden von Diagnoseantworten an den Host Bit 0 = 0: deaktivieren Bit 0 = 1: aktivieren Bits 1..4: Reserviert Verifizieren der Prüfsumme (KWP2000 und ISO-9194-Ford) Bit 5 = 0: deaktivieren Bit 5 = 1: aktivieren Bits 6..15: Reserviert
6, 7	reserved	Reserviert

Ist die Verifizierung der Prüfsumme abgeschaltet, wird der Wert des Prüfsummenfeldes ignoriert, andernfalls erfolgt eine Verifizierung.
Ungültige Prüfsummen führen dann zu „invalid checksum“-Fehlern.

Parametrierung Keyword Protocol 2000 (KWP2000):

Byte	Bezeichnung	Bedeutung
8, 9	SourceAddress	Adresse des Testers zur Verwendung während der Diagnose, z.B. 0xF1
10, 11	TargetAddress	Adresse des Steuergerätes zur Verwendung während der Diagnose
12, 13	P1min	Minimale Interbytezeit bei Antworten vom SG, z.B. 0ms
14, 15	P1max	Maximale Interbytezeit bei Antworten vom SG, z.B. 20ms
16, 17	P2min	Minimale Zeit zwischen Anforderung vom Tester und Antwort vom SG, oder minimale Zeit zwischen zwei Antworten vom SG, z.B. 25ms
18, 19	P2max	Maximale Zeit zwischen Anforderung vom Tester und Antwort vom SG, oder maximale Zeit zwischen zwei Antworten vom SG, z.B. 50ms
20, 21	P3min	Minimale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 55ms
22, 23	P3max	Maximale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 2000ms
24, 25	P4min	Minimale Interbytezeit bei Anforderungen vom Tester, z.B. 5ms
26, 27	P4max	Maximale Interbytezeit bei Anforderungen vom Tester, z.B. 20ms
28, 29	TesterPresent.-SourceAddress	Adresse des Testers zur Verwendung während des ***TesterPresent*** Service, z.B. wie SourceAddress
30, 31	TesterPresent.-TargetAddress	Adresse des Steuergerätes zur Verwendung während des ***TesterPresent*** Service, z.B. wie TargetAddress
32	TesterPresent.-UseResponseRequired-Parameter	Der ***Tester Present ResponseRequired*** Parameter wird 0: Nicht verwendet 1: Verwendet
33	TesterPresent.ResponseRequiredParameter	Wert des ***TesterPresent Response Required*** Parameters (falls verwendet), sonst 0
34, 35	reserved	Reserviert
36, 37	InitType	Typ der Initialisierung: 0: 5 Baud Reizung 1: Schnelle Reizung
38, 39	reserved	Reserviert

Parametrierung KWP2000 für 5 Baud Reizung:

Byte	Bezeichnung	Bedeutung
40, 41	reserved	Reserviert
42, 43	TargetAddress	5 Baud Adresse des Steuergerätes
44, 45	W1min	Minimale Zeit zwischen Ende des Adressbytes und Start des Synchronisationsmusters, z.B. 60ms
46, 47	W1max	Maximale Zeit zwischen Ende des Adressbytes und Start des Synchronisationsmusters, z.B. 300ms
48, 49	W2min	Minimale Zeit zwischen Ende des Synchronisationsmusters und Start des ***Keybyte 1***, z.B. 5ms
50, 51	W2max	Maximale Zeit zwischen Ende des Synchronisationsmusters und Start des ***Keybyte 1***, z.B. 20ms
52, 53	W3min	Minimale Zeit zwischen ***Keybyte 1*** und ***Keybyte 2***, z.B. 0ms
54, 55	W3max	Maximale Zeit zwischen ***Keybyte 1*** und ***Keybyte 2***, z.B. 20ms
56, 57	W4min	Minimale Zeit zwischen ***Keybyte 2*** vom SG und seiner Invertierung vom Tester sowie minimale Zeit zwischen invertiertem ***Keybyte 2*** vom Tester und invertiertem Adressbyte vom SG, z.B. 25ms
58, 59	W4max	Maximale Zeit zwischen ***Keybyte 2*** vom SG und seiner Invertierung vom Tester sowie maximale Zeit zwischen invertiertem ***Keybyte 2*** vom Tester und invertiertem Adressbyte vom SG, z.B. 50ms
60, 61	W5min	Minimale Zeit bevor der Tester beginnt das Adressbyte zu senden, z.B. 300ms
62, 63	W5max	Maximale Zeit bevor der Tester beginnt das Adressbyte zu senden, z.B. 300ms
64..71	reserved	Reserviert
72, 73	Parity	Parität beim Senden des Adressbytes: 0: even (gerade) 1: odd (ungerade)
74, 75	reserved	Reserviert

Parametrierung KWP2000 für schnelle Reizung:

