

TERM-Light COM QVis Laufzeitsys Handbuch

PC/104*i*
ModuNORM^â

Änderungsnachweis

Änderung:	Mutationsstand:	Datum / Visum:
Erstausgabe TERM-Light COM V2.60 vom 02.02.1998 / FK	900720A	12.04.1998 / BT

© Copyright 1996 durch:

ModuNORM GmbH
CH-8840 Einsiedeln
Switzerland

CoDeSys ist Warenzeichen der 3S GmbH
ModuNORM® ist eingetragenes Warenzeichen der ModuNORM GmbH
RTXDOS und IBIOS sind Warenzeichen der Technosoftware AG
QVis ist Warenzeichen der Kinz Elektronik
WINbloc ist Warenzeichen der Weidmüller GmbH & Co.

Inhaltsverzeichnis**Seite:**

1.	Onboard CAN-Schnittstellen.....	4
1.1	Funktions-Beschreibung	4
1.2	Programm-Module (can8191.obj, can8191.h)	6
2.	PC-Adapter Dual-CAN.....	10
2.1	Funktions-Beschreibung	10
2.2	Programm-Module (dualcan.obj, dualcan.h).....	12
3.	PC-Adapter Quad-COM	17
3.1	Funktions-Beschreibung	17
3.2	Programm-Module (quadcom.obj, quadcom.h)	19

1. Onboard CAN-Schnittstellen

Die hier angebotenen Funktionen ermöglichen ein Senden und Empfangen von Daten über die CAN-Schnittstelle CAN1 und CAN2 auf der Prozessorplatine.

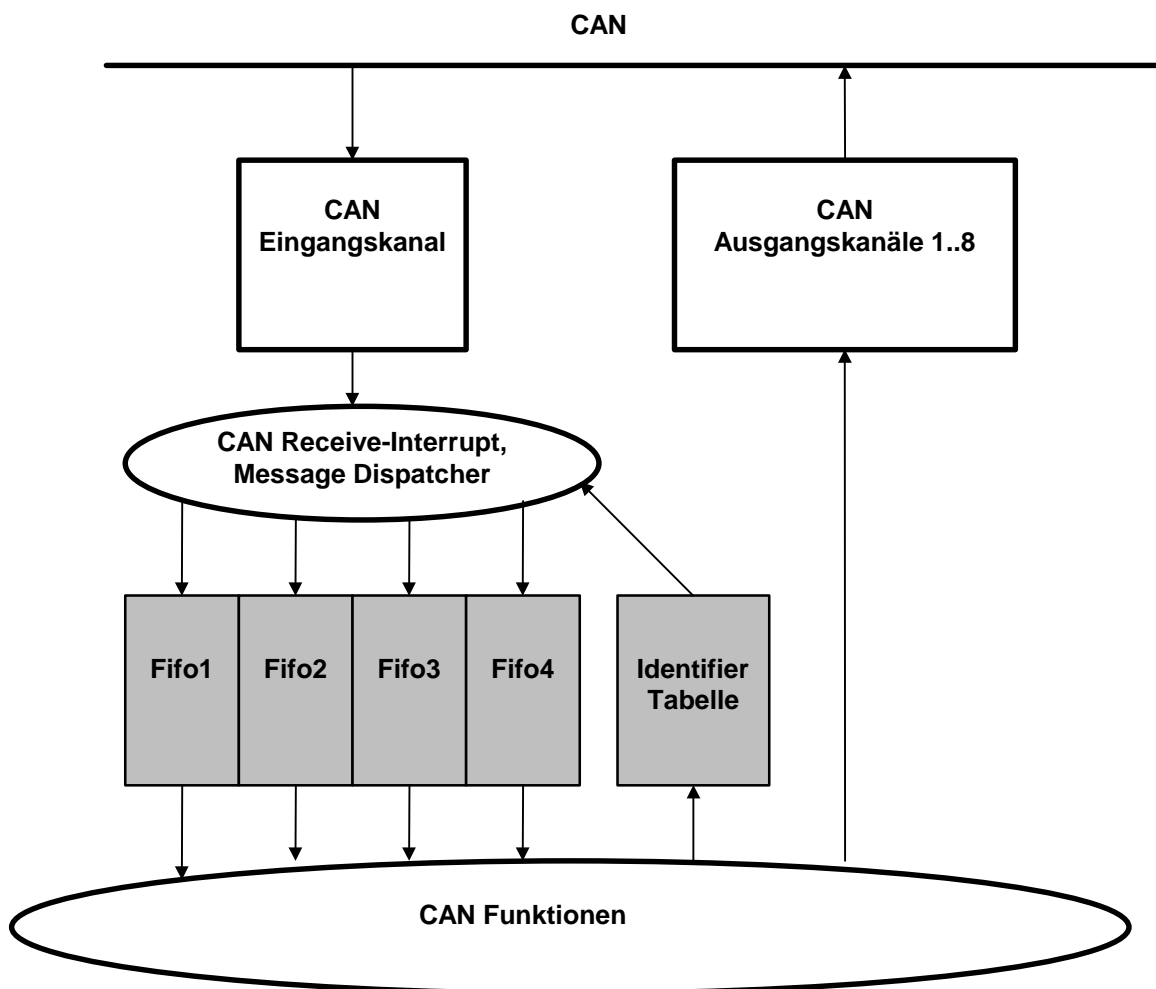
Das Format der CAN-Meldungen ist frei durch den Anwender bestimmbar.

1.1 Funktions-Beschreibung

1.1.1 Meldungshandling

Jede der zwei CAN-Schnittstellen besitzt hardwaremässig ein Eingangskanal und 8 Ausgangskanäle für eine komplette CAN-Meldung von max. 8 Byte. Pro Eingangskanal sind 4 Software Fifo-Buffer vorgesehen mit je einer Tiefe von 64 Meldungen. Für die Selektierung der Meldung ist eine Identifier-Tabelle vorgesehen, die frei konfigurierbar ist. Für jeden Identifier (0..2047) kann ein Fifo-Buffer (1..4) zugeordnet werden bzw. deaktiviert werden.