Byte	Bezeichnung	Bedeutung
40, 41	SourceAddress	Adresse des Testers zur Verwendung während der Initialisierung, z.B. 0xF1
42, 43	TargetAddress	Adresse des Steuergerätes zur Verwendung während der Initialisierung
44, 45	W5min	Minimale Zeit bevor der Tester beginnt das Adressbyte zu senden, z.B. 300ms
46, 47	W5max	Maximale Zeit bevor der Tester beginnt das Adressbyte zu senden, z.B. 300ms
48, 49	TWuPmin	Minimale Zeitdauer für das Wake up Pattern, z.B. 50ms
50, 51	TWuPmax	Maximale Zeitdauer für das Wake up Pattern, z.B. 50ms
52, 53	TIniLmin	Minimale Zeitdauer für die Low-Phase des Wake up Pattern, z.B. 25ms
54, 55	TIniLmax	Maximale Zeitdauer für die Low-Phase des Wake up Pattern, z.B. 25ms
Die nachfolgenden Parameter sind gültig bis zum Abschluss der Reizung, d.h., über diese Parameter kann das Zeitverhalten des Protokolls während der Reizung gesondert eingestellt werden.		
56, 57	P1min	Minimale Interbytezeit bei Antworten vom SG, z.B. 0ms
58, 59	P1max	Maximale Interbytezeit bei Antworten vom SG, z.B. 20ms
60, 61	P2min	Minimale Zeit zwischen Anforderung vom Tester und Antwort vom SG, z.B. 25ms
62, 63	P2max	Maximale Zeit zwischen Anforderung vom Tester und Antwort vom SG, z.B. 50ms
64, 65	P3min	Minimale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 55ms
66, 67	P3max	Maximale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 2 000ms
68, 69	P4min	Minimale Interbytezeit bei Anforderungen vom Tester, z.B. 5ms
70, 71	P4max	Maximale Interbytezeit bei Anforderungen vom Tester, z.B. 20ms
72, 73	BaudRate	BaudRate (i. Allg. 10 400Hz)
74, 75	reserved	Reserviert

Parametrierung KWP2000 (Fortsetzung)

76, 77	BusyRepeatRequest-Max	Maximale Anzahl ***Busy Repeat Request (0x21)*** Antworten auf einen Request Bei Überschreiten der maximalen Anzahl wird die Antwort mit dem Fehlercode an den Host gegeben – die Anforderung wird nicht wiederholt. 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
78, 79	RoutineNotComplete-Max	Maximale Anzahl ***Routine Not Complete (0x23)*** Antworten auf einen Request Bei Überschreiten der maximalen Anzahl wird die Antwort mit dem Fehlercode an den Host gegeben – die Anforderung wird nicht wiederholt. 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
80, 81	RequestCorrectly-ReceivedResponse-PendingMax	Maximale Anzahl ***Request Correctly Received Response Pending (0x78)*** Antworten auf einen Request Bei Überschreiten der maximalen Anzahl wird die Kommunikation abgebrochen und ein NO_RESPONSE Fehler generiert. 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
82	Flags	Bit 0 = 0: Kein separates Längenbyte im Header des Frames Bit 0 = 1: Separates Längenbyte im Header des Frames Bit 1 = 0: Target- und Source-Byte im Header des Frames Bit 1 = 1: Keine Adress-Bytes im Header des Frames Bits 2..7: Reserviert
83	reserved	Reserviert

Parametrierung Keyword Protocol 1281 (KWP1281):

Byte	Bezeichnung	Bedeutung
8..11	reserved	Reserviert
12, 13	t7min	Minimale Zeit zwischen Bytes innerhalb eines Blocks, z.B. 1ms
14, 15	t7max	Maximale Zeit zwischen Bytes innerhalb eines Blocks, z.B. 55ms
16, 17	t8min	Minimale Zeit für erneuten Empfang des ersten Bytes eines Blocks (falls der Slave das letzte Byte eines Blocks nicht empfangen hat), z.B. 1ms
18, 19	t8max	Maximale Zeit für erneuten Empfang des ersten Bytes eines Blocks (falls der Slave das letzte Byte eines Blocks nicht empfangen hat), z.B. 200ms
20, 21	t9min	Minimale Zeit zwischen Ende eines Blocks und Start des nächsten Blocks, z.B. 1ms
22, 23	t9max	Maximale Zeit zwischen Ende eines Blocks und Start des nächsten Blocks, z.B. 500ms
24..35	reserved	Reserviert
36, 37	InitType	Typ der Initialisierung 0: 5 Baud Reizung
38..41	reserviert	Reserviert
42, 43	TargetAddress	5 Baud Adresse des Steuergerätes
44, 45	t0min	Minimale Idle Line vor Beginn der Reizung, z.B. 60ms
46, 47	t0max	Maximale Idle Line vor Beginn der Reizung, z.B. 300ms
48, 49	t1min	Minimale Zeit zwischen korrekter Reizung und Start des Synchronbytes, z.B. 80ms
50, 51	t1max	Maximale Zeit zwischen korrekter Reizung und Start des Synchronbytes, z.B. 210ms
52, 53	t2min	Minimale Zeit zwischen Synchronbyte und ***Keybyte 1***, z.B. 5ms
54, 55	t2max	Maximale Zeit zwischen Synchronbyte und ***Keybyte 1***, z.B. 20ms
56, 57	t3min	Minimale Zeit zwischen ***Keybyte 1*** und ***Keybyte 2***, z.B. 1ms
58, 59	t3max	Maximale Zeit zwischen ***Keybyte 1*** und ***Keybyte 2***, z.B. 20ms
60, 61	t4min	Minimale Zeit zwischen ***Keybyte 2*** und Komplement von ***Keybyte 2***, z.B. 25ms
62, 63	t4max	Maximale Zeit zwischen ***Keybyte 2*** und Komplement von ***Keybyte 2***, z.B. 50ms
64, 65	t5min	Minimale Zeit zwischen Komplement von ***Keybyte 2*** und erneuter Ausgabe des Synch.-Bytes (falls das Komplement von ***Keybyte 2*** vom Steuergerät falsch empfangen wurde), z.B. 240ms
66, 67	t5max	Maximale minimale Zeit zwischen Komplement von ***Keybyte 2*** und erneuter Ausgabe des Synch.-Bytes (falls das Komplement von ***Keybyte 2*** vom Steuergerät falsch empfangen wurde), z.B. 1 000ms
68, 69	t6min	Minimale Zeit zwischen Komplement von ***Keybyte 2*** und Start der SG-Identifikation, z.B. 25ms
70, 71	t6max	Maximale Zeit zwischen Komplement von ***Keybyte 2*** und Start der SG-Identifikation, z.B. 50ms