Struktur pro CAN-Schnittstelle:



1.1.2 Hardware-Anforderungen

- Modul-Typ: ModuNORM PC-386CAN/VGA, 104151A
- Rechner: i386EX, 25 MHz
- CAN-Controller: Siemens 81C91, 16 MHz Clock

1.1.3 CAN-Übertragungsrate

Die Übertragungsrate wird über eine Funktion eingestellt. Die Sample-Time und Jump-Time wird fix von der Übertragungsrate abgeleitet.

Übertragungsrate	Bit-Time	Sample-Time*	Jump-Time
20 kbit/s	50us	40,125us	3,125us
50 kbit/s	20us	16,25us	1,25us
100 kbit/s	10us	8,125us	0,625us
125 kbit/s	8us	6,5us	0,5us
250 kbit/s	4us	3,25us	0,25us
500 kbit/s	2us	1,625us	0,125us
800 kbit/s	1,25us	0,875us	0,125us
1000 kbit/s	1us	0,625us	0,125us

*** CAN-Controller-Fehler:**

Die Sample-Time wird wegen CAN-Controller-Fehler noch um die Jump-Time vergrößert. Nach Behebung des Fehlers gelten die Werte in der Tabelle.

1.1.4 Interrupthandling

Für jeden Kanal wird eine getrennte Interruptleitung verwendet und die Interruptvektoren sind softwaremässig einstellbar und somit sind die Treiber auch für andere Terminals verwendbar.

Der Interrupt ist innerhalb der Interruptroutine gesperrt!

Interruptvektoren:

- INT4/IRQ8:
- INT5/IRQ9: CAN2 fix bei PC-386CAN/VGA, 104151A
- INT6/IRQ13:
- INT7/IRQ14: CAN1 fix bei PC-386CAN/VGA, 104151A

Interrupt-Bearbeitungszeit: 70µs für einen Kanal.

1.1.5 Adressierung

Die CAN-Controller werden über den Memorybereich adressiert. Die Basis-Adressen sind softwaremässig einstellbar und somit sind die Treiber auch für andere Terminals verwendbar.

Adressen:

- 0xC8000..0xC80FF: CAN1 fix bei PC-386CAN/VGA, 104151A
- 0xC8100..0xC81FF: CAN2 fix bei PC-386CAN/VGA, 104151A

1.2 Programm-Module (can8191.obj, can8191.h)

Die hier angebotenen Funktionen wurden mit Borland C/C++ 3.0 erstellt und in der Programmiersprache Borland C++ geschrieben, wobei kein C++ Syntax verwendet wurde.

Initialisierungs-Funktionen:

- **CanInit** initialisiert den CAN-Controller und aktiviert den Empfangs-Interrupt.
- **CanSelFifo** aktiviert die zu empfangenen Identifier (0..2047) und ordnet sie dem entsprechenden Empfangs-FIFO (1..4) zu.

Beispiel:

```
// CAN1 initialisieren und Fifo 1 Id=234 aktivieren
CanInit(CAN1_SELECT, 0xC800, 5, IRQ14, CAN_125K); // CAN1 init.
CanSelFifo(CAN1_SELECT, 234, 1); // CAN1, Id = 234, Fifo = 1
```

Run-Funktionen:

- **CanStatus** liest den CAN-Controller Status. Der Status sollte laufend ausgewertet werden und bei CAN-Controller Fehler muß dieser wieder initialisiert werden.
- **CanDisable** deaktiviert den CAN-Controller. Diese Funktion sollte immer am Ende eines Programmes aufgerufen werden.
- **CanRxData** liest eine CAN-Meldung aus dem Empfangs-Fifo.
- **CanTxReady** testet ob der Sendebuffer frei ist.
- **CanSend** versendet eine CAN-Meldung.

Beispiel:

```
// Meldung versenden, CAN2/Kanall
if (CanTxReady(CAN2_SELECT, 1) == CAN_TRUE )
{
    // CAN2 versendet Daten
    CanSend(CAN2_SELECT, 1, &buffer);
}
// Meldung empfangen, CAN2/Fifo2
if ((CanRxData(CAN1_SELECT, 2, &buffer) & 0x81 ) == 0x80)
```

1.2.1 Datenstruktur CAN-Message

```
typedef struct // Variablen Definition
{
    unsigned int id;           // Identifier
    unsigned char len;        // Anzahl Datenbytes
    unsigned char dat0;       // Daten
    unsigned char dat1;       // Daten
    unsigned char dat2;       // Daten
    unsigned char dat3;       // Daten
    unsigned char dat4;       // Daten
    unsigned char dat5;       // Daten
    unsigned char dat6;       // Daten
    unsigned char dat7;       // Daten
    unsigned long time;       // Zeit
    unsigned char res1;       // Reserve1
} CAN_BUF;
```

```
// Die Elemente time und res1 duerfen nicht beschrieben werden, wohl aber
// ausgelesen
```

1.2.2 Konstanten, vom Anwender nicht veränderbar

```
// CAN-Kanal Auswahl
#define CAN1_SELECT      1
#define CAN2_SELECT      2

// Uebertragungsrate
#define CAN_20K          0
#define CAN_50K          1
#define CAN_100K         2
#define CAN_125K         3
#define CAN_250K         4
#define CAN_500K         5
#define CAN_800K         6
#define CAN_1M           7

// Status
#define CAN_FALSE        0
#define CAN_TRUE         1
```

1.2.3 Funktionen

```
//-----
// Funktion:      CanInit(chan, addr, cs, intNr, rate)
// Beschreibung : CAN Baustein initialisieren
// Parameter :    chan - CAN1/CAN2
//               addr - Port-Adresse
//               cs  - Chip-Select 0..5 (CS0..CS5)
//               intNr - Interrupt 4..7 (Int4..INT7)
//               INT4 = IRQ8
//               INT5 = IRQ9
//               INT6 = IRQ13
//               INT7 = IRQ14
//               rate - Uebertragungsrate: 20k..1Mbit/s
// Returnwert :   CAN_TRUE  - o.k.
//               CAN_FALSE - Fehler, kein CAN verfuegbar
//-----
```

```
unsigned char CanInit(unsigned char chan,
                      unsigned int addr,
                      unsigned char cs,
                      unsigned char intNr,
                      unsigned char rate);
```

```
//-----
// Funktion:      CanSelFifo(chan, id, fifo)
// Beschreibung : Identifier auf Empfangs-Fifo legen
// Parameter :    chan - CAN1_SELECT/CAN2_SELECT
//               id  - 0..2047
//               fifo - Fifo-Nummer 1..4, 0 = Identifier deaktiviert
// Returnwert :   CAN_TRUE  - o.k.
//               CAN_FALSE - ungueltige Parameter
//-----
```

```
unsigned char CanSelFifo(unsigned char chan,
                        unsigned int id,
                        unsigned char fifo);
```

```
//-----
// Funktion:      CanStatus(chan)
// Beschreibung :  CAN-Status ermitteln.
// Parameter :    chan - CAN1_SELECT/CAN2_SELECT
// Returnwert :   Status-Register
//   bit0: Init Mode
//   bit1: Reset Request
//   bit2: Bus Stat (1: Bus Off State, 0: Normal Mode)
//
// Wenn Meldung "Warning-Level" erscheint, ist der Baustein noch
// kommunikationsfaehig, jedoch kann der Errorcount nur durch
// weitere fehlerfreie Kommunikation verbessert werden.
//
//   bit3: Receiver Warning Level - 1: Receive Error Level >= 96
//                                       0: Receive Error Level < 96
//   bit4: Transmit Warning Level - 1: Transmit Error Level >= 96
//                                       0: Transmit Error Level < 96
//
//   bit5: Transmission Complete - 1: Letzte Uebertr. war erfolgreich
//                                       0: Letzte Uebertr. war nicht erfolgr.
//   bit6: nicht benutzt
//
//   bit7: Error Passiv Interrupt - 1: Error Passive Interrupt
//                                       (Achtung!! CAN-Kanal wurde abgeschaltet)
```

```
unsigned char CanStatus(unsigned char chan);
```

```
//-----
// Funktion:      CanDisable(chan, intNr)
// Beschreibung :  ReInitialisiert die CAN-Controller fuer Interruptbetrieb.
//               Diese Funktion muss unbedingt am Programmende aufgerufen
//               werden, um die Int.Vektoren wieder zurueckzusetzen.
// Parameter :    chan - CAN1_SELECT/CAN2_SELECT
//               intNr - Interrupt 4..7 (INT4, INT5, INT6, INT7)
//               INT4 = IRQ8
//               INT5 = IRQ9
//               INT6 = IRQ13
//               INT7 = IRQ14
// Returnwert :   CAN_TRUE - o.k.
//               CAN_FALSE - Fehler, ungueltiger Kanal
```

```
unsigned char CanDisable(unsigned char chan,
                        unsigned char intNr);
```

```
//-----
// Funktion:      CanRxData(chan, fifo, *ptr)
// Beschreibung :  CAN-Empfangsdaten auslesen.
// Parameter :    chan - CAN1_SELECT/CAN2_SELECT
//               fifo - Receive-Fifo Nummer (1..4)
//               *ptr - Zieladresse fuer die Daten
// Returnwert :   CAN Status-Register
//   bit0:   Empfangs-Fifo Overrun
//   bit1..6: nicht benutzt
//   bit7:   Daten empfangen
```

```
unsigned char CanRxData(unsigned char chan,
                        unsigned char fifo,
                        CAN_BUF *ptr);
```

```
//-----
// Funktion:      CanTxReady(chan, tbuf)
// Beschreibung :  Prueft, ob gesendet werden kann
// Parameter :    chan - CAN1_SELECT/CAN2_SELECT
//               tbuf - Transmit-Kanal-Nr.: 1..8
// Returnwert :   CAN_TRUE - CAN bereit
//               CAN_FALSE - CAN beschaeftigt
```

```
unsigned char CanTxReady(unsigned char chan,  
                          unsigned char tbuf);
```

```
//-----  
// Funktion:      CanSend(can, tbuf, *ptr)  
// Beschreibung : Sendet Message ueber Transmit-Kanal 1..8  
// Parameter :   chan - CAN1_SELECT/CAN2_SELECT  
//              tbuf - Transmit-Kanal-Nr.: 1..8,  
//              Kanal8 hat hoechste Prioritaet.  
// !!! Wegen CAN-Chip-Fehler muessen bei Verwendung von mehreren Kanaelen,  
//      auch die Kanaele mit niederer Prioritaet frei sein.  
//      Ist ein Kanal mit niedriger Prioritaet am senden, so koennte ein  
//      zweites senden dieser Nachricht ausgeloeset werden.  
//  
//      *ptr - Identifier, Laenge (0..8) und Datenbytes  
// Returnwert :  CAN_TRUE - o.k.  
//              CAN_FALSE - Fehler
```

```
unsigned char CanSend(unsigned char chan,  
                      unsigned char tbuf,  
                      CAN_BUF *ptr);
```

2. PC-Adapter Dual-CAN

Die hier angebotenen Funktionen ermöglichen ein Senden und Empfangen von Daten über die CAN-Schnittstelle CAN1 und CAN2 des Dual-CAN PC-Adapters.