72, 73	Parity	Parität beim Senden des Adressbytes 0: even (gerade) 1: odd (ungerade)
74, 75	reserved	Reserviert
76, 77	MasterMaxBlockRetry	Maximale Anzahl von Neuversuchen bei Fehlern während des Sendens eines Blocks, z.B. 5 Vorgabe der maximalen Anzahl von Wiederholungsversuchen einen Block abzusetzen, wenn beim Senden des Blocks (Protokolltreiber ist Master) Fehler auftreten (Fehler z.B. kein oder fehlerhaftes Echo vom Slave, NO_ACK-1 vom Slave) 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
78, 79	SlaveMaxBlockRetry	Maximale Anzahl von Neuversuchen bei Fehlern während des Empfangs eines Blocks, z.B. 5 Vorgabe der maximalen Anzahl von Wiederholungsversuchen einen Block zu empfangen, wenn beim Empfang des Blocks (Protokolltreiber ist Slave) Fehler auftreten (Fehler z.B. Timeout beim Empfang des nächsten Bytes innerhalb eines Blocks vom Master) 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
80..83	reserved	Reserviert

Parametrierung ISO-9141-Ford:

Byte	Bezeichnung	Bedeutung
8, 9	SourceAddress	Adresse des Testers zur Verwendung während der Diagnose, z.B. 0xF1
10, 11	TargetAddress	Adresse des SG zur Verwendung während der Diagnose
12, 13	ReceiveInterByte-GapMin	***Tester Reception: Interbyte Gap Time min*** Minimale Interbytezeit bei Antworten vom SG, z.B. 0ms
14, 15	ReceiveInterByte-GapMax	***Tester Reception: Interbyte Gap Time max*** Maximale Interbytezeit bei Antworten vom SG, z.B. 22ms
16, 17	ResponseInterMsg-GapMin	***ECU Response Following A Tester Request & ECU Response Following Another ECU Message In A Sequence: Intermessage Gap Time min*** Minimale Zeit zwischen Anforderung vom Tester und Antwort vom SG, oder minimale Zeit zwischen zwei Antworten vom SG, z.B. 0ms
18, 19	ResponseInterMsg-GapMax	***ECU Response Following A Tester Request & ECU Response Following Another ECU Message In A Sequence: Intermessage Gap Time max*** Maximale Zeit zwischen Anforderung vom Tester und Antwort vom SG, oder maximale Zeit zwischen zwei Antworten vom SG, z.B. 50ms
20, 21	RequestInterMsg-GapMin	***Tester Request Following An ECU Response: Intermessage Gap Time min*** Minimale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 55ms
22, 23	RequestInterMsg-GapMax	***Tester Request Following An ECU Response: Intermessage Gap Time max*** Maximale Zeit zwischen Antwort vom SG und neuer Anforderung vom Tester, z.B. 2 000ms
24, 25	TransmitInterByte-GapMin	***Tester Transmissions: Interbyte Gap Time min*** Minimale Interbytezeit bei Anforderungen vom Tester, z.B. 6ms
26, 27	TransmitInterByte-GapMax	***Tester Transmissions: Interbyte Gap Time max*** Maximale Interbytezeit bei Anforderungen vom Tester, z.B. 6ms
28..35	reserved	Reserviert
36, 37	InitType	Typ der Initialisierung: 2: keine spezielle Initialisierung notwendig
38..75	reserviert	Reserviert
76, 77	BusyRepeatRequest-Max	Maximale Anzahl von ***Busy Repeat Request (0x21)*** Antworten auf einen Request Bei Überschreiten der maximalen Anzahl wird die Antwort mit dem Fehlercode an den Host gegeben – der Request wird nicht wiederholt. 0x0000..0xFFFE: Anzahl 0xFFFF: unbegrenzt
78..83	reserviert	Reserviert