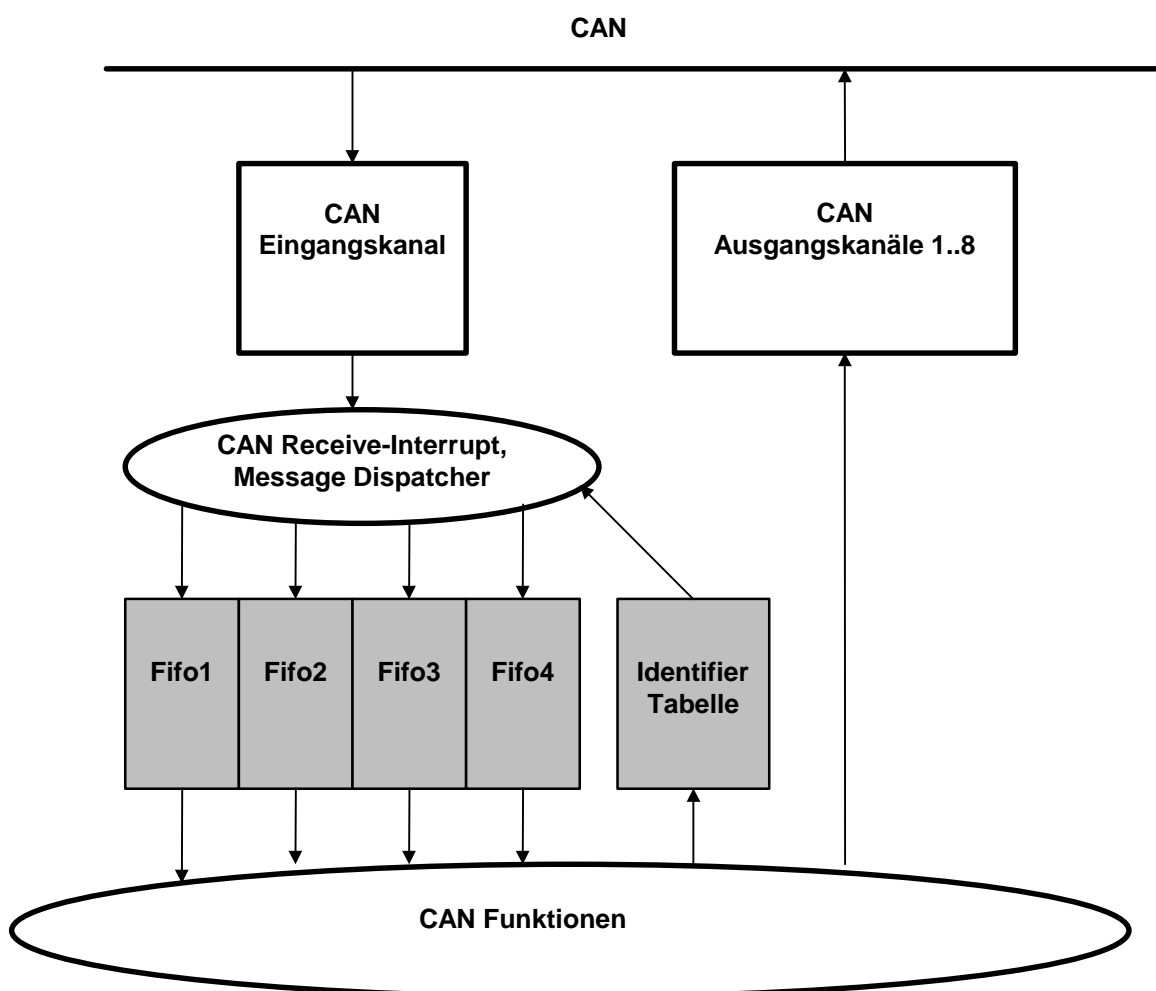
Das Format der CAN-Meldungen ist frei durch den Anwender bestimmbar.

2.1 Funktions-Beschreibung

2.1.1 Meldungshandling

Jede der zwei CAN-Schnittstellen besitzt hardwaremässig ein Eingangskanal und 8 Ausgangskanäle für eine komplette CAN-Meldung von max. 8 Byte. Pro Eingangskanal sind 4 Software Fifo-Buffer vorgesehen mit je einer Tiefe von 64 Meldungen. Für die Selektierung der Meldung ist eine Identifier-Tabelle vorgesehen, die frei konfigurierbar ist. Für jeden Identifier (0..2047) kann ein Fifo-Buffer (1..4) zugeordnet werden bzw. deaktiviert werden.

Struktur pro CAN-Schnittstelle:



2.1.2 Hardware-Anforderungen

- Modul-Typ: ModuNORM PC-Adapter Dual-CAN, 103031A
- Rechner: i386EX, 25 MHz
- CAN-Controller: Siemens 81C91, 16 MHz Clock

2.1.3 CAN-Übertragungsrate

Die Übertragungsrate wird über eine Funktion eingestellt. Die Sample-Time und Jump-Time wird fix von der Übertragungsrate abgeleitet.

Übertragungsrate	Bit-Time	Sample-Time*	Jump-Time
20 kbit/s	50us	40,125us	3,125us
50 kbit/s	20us	16,25us	1,25us
100 kbit/s	10us	8,125us	0,625us
125 kbit/s	8us	6,5us	0,5us
250 kbit/s	4us	3,25us	0,25us
500 kbit/s	2us	1,625us	0,125us
800 kbit/s	1,25us	0,875us	0,125us
1000 kbit/s	1us	0,625us	0,125us

* CAN-Controller-Fehler:

Die Sample-Time wird wegen CAN-Controller-Fehler noch um die Jump-Time vergrößert. Nach Behebung des Fehlers gelten die Werte in der Tabelle.

2.1.4 Interrupthandling

Es wird eine Interruptleitung für beide Kanäle verwendet. Innerhalb der Interruptroutine werden beide Kanäle hintereinander abgefragt. Der Interrupt wird nach sperren des DualCAN-Interrupts für andere Interrupts freigegeben und am Ende wieder gesperrt, bis der Interruptbearbeitung abgeschlossen ist. Die Interruptvektoren sind softwaremässig einstellbar und müssen mit der Hardwareeinstellung (Jumper) übereinstimmen.

Interruptvektoren:

- INT0/IRQ12 nur mit ModuNORM Modulen verwendbar
- INT1/IRQ5:
- INT2/IRQ6:
- INT3/IRQ7:
- INT4/IRQ8: nicht verwendbar mit PC-Adapter Dual-CAN
- INT5/IRQ9:
- INT6/IRQ13: nicht verwendbar mit PC-Adapter Dual-CAN
- INT7/IRQ14:

Interrupt-Sperrzeit: 15µs beim Aufruf und 15us am Ende.

Interrupt-Bearbeitungszeit: 105µs für einen Kanal, 165us wenn beide Kanäle Daten empfangen.

2.1.5 Adressierung

Die CAN-Controller werden über den Memory-Bereich adressiert. Die Basis-Adressen sind softwaremässig einstellbar und müssen mit der Hardwareeinstellung übereinstimmen.

2.2 Programm-Module (dualcan.obj, dualcan.h)

Die hier angebotenen Funktionen wurden mit Borland C/C++ 3.0 erstellt und in der Programmiersprache Borland C++ geschrieben, wobei kein C++ Syntax verwendet wurde.

Initialisierungs-Funktionen:

- **DualCanStartup** initialisiert die Interruptlogik und die CAN Basis-Adresse.
- **DualCanInit** initialisiert den CAN-Controller und aktiviert den Empfangs-Interrupt.
- **DualCanSelFifo** aktiviert die zu empfangenen Identifier (0..2047) und ordnet sie dem dem entsprechenden Empfangs-Fifo (1..4) zu.

Beispiel:

```
// CAN1 initialisieren und Id=234 in Fifo1
DualCanStartup(0xD800, 0xD810, 1); // CAN Basis-Adressen, INT1(IRQ5)
DualCanInit(CAN1_SELECT, CAN_125K); // CAN1 125kbit/s
DualCanSelFifo(CAN1_SELECT, 234, 1); // id = 234, Fifo = 1
```

Run-Funktionen:

- **DualCanStatus** liest den CAN-Controller Status. Der Status sollte laufend ausgewertet werden und bei CAN-Controller Fehler muß dieser wieder initialisiert werden.
- **DualCanReinit** deaktiviert ein CAN-Kanal.
- **DualCanDisable** deaktiviert die Interrupt-Logik. Diese Funktion sollte immer am Ende eines Programmes aufgerufen werden.
- **DualCanRxData** liest eine CAN-Meldung aus dem Empfangs-Fifo.
- **DualCanTxReady** testet ob der Sendebuffer frei ist.
- **DualCanSend** versendet eine CAN-Meldung.

Beispiel:

```
// Meldung versenden, CAN2/Kanal1
if (DualCanTxReady(CAN2_SELECT, 1) == CAN_TRUE )
{
    // CAN2 versendet Daten
    DualCanSend(CAN2_SELECT, 1, &buffer);
}

// Meldung empfangen, CAN2/Fifo2
if ((DualCanRxData(CAN1_SELECT, 2, &buffer) & 0x81 ) == 0x80)
```

2.2.1 Datenstruktur CAN-Message

```
typedef struct // Variablen Definition
{
    unsigned int id; // Identifier
    unsigned char len; // Anzahl Datenbytes
    unsigned char dat0; // Daten
    unsigned char dat1; // Daten
    unsigned char dat2; // Daten
    unsigned char dat3; // Daten
    unsigned char dat4; // Daten
    unsigned char dat5; // Daten
    unsigned char dat6; // Daten
    unsigned char dat7; // Daten
    unsigned long time; // Zeit
    unsigned char res1; // Reserve1
} CAN_BUF;
// Die Elemente time und res1 duerfen nicht beschrieben werden, wohl aber
// ausgelesen
```

2.2.2 Konstanten, vom Anwender nicht veränderbar

```
// CAN-Kanal Auswahl
#define CAN1_SELECT      1
#define CAN2_SELECT      2

// Uebertragungsrate
#define CAN_20K          0
#define CAN_50K          1
#define CAN_100K         2
#define CAN_125K         3
#define CAN_250K         4
#define CAN_500K         5
#define CAN_800K         6
#define CAN_1M           7

// Status
#define CAN_FALSE        0
#define CAN_TRUE         1
```