Hinweise zur Parametrierung des Tester Present-Service:

Der sogenannte ***Tester Present*** Service (bei KWP1281 als „Austausch von Acknowledge Blöcken“ bezeichnet) dient dem Aufrechterhalten der Kommunikation. D.h., falls über einen bestimmten Zeitraum keine Requests vom Host beim Protokolltreiber (Tester) eintreffen, muss dieser durch das Versenden bestimmter Nachrichten den Kommunikationsabbau (SG geht in Timeout) verhindern.

Dabei ist die maximale Zeitspanne zwischen Antwort vom SG und neuer Anforderung vom Tester der entscheidende Parameter (KWP2000: P3max,
KWP1281: t9max,
ISO-9141-Ford: RequestInterMsgGapMax).

Die entsprechende Nachricht wird durch den Protokolltreiber jeweils kurz vor Ablauf der durch den Host vorgegebenen Zeitspanne generiert.

Adressierungsarten:

physical: Kommunikation mit einem Steuergerät (Punkt-zu-Punkt-Verbindung, Unicast)

functional: Kommunikation mit einer Gruppe von Steuergeräten (Punkt-zu-Mehrpunkt-Verbindung, Broadcast)

4.4.6 0xA1 KLine Diagnose – Sitzung starten

Dieser Befehl dient zum Starten einer Diagnose-Sitzung für den durch Channel festgelegten Multisession-Kanal. Dabei wird die Diagnose-Verbindung aufgebaut.

Voraussetzung für die Ausführung des Befehls ist die vorherige Ausführung von 0xA0 KLine Diagnose – Konfiguration.

Dieser Befehl ist Voraussetzung für das Absetzen eines Requests (0xA2 KLine Diagnose – Request senden). Die Diagnose bleibt bestehen, bis sie explizit mit 0xA4 KLine Diagnose – Sitzung stoppen wieder beendet wird. Nach erfolgreichem Diagnoseaufbau (Reizung) wird der ***Tester Present*** Service (***ACK Block*** Austausch bei KWP1281) aktiv, sobald auf der K-Leitung das „Idle- Timeout“ überschritten wird (d.h., nachdem für eine bestimmte Zeit vor Ablauf der maximalen Interframe- bzw. Interblockzeit kein Request vom Tester kam).

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig.
2, 3	Length	Länge des Requests Zur Zeit wird nur Length = 0 unterstützt und fest je nach Diagnose-Typ der entsprechende Start-Service geschickt.
4.. (3+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

Der Befehl 0xA1 KLine Diagnose – Sitzung starten zum Starten der Diagnose (bzw. Eröffnen der Kommunikation) ist von der Hostseite gesehen auf den ersten Blick für alle K-Leitungs Protokolle identisch. Innerhalb des K-Leitungs Treibers werden jedoch spezifische, zum jeweilig aktiven Protokoll passende Aktionen ausgelöst. Alle Protokolle reagieren im Rahmen der verschiedenen Eröffnungsvarianten unterschiedlich.

Generell gilt: Der K-Leitungs Protokolltreiber liefert immer eine Antwort auf den 0xA1 KLine Diagnose – Sitzung starten Befehl (entweder automatisch oder über Abfrage mit 0xA3 KLine Diagnose – Response-Puffer abfragen, je nach eingestelltem Antwortmodus)!

Die Bedeutung der Antwortdaten ist jedoch je nach Protokoll unterschiedlich. Die richtige Interpretation liegt in der Verantwortung des Empfängers (Host).

Die folgende Tabelle soll die Vorgänge innerhalb des Protokolltreibers in Reaktion auf ein 0xA1 Diagnose – Sitzung starten verdeutlichen.

Zu beachten ist, dass es auf der K-Leitung keine echte Trennung zwischen Diagnose- und Transportprotokoll gibt.
Vereinfacht gesagt ist „Starten der Kommunikation“ auf der K-Leitung gleichzusetzen mit „Starten der Diagnose“.

KWP2000 (schnelle Reizung)	
Beschreibung	Response vom KLine Treiber
StartCommunication ist hier ein „normaler“ Request (KWP2000 Frame mit drei Byte Header, ***SID*** = 0x81 – „***StartCommunication***“), der nach einer bestimmten „idle“- Zeit und einem speziellen ***Wake up Pattern (WuP)*** auf der K-Leitung mit der voreingestellten Baudrate gesendet wird. Das Steuergerät reagiert im Erfolgsfall mit einem Response Frame (Form des Headers je nach Fähigkeit des Steuergerätes, ***SID*** = 0xC1, zwei ***Keybytes*** im Datenteil). Danach gilt die Kommunikation als eröffnet, d.h. der ***Tester Present*** Service wird aktiv, falls keine Requests vorliegen.	Datenteil des vom SG empfangenen Response Frames mit den zwei ***Keybytes***. ACHTUNG: Mit ***Keybyte 1*** gibt das Steuergerät Auskunft über seine Fähigkeiten.

KWP2000 (langsame Reizung – 5 Baud)	
Beschreibung	Response vom KLine Treiber
Nach einer „idle“-Zeit (Busruhe) wird die spezielle „Reizadresse“ mit 5 Baud gesendet. Darauf reagiert das Steuergerät im Erfolgsfall mit Ausgabe eines Musters, welches es dem Tester (KLine-Treiber) ermöglicht, sich auf die Baudrate des Steuergerätes zu synchronisieren. Nachfolgend sendet das SG zwei ***Keybytes***. Den Empfang der Keybytes bestätigt der Tester seinerseits, indem er das ***Keybyte 2*** bitweise invertiert an das SG zurück sendet. Abschließend sendet das SG die „Reizadresse“ bitweise invertiert zurück. Danach gilt die Kommunikation als eröffnet, d.h. Austausch von ***Tester Present*** Blöcken, falls keine Requests vorliegen. Achtung: jetzt evtl. abweichende Adresse des SG.	Der KLine Treiber generiert aus den im Rahmen der Reizung empfangenen ***Keybytes*** eine Response identisch zu „KWP2000 (schnelle Reizung)“. Damit gibt es keine Unterschiede (von der Hostseite) zur schnellen Reizung.

KWP1281 (langsame Reizung – 5 Baud)	
Beschreibung	Response vom KLine Treiber
Ähnlich KWP2000 (langsame Reizung), jedoch beginnt das Steuergerät statt die bitweise invertierte Adresse zu senden nach dem Empfang des ***Keybyte 2*** -Komplements automatisch mit der Ausgabe seines Identifikations-Strings entsprechend der Protokollvorschriften nach KWP1281. Diese Identifikation verteilt sich unter Umständen auf mehrere Blöcke. Danach gilt die Kommunikation als eröffnet, d.h. Austausch von ***Acknowledge*** Blöcken, falls keine Requests vorliegen.	KLine Treiber gibt die empfangene SG-ID als Response an das Hostinterface. ACHTUNG: Die ***Keybytes*** treffen bei KWP1281 keinerlei Aussage über das SG (immer identisch).

ISO-9141-Ford	
Beschreibung	Response vom KLine Treiber
Tester (KLine Treiber) generiert und sendet einen „normalen“ Request (***Mode*** = 0x10 – ***Diagnostic Mode Entry***) mit 10 400 Baud und den vorgegebenen Adressparametern auf der K-Leitung. Im Erfolgsfall reagiert das Steuergerät darauf mit einem ***General Response*** Block (***Mode*** = 0x7F, ***Response Code*** = 0x00). Danach gilt die Kommunikation als eröffnet, d.h. ***Tester Present*** Service wird aktiv, falls keine Requests vorliegen.	Datenteil des vom SG gesendeten ***General Response*** Blocks.

4.4.7 0xA2 KLine Diagnose – Request senden

Dieser Befehl dient zum Senden einer Diagnose-Anforderung für den durch Channel festgelegten Multisession-Kanal.

Voraussetzung ist die vorherige erfolgreiche Ausführung von 0xA1 KLine Diagnose – Sitzung starten, wobei die Diagnose-Verbindung später nicht wieder getrennt worden ist.

Die Antwort auf diesen Request (Response) wird je nach Einstellung (Bit 0 im Parameter Flags beim Befehl 0xA0 KLine Diagnose – Konfiguration) entweder automatisch an den Host zurückgesendet oder muss mittels 0xA3 KLine Diagnose – Response-Puffer abfragen abgefragt werden.

Beim Request werden nur die tatsächlichen Nutzdaten des vom Protokolltreiber zu bildenden Telegramms übergeben (Bei KWP2000, ISO-9141-Ford: kein Header, keine Prüfsumme – nur ***ServiceID*** (bzw. ***MODE*** Byte) und Daten. Bei KWP1281: keine Blocklänge, kein Blockzähler, kein ETX- Blockendebyte – nur Blocktitel und Daten).

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2	Send	0: Nicht senden (nur Puffer füllen) 1: Senden
3	Concatenate	0: Von Pufferanfang schreiben 1: Anhängen
4	Segmentation	Segmentierungs-Flag für Segmentierung auf Diagnose-Ebene 0: Request nicht segmentiert 1: Request segmentiert
5	reserved	Reserviert
6, 7	Length	Länge des Requests (1..(PARAM_SIZE – 8)) (Bei Länge gleich Null wird kein Request gesendet)
8.. (7+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

Das Flag Segmentation ist auf das Diagnose-Protokoll bezogen und darf in der Regel von einem Diagnose-Tester nicht gesetzt werden.

Mit EINEM 0xA2 Kline Diagnose – Request senden Befehl werden höchstens PARAM_SIZE – 8 Request Bytes gesendet.