2.2.3 Funktionen

```
//-----
// Funktion:      DualCanStartup(addr, intNr)
// Beschreibung : Interrupt/Port-Adresse initialisieren
//               Diese Funktion muss als erste Funktion aufgefufen werden.
// Parameter :    addr1 - Adresse CAN1 (xxxx)
//               addr2 - Adresse CAN2 (xxxx)
//               intNr - Interrupt INT0..INT7 (0..7)
//               INT0 = IRQ12 (nur Mikrap-Module)
//               INT1 = IRQ5
//               INT2 = IRQ6
//               INT3 = IRQ7
//               INT4 = IRQ8 (nicht verwendbar)
//               INT5 = IRQ9
//               INT6 = IRQ13 (nicht verwendbar)
//               INT7 = IRQ14
// Returnwert :   CAN_TRUE - o.k.
//               CAN_FALSE - Fehler, nicht ausgefuehrt
```

```
unsigned char DualCanStartup(unsigned int  addr1,
                             unsigned int  addr2,
                             unsigned char intNr);
```

```
//-----
// Funktion:      DualCanInit(chan, rate)
// Beschreibung : CAN Baustein initialisieren
// Parameter :    chan - CAN1/CAN2
//               rate - Uebertragungsrate: 20k..1Mbit/s
// Returnwert :   CAN_TRUE - o.k.
//               CAN_FALSE - Fehler, kein CAN verfuegbar
```

```
unsigned char DualCanInit(unsigned char chan,
                          unsigned char rate);
```

```
//-----
// Funktion:      DualCanReinit(chan)
// Beschreibung : CAN Baustein deaktivieren, Interrupt disable
// Parameter :    chan - CAN1/CAN2
// Returnwert :   CAN_TRUE - o.k.
//               CAN_FALSE - Fehler
```

```
unsigned char DualCanReinit(unsigned char chan);
```

```
//-----
// Funktion:      DualCanSelFifo(chan, id, fifo)
// Beschreibung : Identifier auf Empfangs-Fifo legen
// Parameter :   chan - CAN1_SELECT/CAN2_SELECT
//              id  - 0..2047
//              fifo - Fifo-Nummer 1..4, 0 = Identifier deaktiviert
// Returnwert :  CAN_TRUE  - o.k.
//              CAN_FALSE - ungueltige Parameter
```

```
unsigned char DualCanSelFifo(unsigned char chan,
                             unsigned int id,
                             unsigned char fifo);
```

```
//-----
// Funktion:      DualCanStatus(chan)
// Beschreibung : CAN-Status ermitteln.
// Parameter :   chan - CAN1_SELECT/CAN2_SELECT
// Returnwert :  Status-Register
//              bit0: Init Mode
//              bit1: Reset Request
//              bit2: Bus Stat (1: Bus Off State, 0: Normal Mode)
//
//              Wenn Meldung "Warning-Level" erscheint, ist der Baustein noch
//              kommunikationsfaehig, jedoch kann der Errorcount nur durch
//              weitere fehlerfreie Kommunikation verbessert werden.
//
//              bit3: Receiver Warning Level - 1: Receive Error Level >= 96
//              bit4: Transmit Warning Level - 1: Transmit Error Level >= 96
//              bit5: Transmission Complete - 1: Letzte Uebertr. war erfolgreich
//              bit6: nicht benutzt
//              bit7: Error Passiv Interrupt - 1: Error Passive Interrupt
//              (Achtung!! CAN-Kanal wurde abgeschaltet)
```

```
unsigned char DualCanStatus(unsigned char chan);
```

```
//-----
// Funktion:      DualCanDisable()
// Beschreibung : ReInitialisiert den CAN-Controller fuer Interruptbetrieb.
//              Diese Funktion muss unbedingt am Programmende aufgerufen
//              werden, um die Int.Vektoren wieder zurueckzusetzen.
// Parameter :   keine
// Returnwert :  CAN_TRUE  - o.k.
//              CAN_FALSE - Fehler, ungueltiger Kanal
```

```
unsigned char DualCanDisable(void);
```

```
//-----
// Funktion:      DualCanRxData(chan, fifo, *ptr)
// Beschreibung : CAN-Empfangsdaten auslesen.
// Parameter :   chan - CAN1_SELECT/CAN2_SELECT
//              fifo - Receive-Fifo Nummer (1..4)
//              *ptr - Zieladresse für die Daten
// Returnwert :  CAN Status-Register
//              bit0: Empfangs-Fifo Overrun
//              bit1..6: nicht benutzt
//              bit7: Daten empfangen
```

```
unsigned char DualCanRxData(unsigned char chan,
                             unsigned char fifo,
                             CAN_BUF *ptr);
```



```
//-----  
// Funktion:      DualCanTxReady(chan, tbuf)  
// Beschreibung : Prueft, ob gesendet werden kann  
// Parameter :   chan - CAN1_SELECT/CAN2_SELECT  
//              tbuf - Transmit-Kanal-Nr.: 1..8  
// Returnwert  : CAN_TRUE  - CAN bereit  
//              CAN_FALSE - CAN beschaeftigt
```

```
unsigned char DualCanTxReady(unsigned char chan,  
                             unsigned char tbuf);
```

```
//-----  
// Funktion:      DualCanSend(can, tbuf, *ptr)  
// Beschreibung : Sendet Message ueber Transmit-Kanal 1..8  
// Parameter :   chan - CAN1_SELECT/CAN2_SELECT  
//              tbuf - Transmit-Kanal-Nr.: 1..8,  
//                  Kanal8 hat hoechste Prioritaet.  
// !!! Wegen CAN-Chip-Fehler muessen bei Verwendung von mehreren Kanaelen,  
//      auch die Kanaele mit niederer Prioritaet frei sein.  
//      Ist ein Kanal mit niedriger Prioritaet am senden, so koennte ein  
//      zweites senden dieser Nachricht ausgeloeset werden.  
//  
//              *ptr - Identifier, Laenge (0..8) und Datenbytes  
// Returnwert  : CAN_TRUE  - o.k.  
//              CAN_FALSE - Fehler
```

```
unsigned char DualCanSend(unsigned char chan,  
                           unsigned char tbuf,  
                           CAN_BUF *ptr);
```