Um größere Diagnose-Requests (z.B. 1 100 Bytes) zu versenden, ist durch die begrenzte Befehlsgröße (MESSAGE_SIZE) eine mehrmalige Ausführung des Befehls notwendig.
Dabei sind die Flags Concatenate und Send entsprechend zu setzen.

Beispiel zur Segmentierung von Host-Requests

Anhand des folgenden Beispiels soll die Segmentierung von Befehlen an den Treiber demonstriert werden. Es wird angenommen, dass bereits erfolgreich eine KWP2000- Diagnose eröffnet wurde.
Der Befehl **„0x1A, 0x9B“** -- „Steuergeräte Identifikation lesen (***ReadECUIdentification Service***)“ soll abgesetzt werden.

Variante 1) monolithischer Befehl:

Zeitpunkt t1:
Request (0xA2)-Telegramm vom Host an Protokolltreiber

Befehls-Header mit *OpCode* = 0xA2	**(u8)** 0x00 *Channel* = 0	**(u8)** 0x00 *Mode* = 0	**(u8)** 0x01 *Send* = 1	**(u8)** 0x00 *Concatenate* = 0	**(u8)** 0x00 *Segmentation* = 0	**(u8)** 0x00	**(u16)** 0x0002 *Length* = 2	**(u8)** 0x1A	**(u8)** 0x9B

Zeitpunkt t2 (t2 = t1 + x):
KWP2000-Telegramm wird vom Protokolltreiber gebildet und gesendet

KWP2000 Header	**(u8)** 0x1A	**(u8)** 0x9B	**(u8)** KWP2000 Prüfsumme

Variante 2) segmentierter Befehl:

Zeitpunkt t1:
Request (0xA2)-Telegramm vom Host an Protokolltreiber

Befehls-Header mit *OpCode* = 0xA2	**(u8)** 0x00 *Channel* = 0	**(u8)** 0x00 *Mode* = 0	**(u8)** 0x00 *Send* = 0	**(u8)** 0x00 *Concatenate* = 0	**(u8)** 0x00 *Segmentation* = 0	**(u8)** 0x00	**(u16)** 0x0001 *Length* = 1	**(u8)** 0x1A

Zeitpunkt t2 (t2 = t1 + x):
zweites *Request (0xA2)*-Telegramm vom Host an Protokolltreiber

Befehls-Header mit *OpCode* = 0xA2	**(u8)** 0x00 *Channel* = 0	**(u8)** 0x00 *Mode* = 0	**(u8)** 0x01 *Send* = 1	**(u8)** 0x01 *Concatenate* = 1	**(u8)** 0x00 *Segmentation* = 0	**(u8)** 0x00	**(u16)** 0x0001 *Length* = 1	**(u8)** 0x9B

Zeitpunkt t3 (t3 = t2 + y):
KWP2000-Telegramm wird vom Protokolltreiber gebildet und gesendet

KWP2000 Header	**(u8)** 0x1A	**(u8)** 0x9B	**(u8)** KW2000 Prüfsumme

Das Flag Segmentation ist im Beispiel nicht gesetzt, da es sich auf das Diagnose-Protokoll bezieht.

Grundlage des Datenaustausches über die K-Leitung und der Kommunikation zwischen Host und Protokolltreiber ist das „Frage-Antwort“- Prinzip. D.h., jede Anforderung hat eine (!) Antwort zur Folge.

Innerhalb der verschiedenen Protokolle gibt es teilweise Ausnahmen von diesem Prinzip. Diese Abweichungen werden über verschiedene Mechanismen durch den Protokolltreiber abgefangen. Eine Anfrage vom Host hat im Erfolgsfall immer eine Antwort vom Treiber zur Folge.

Falls beim Host innerhalb des eingestellten Zeitschemas keine Antwort vom Protokolltreiber eintrifft, kann (sollte) dessen Status über den Befehl 0xA5 KLine Diagnose – Zustand abfragen kontrolliert werden.

Der Sachverhalt soll kurz an einem spezifischen Beispiel veranschaulicht werden:
Ein SG nach KWP1281 antwortet auf den Tester- Request ***Steuergeräteidentifikation lesen*** (***BT*** = 0x00) mit einem segmentierten Identifikationsstring. D.h. die Antwort des Steuergerätes (der Identifikationsstring) ist auf mehrere Antwortblöcke verteilt. Jeder dieser Antwortblöcke muss laut KWP1281 beim Empfang vom Tester (Treiber) durch einen ***Acknowledge*** Block bestätigt werden. Der Treiber erkennt das Ende der Steuergeräteantwort, wenn er als direkte Reaktion auf einen solchen ***Acknowledge*** Block seinerseits einen ***Acknowledge*** Block vom SG erhält. Der Protokolltreiber bildet jetzt aus den empfangenen Segmenten die Antwort für den Host. In diesem Fall dient der ***Acknowledge*** Block lediglich zur Ablaufsteuerung des Protokolls. Er ist in der Antwort vom Treiber zum Host nicht enthalten!

Als Gegenbeispiel soll der KWP1281-Befehl ***Fehlerspeicher löschen*** (***BT*** = 0x05) dienen. Dessen erfolgreiche Abarbeitung meldet das SG direkt durch einen ***Acknowledge*** Block (keine weiteren Antworten!). Jetzt dient diese Antwort nicht der Ablaufsteuerung des Protokolls, sondern als Quittierung für die Ausführung eines Befehls. In diesem Falle wird der ***Acknowledge*** Block durch den Treiber als Antwort an den Host weitergereicht.

4.4.8 0xA3 KLine Diagnose – Responsepuffer abfragen

Mit diesem Befehl wird die Antwort (Response) auf einen vorhergehenden Diagnose Request (0xA2 KLine Diagnose – Request senden) abgeholt. **Voraussetzung für den Befehl ist die Deaktivierung des automatischen Sendens von Diagnoseantworten** (Bit 0 im Flags Parameter beim 0xA0 KLine Diagnose – Konfiguration Befehl ist auf 0 gesetzt). Dieser Befehl wird auch verwendet, um die Response auf 0xA1 KLine Diagnose – Sitzung starten sowie die Response auf 0xA3 KLine Diagnose – Response-Puffer abfragen (nicht bei KWP1281!) abzufragen.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1..3	reserved	Reserviert

Die vom Protokolltreiber zurück gelieferte Antwort enthält innerhalb des Nutzdatenfeldes nur den Nutzdatenteil der entsprechenden Response (mit ***ServiceID*** bzw. ***Blocktitel***, ohne Header, Prüffelder usw.). Die Antwort kann unter Umständen segmentiert sein (d.h., auf mehrere Befehlstelegramme verteilt).

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	Flags	Bit 0 = 0: Keine Segmentierung auf Diagnose-Ebene Bit 0 = 1: Segmentation (Segmentierung auf Diagnose-Ebene) Bit 1 = 0: Idle Bit 1 = 1: Busy (Ein Request wurde noch nicht beantwortet/ erfolgreich abgesetzt) Bit 2 = 0: Invalid (Dieser Puffereintrag ist ungültig) Bit 2 = 1: Valid (Dieser Puffereintrag ist gültig) Bit 3 = 0: BufferEmpty (Der Diagnose-Response-Puffer ist leer) Bit 3 = 1: BufferNotEmpty (Der Puffer ist noch nicht leer) Bits 4..7: Reserviert
3	State	Diagnose-Zustand 0: Nicht initialisiert 1: Keine Verbindung 2: Verbindung wird aufgebaut 3: Verbindung steht 4: Verbindung wird abgebaut
4, 5	Length	Anzahl der Response-Bytes (0..(PARAM_SIZE – 8))
6, 7	RemainingLength	Anzahl der verbleibenden Response-Bytes
8.. (7+Length)	Response	Response, bestehend aus SID (Service-Identifier) und Daten

Wenn eine Diagnose-Response nicht in eine einzige Host-Antwort passt, muss der Host mehrere Antworten abholen.

Die letzte dieser Antworten enthält im Parameter RemainingLength eine Null.

Außerdem sollte der Diagnose-Response-Puffer solange gelesen werden, wie das Segmentation-Bit, das Busy-Bit oder das BufferNotEmpty-Bit von Flags gesetzt sind.

Interpretation der Antwort:

Die Inhalte der Antworten, die mit diesem Befehl nach
0xA1 KLine Diagnose – Sitzung starten,
0xA2 KLine Diagnose – Request senden und
0xA4 KLine Diagnose – Sitzung stoppen
abgeholt werden können, haben je nach verwendetem Protokoll verschiedene Bedeutung.

Die Interpretation dieser Antworten ist NICHT Aufgabe des K-Leitungs Protokolltreibers!

4.4.9 0xA4 KLine Diagnose – Sitzung stoppen

Dieser Befehl stoppt eine laufende Diagnose-Sitzung für den durch Channel festgelegten Multisession-Kanal.
Dabei wird die Diagnose-Verbindung abgebaut.

Nach dem Befehl sind bis zum nächsten
0xA1 KLine Diagnose – Sitzung starten Befehl keine weiteren
0xA2 KLine Diagnose – Request senden Befehle möglich.

Die mittels 0xA0 KLine Diagnose – Konfiguration gesetzten Einstellungen werden durch diesen Befehl NICHT zurückgesetzt!

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	Mode	0: Physikalische Adressierung 1: Funktionale Adressierung **Außerdem**: Wird das höchstwertige Bit gesetzt (0x80), ist keine Antwort (Response) auf die Anforderung (Request) nötig
2, 3	Length	Länge des Requests (Zur Zeit wird nur Length = 0 unterstützt und fest je nach Diagnose-Typ der entsprechende Stop-Service geschickt)
4.. (3+Length)	Request	Request, bestehend aus SID (Service-Identifier) und Daten

Der Befehl 0xA4 KLine Diagnose – Sitzung stoppen zum Beenden der Diagnose (bzw. Beenden der Kommunikation) ist von der Hostseite gesehen auf den ersten Blick für alle K-Leitungs-Protokolle identisch.