3. PC-Adapter Quad-COM

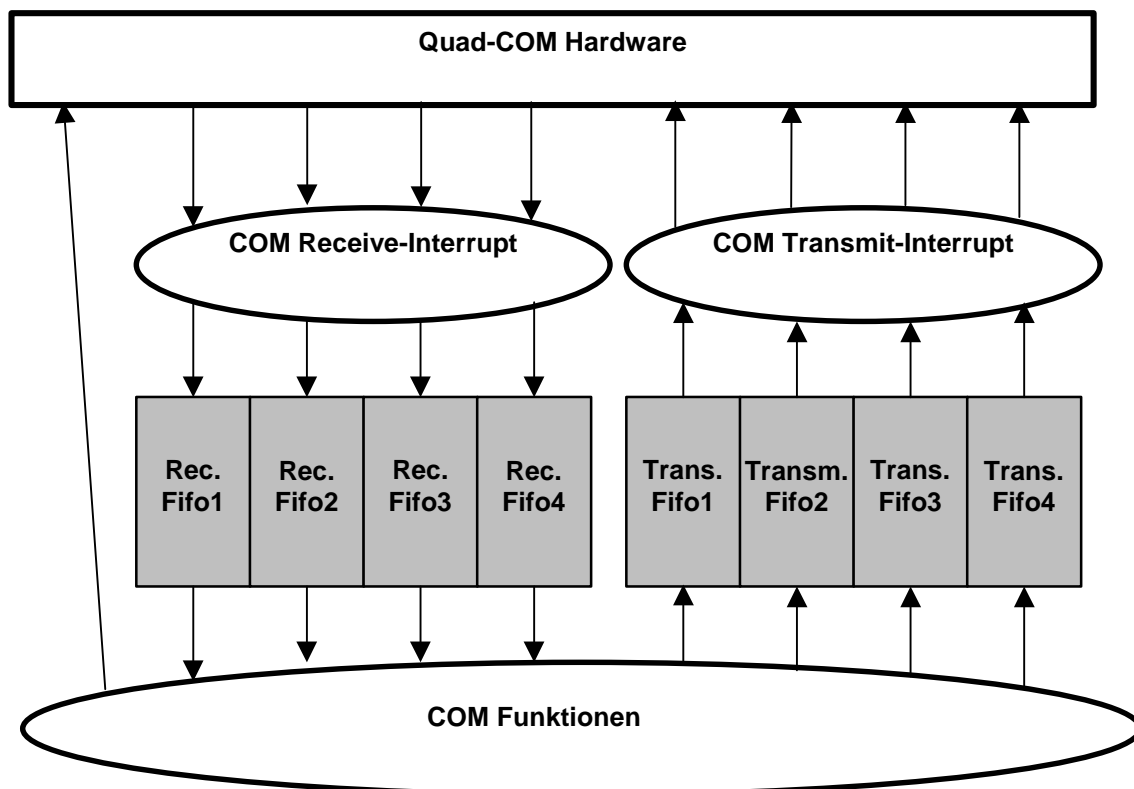
Die hier angebotenen Funktionen ermöglichen ein Senden und Empfangen von Daten über die 4 COM-Schnittstellen des QuadCOM-Moduls.

3.1 Funktions-Beschreibung

3.1.1 Meldungshandling

Jeder der 4 COM-Schnittstellen besitzt ein Software Fifo-Buffer für den Empfangskanal und für den Sendekanal. Die Tiefe aller Fifo-Buffer beträgt **1024** Zeichen.

Struktur der Quad-COM Schnittstellen:



3.1.2 Hardware-Anforderungen

- Modul-Typ: ModuNORM PC-Adapter Quad-COM
- Rechner: i386EX, 25 MHz
- COM-Controller: TL16C554

3.1.3 COM-Übertragungsrate

Die Übertragungsrate wird über eine Funktion eingestellt.

Übertragungsrate: 300..38400Baud

3.1.4 Interrupthandling

Es wird eine Interruptleitung für die vier Kanäle verwendet. Innerhalb der Interruptroutine werden die vier Kanäle hintereinander abgefragt. Der Interrupt wird nach sperren des QuadCOM-Interrupts für andere Interrupts freigegeben und am Ende wieder gesperrt, bis der Interruptbearbeitung abgeschlossen ist. Die Interruptvektoren sind softwaremässig einstellbar und müssen mit der Hardwareeinstellung (Jumper) bereinstimmen.

Interruptvektoren:

- INT0/IRQ12 nur mit ModuNORM Modulen verwendbar
- INT1/IRQ5:
- INT2/IRQ6:
- INT3/IRQ7:
- INT4/IRQ8: nicht verwendbar mit PC-Adapter Quad-COM
- INT5/IRQ9:
- INT6/IRQ13: nicht verwendbar mit PC-Adapter Quad-COM
- INT7/IRQ14:

Interrupt-Sperrzeit: 15µs beim Aufruf und 15us am Ende.

Interrupt-Bearbeitungszeit: 75µs für einen Kanal RX oder TX,
90us für einen Kanal RX und TX gleichzeitig,
180us für 4 Kanäle RX und TX gleichzeitig.

3.1.5 Adressierung

Die CAN-Controller werden über den I/O-Bereich adressiert. Die Basis-Adressen sind softwaremässig einstellbar und müssen mit der Hardwareeinstellung übereinstimmen.

3.2 Programm-Module (quadcom.obj, quadcom.h)

Die hier angebotenen Funktionen wurden mit Borland C/C++ 3.0 erstellt und in der Programmiersprache Borland C++ geschrieben, wobei kein C++ Syntax verwendet wurde.

Initialisierungs-Funktionen:

- **QuadcomStartup** initialisiert die Interruptlogik und die COM Basis-Adressen.
- **QuadcomInit** initialisiert den COM-Controller und aktiviert den Empfangs-Interrupt.

Beispiel:

```
// COM1..4 initialisieren
QuadcomStartup(0x280, 0x288, 0x290, 0x298, 1);
QuadcomInit(1, 0, 38400, 8, 1, 0);
QuadcomInit(2, 0, 38400, 8, 1, 0);
QuadcomInit(3, 0, 38400, 8, 1, 0);
QuadcomInit(4, 0, 38400, 8, 1, 0);
```

Run-Funktionen:

- **QuadcomModemStatus** liest den COM-Modem Status.
- **QuadcomReinit** deaktiviert ein COM-Kanal.
- **QuadcomDisable** deaktiviert die Interrupt-Logik. Diese Funktion sollte immer am Ende eines Programmes aufgerufen werden.
- **QuadcomGetChar** liest ein Charakter aus dem Empfangs-Fifo.
- **QuadcomTxStatus** testet das Sende-Fifo.
- **QuadcomPutChar** schreibt ein Charakter in das Sende-Fifo.
- **QuadcomPutBlock** schreibt ein Datenblock in das Sende-Fifo.
- **QuadcomSend** versendet alle Charakter im Sende-Fifo.