Innerhalb des K-Leitungs-Treibers werden jedoch spezifische, zum jeweils aktiven Protokoll passende Aktionen ausgelöst.

Alle Protokolle reagieren unterschiedlich.
Generell gilt: Der K-Line Protokolltreiber liefert IMMER eine Antwort auf den 0xA4 KLine Diagnose – Sitzung stoppen Befehl (entweder automatisch oder über die Abfrage 0xA3 KLine Diagnose – Response-Puffer abfragen, je nach eingestelltem Antwortmodus)!
Die Bedeutung der Antwortdaten ist jedoch je nach Protokoll unterschiedlich. Die richtige Interpretation liegt in der Verantwortung des Empfängers (Host).

Die folgende Tabelle soll die Vorgänge innerhalb des Protokolltreibers in Reaktion auf einen 0xA4 KLine Diagnose – Sitzung stoppen Befehl verdeutlichen.
Zu beachten ist, dass es auf der K-Leitung keine echte Trennung zwischen Diagnose- und Transportprotokoll gibt.
Vereinfacht gesagt ist „Beenden der Kommunikation“ auf der K-Leitung gleichzusetzen mit „Beenden der Diagnose“.

KWP2000	
Beschreibung	Antwort vom KLine Treiber
Der KLine Treiber generiert und sendet einen ***Stop Communication*** Request (KWP2000 Frame, **SID** = 0x82). Das Steuergerät reagiert mit ***Stop Communication*** positive (oder negative) Response (KWP2000 Frame, ***SID*** = 0xC2 bzw. ***SID*** = 0x7F, 0x82, 0xXX). Nach einer ***Positive Response*** ist die Kommunikation beendet.	Datenteil der ***Stop Communication*** Response vom SG

KWP1281	
Beschreibung	Antwort vom KLine Treiber
Der KLine Treiber generiert und sendet einen ***DiagnoseEnde*** Block (KWP1281 Block, ***BT*** = 0x06). Das Steuergerät reagiert im Erfolgsfall vor dem Kommunikationsende mit einem ***Acknowledge*** Block (***BT*** = 0x09) oder beendet die Kommunikation sofort ohne Rückmeldung.	Datenteil (***BT***) des ***Acknowledge*** Blocks (immer - auch falls kein Block vom SG)

ISO-9141-Ford	
Beschreibung	Antwort vom KLine Treiber
Der Kline Treiber generiert und sendet einen ***Request Operational State Entry*** Block (***Mode*** = 0x20). Das Steuergerät reagiert darauf mit einem ***General Response*** Frame (***Mode*** = 0x7F, im Erfolgsfall ***Response Code*** = 0x00). Im Erfolgsfall wird die Kommunikation danach beendet.	Datenteil des ***General Response*** Blocks

4.4.10 0xA5 KLine Diagnose – Zustand abfragen

Mit diesem Befehl wird der Diagnose-Zustand für den durch Channel festgelegten Multisession-Kanal abgefragt. Zusätzlich kann der Firmware-interne LastErrorCode zurückgesetzt werden.

Der Wert von LastErrorCode in der Antwort entspricht dem Wert des Firmware-internen LastErrorCode vor dessen Rücksetzen.

Der Firmware-interne LastErrorCode wird I. Allg. ohne Aufruf von 0xA5 LIN Diagnose – Zustand abfragen nach Start einer Diagnose-Sitzung mit 0xA1 KLine Diagnose – Sitzung starten sowie nach Stop einer Diagnose-Sitzung mit 0xA4 KLine Diagnose – Sitzung stoppen und Length ≠ 0 automatisch zurückgesetzt.

Befehl:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	ResetLastError	0: LastErrorCode nicht zurücksetzen 1: LastErrorCode zurücksetzen
2, 3	reserved	Reserviert

Antwort:

Byte	Bezeichnung	Bedeutung
0	Channel	Multisession-Kanal (beginnend mit 0)
1	LastErrorCode	Fehlercode (0 = kein Fehler)
2	DiagType	Diagnose Typ: 0: Kein Protokoll 1: Diagnose KWP2000 2: Diagnose KWP1281 3: Diagnose ISO-9141-Ford
3	State	Diagnose-Zustand 0: Nicht initialisiert 1: Keine Verbindung 2: Verbindung wird aufgebaut 3: Verbindung steht 4: Verbindung wird abgebaut
4	Flags	Bit 0 = 0: Idle Bit 0 = 1: Busy (Ein Request wurde noch nicht beantwortet/ erfolgreich abgesetzt) Bit 1 = 0: Der Diagnose-Response-Puffer ist leer Bit 1 = 1: RxBufferNotEmpty (Der Puffer ist noch nicht leer) Bits 2..7: Reserviert
5..7	reserved	Reserviert

L

M

N

P

R

S

T

U

V

W