Beispiel:

```
// Zeichen-Ausgabe
stat1 = QuadcomTxStatus(1); // Sende-Fifo 1 Status lesen
if (stat1 = 1) // Sende-Fifo 1 leer ?
{
    QuadcomPutChar(1, tmp1); // 1 Byte ins Sende-Fifo 1
    QuadcomPutBlock(1, dat, 10); // 10 Bytes ins Sende-Fifo 1
    QuadcomSend(1); // Ausgabe der Daten im Sende-Fifo 1
}
```

3.2.1 Konstanten, vom Anwender nicht veränderbar

```
// Status
#define QCOM_FALSE 0
#define QCOM_TRUE 1
```

3.2.2 Funktionen

```
//-----
// Funktion:      QuadcomStartup(addr1, addr2, addr3, addr4, intNr)
// Beschreibung : Interrupt/Port-Adresse initialisieren
//               Diese Funktion muss als erste Funktion aufgefufen werden.
// Parameter :   addr1 - Port-Adresse COM1 (0..xxx)
//               addr2 - Port-Adresse COM2 (0..xxx)
//               addr3 - Port-Adresse COM3 (0..xxx)
//               addr4 - Port-Adresse COM4 (0..xxx)
//               intNr - Interrupt INT0..INT7 (0..7)
//               INT0 = IRQ12 (nur Mikrap-Module)
//               INT1 = IRQ5
//               INT2 = IRQ6
//               INT3 = IRQ7
//               INT4 = IRQ8 (nicht verwendbar)
//               INT5 = IRQ9
//               INT6 = IRQ13 (nicht verwendbar)
//               INT7 = IRQ14
// Returnwert :  COM_TRUE  - o.k.
//               COM_FALSE - Fehler, nicht ausgeführt
```

```
unsigned char QuadcomStartup(unsigned int  addr1,
                             unsigned int  addr2,
                             unsigned int  addr3,
                             unsigned int  addr4,
                             unsigned char  intNr);
```

```
//-----
// Funktion:      QuadcomInit(chan,signal,baudRate,charLen,stopBit,parity)
// Beschreibung : COM Kanal initialisieren und freigeben
// Parameter :   chan      - COM1..COM4 (1..4)
//               signal    - 0 = RS232, 1 = RS485
//               baudRate  - 300..38400 Baud
//               charLen   - 5..8
//               stopBit   - 0..2
//               parity    - 0 = kein, 1 = gerade, 2 = ungerade
// Returnwert :  COM_TRUE  - o.k.
//               COM_FALSE - Fehler
```

```
unsigned char QuadcomInit(unsigned char  chan,
                          unsigned char  signal,
                          unsigned int   baudRate,
                          unsigned char  charLen,
                          unsigned char  stopBit,
                          unsigned char  parity);
```

```
//-----
// Funktion:      QuadcomReinit(chan)
// Beschreibung : COM Kanal ausschalten (Interrupt Disable)
// Parameter :   chan      - COM1..COM4 (1..4)
// Returnwert :  QCOM_TRUE - o.k.
//               QCOM_FALSE - Fehler
```

```
unsigned char QuadcomReinit(unsigned char  chan);
```

```
//-----
// Funktion:      QuadcomDisable()
// Beschreibung : ReInitialisiert den COM-Controller fuer Interruptbetrieb.
//               Diese Funktion muss unbedingt am Programmende aufgerufen
//               werden, um die Int.Vektoren wieder zurueckzusetzen.
// Parameter :   kein
// Returnwert :  COM_TRUE  - o.k.
//               COM_FALSE - Fehler
```

```
unsigned char QuadcomDisable(void);
```



```
//-----
// Funktion:      QuadcomModemStatus(chan)
// Beschreibung : Modem-Status lesen
// Parameter :    chan - COM1..COM4 (1..4)
// Returnwert :   bit0 - CTS Signal
//               bit4 - CTS pos. Flanke
```

```
unsigned char QuadcomModemStatus(unsigned char chan);
```

```
//-----
// Funktion:      QuadcomGetChar(chan)
// Beschreibung : Charakter aus Eingabe-Fifo auslesen
// Parameter :    chan - COM1..COM4 (1..4)
// Returnwert :   bit0..7: Zeichen
//               bit8:   Software Empfangspuffer-Overrun
//               bit9:   Hardware Receive-Overrun
//               bit10:  Hardware Parity-Error
//               bit11:  Hardware Framing-Error
//               bit12:  Hardware Break Interrupt
//               bit13..14: nicht benutzt
//               bit15:  Zeichen empfangen
```

```
unsigned int QuadcomGetChar(unsigned char chan);
```

```
//-----
// Funktion:      QuadcomTxStatus(chan)
// Beschreibung : Prueft Ausgabe-Fifo
// Parameter :    chan - COM1..COM4 (1..4)
// Returnwert :   0 = Kanal ungueltig
//               1 = Fifo leer
//               2 = Fifo nicht leer und nicht voll
//               3 = Fifo voll
```

```
unsigned char QuadcomTxStatus(unsigned char chan);
```

```
//-----
// Funktion:      QuadcomPutChar(chan, chr)
// Beschreibung : Charakter in Ausgabe-Fifo ablegen
// Parameter :    chan - COM1..COM4 (1..4)
//               chr - Charakter
// Returnwert :   QCOM_TRUE - o.k.
//               QCOM_FALSE - Fehler
```

```
unsigned char QuadcomPutChar(unsigned char chan, unsigned char chr);
```

```
//-----
// Funktion:      QuadcomPutBlock(chan, *ptr, len)
// Beschreibung : Charakter-Block in Ausgabe-Fifo ablegen
// Parameter :    chan - COM1..COM4 (1..4)
//               ptr - Pointer auf Charakter-Block
//               len - Laenge
// Returnwert :   QCOM_TRUE - o.k.
//               QCOM_FALSE - Fehler
```

```
unsigned char QuadcomPutBlock(unsigned char chan,
                               unsigned char *ptr,
                               unsigned int len);
```

```
//-----
// Funktion:      QuadcomStartSend(chan)
// Beschreibung : Ausgabe starten
// Parameter :    chan - COM1..COM4 (1..4)
// Returnwert :   QCOM_TRUE - o.k.
//               QCOM_FALSE - Fehler
```

unsigned char QuadcomStartSend(*unsigned char* chan);