IP-CAN

Technische Unterlagen

Erstellt von: D. Dorsch
Datum: 30.09.1998
Überarbeitet: 15.12.1998

Dokumentationsnr.: 942.1950.01

Die Informationen in diesem Handbuch sind so umfassend, präzise und aktuell wie möglich. DMS behält sich das Recht vor, dieses Handbuch jederzeit ohne Vorankündigung zu ändern.

Wir übernehmen keinerlei Haftung für in diesem Handbuch eventuell enthaltene technische Fehler oder Druckfehler und möglicherweise entstehende direkte oder indirekte Folgeschäden.

Warenzeichen:
Motorola ist ein eingetragenes Warenzeichen von Motorola, Incorporated
OS-9 und Microware sind eingetragene Warenzeichen von Microware Systems Corp.

Vorwort

Um die Leistungsfähigkeit dieser Baugruppe ausnutzen zu können, benötigen Sie als Anwender ausführliche Informationen. Diese technische Unterlage wendet sich an Entwickler, Projektierer und Programmierer, die diese Baugruppe einsetzen wollen.

In vorliegenden technischen Unterlagen haben wir versucht, diese Informationen möglichst vollständig und gegliedert zusammenzustellen. Auf den folgenden Seiten des Vorwortes finden Sie Informationen, die Ihnen den Umgang mit diesen technischen Unterlagen erleichtern sollen. Wir werden Ihnen erläutern, wie wir die Inhalte der technischen Unterlagen gegliedert haben.

Trotz aller Bemühungen können in diesen technischen Unterlagen nicht alle Probleme erläutert werden, die bei den vielfältigen Einsatzmöglichkeiten der Baugruppe auftreten können. Wenden Sie sich in diesen Fällen bitte an Ihren DMS-Ansprechpartner, den Sie jederzeit um Rat fragen können.

Inhaltsbeschreibung

- Beschreibung der Hardware (Kap. 1)
  In diesem Kapitel ist im wesentlichen die Baugruppe selbst beschrieben; wie sie sich in die Familie der DMS VMEbus-Baugruppen einfügt, und wie sie prinzipiell funktioniert.
- Informationen zur Inbetriebnahme (Kap. 2)
  In diesem Kapitel haben wie die Inhalte zusammengefaßt, die Sie für die Inbetriebnahme benötigen. Hier wird deutlich, wie sich Hardware und Software gegenseitig beeinflussen.
- Funktionen der Baugruppe (Kap. 3)
  Dieses Kapitel enthält die komplette Beschreibung einer bestimmten Funktion, d.h. von der Verdrahtung bis zur Programmierung ist die Beschreibung vollständig enthalten.
- Adressen der Baugruppe (Kap. 4)
  In diesem Kapitel werden alle Adreßbereiche und die Registeradressen zusammengefaßt.
- OS9-Software (Kap. 5)
  Dieses Kapitel enthält die komplette Beschreibung über die baugruppenabhängige Software wie Treiber und Descriptoren.
- Wartung und Instandhaltung (Kap. 6)
  In diesem Kapitel sind Angaben zur Wartung und Instandsetzung, sowie Hinweise zu Fehlern, die beim Einsatz der Baugruppe auftreten können.
- Anhang (Kap. 7)
  In diesem Kapitel finden Sie Schaltbilder, Maßbilder usw.

Am Ende der technischen Unterlage sind Korrekturblätter eingeheftet. Tragen Sie dort bitte Ihre "Verbesserungs-, Ergänzungs- und Korrekturvorschläge" ein und senden Sie das Blatt an uns zurück. Sie helfen uns dadurch, die nächste Auflage zu verbessern.

Vereinbarungen

Um die Übersichtlichkeit der technischen Unterlagen zu verbessern, wurde die Gliederung in Menü-Form durchgeführt, das bedeutet:

- Am Anfang der technischen Unterlage finden Sie ein vollständiges Gesamtinhaltsverzeichnis.
- Die einzelnen Kapitel sind bis zur dritten Stufe gegliedert. Zur weiteren Unterteilung werden Überschriften fett gedruckt.
- Seiten, Bilder und Tabellen sind durchgehend numeriert.
- Für bestimmte Begriffe verwenden wir Abkürzungen. Ein Abkürzungsverzeichnis finden Sie in Anhang.
- Fußnoten werden mit kleinen hochgestellten Ziffern (z.B. "[1]"), oder hochgestellten Sternchen "*" gekennzeichnet. Die zugehörigen Erläuterungen finden Sie im allgemeinen am unteren Blattrand. Aufzählungen sind mit einem schwarzen Punkt (?) gekennzeichnet (wie beispielsweise in dieser Aufstellung) oder mit Spiegelstrichen (-).
- Querverweise werden folgendermaßen dargestellt: "(siehe Kap. 3.3.2)" verweist auf den Abschnitt 3.3.2.
- Die Größenangaben in Zeichnungen und Maßbildern werden in "mm" ausgedrückt.
- Wertebereiche werden folgendermaßen dargestellt: 17 .. 21 = 17 bis 21
- Hexadezimale Zahlenangaben sind durch ein "$" gekennzeichnet.
- Besonders wichtige Informationen finden Sie in den gekennzeichneten, schwarz umrandeten "Schaukästen":

Warnung

Die Definition der Begriffe "Warnung", "Gefahr", "Vorsicht", und "Hinweis" entnehmen Sie bitte den "Sicherheitstechnischen Hinweisen für den Benutzer" am Ende dieser Einführung.

Gültigkeit der Dokumentation

Diese Dokumentation gilt für:

Baugruppe: IP-CAN Version ab 1.0

Sicherheitstechnische Hinweise für den Benutzer

Diese Dokumentation enthält die erforderlichen Informationen für den bestimmungsgemäßen Gebrauch der darin beschriebenen Produkte. Sie wendet sich an qualifiziertes Personal. Qualifiziertes Personal im Sinne der sicherheitsbezogenen Hinweise in dieser Dokumentation oder auf dem Produkt selbst sind Personen, die

- entweder als Entwicklungs-
- oder als Projektierungspersonal mit den Sicherheitskonzepten der Automatisierungstechnik vertraut sind.

Gefahrenhinweise

Die folgenden Hinweise dienen einerseits Ihrer persönlichen Sicherheit und andererseits der Sicherheit vor Beschädigung des beschriebenen Produkts oder angeschlossener Geräte.

Sicherheitshinweise und Warnungen zur Abwendung von Gefahren für Leben und Gesundheit von Benutzern oder Instandhaltungspersonal bzw. zur Vermeidung von Sachschäden werden in dieser Dokumentation durch die hier definierten Signalbegriffe hervorgehoben. Die verwendeten Begriffe haben im Sinne der Dokumentation und der Hinweise auf den Produkten selbst folgende Bedeutung:

Gefahr
bedeutet, daß Tod, schwere Körperverletzung oder erheblicher Sachschaden eintreten werden, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Warnung
bedeutet, daß Tod, schwere Körperverletzung oder erheblicher Sachschaden eintreten können, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Vorsicht
bedeutet, daß eine leichte Körperverletzung oder ein Sachschaden eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Hinweis
ist eine wichtige Information über das Produkt, die Handhabung des Produktes oder den jeweiligen Teil der Dokumentation, auf den besonders aufmerksam gemacht werden soll.

Bestimmungsgemäßer Gebrauch

Warnung
- Die Baugruppe darf nur für die im Katalog und in der technischen Beschreibung vorgesehenen Einsatzfälle und nur in Verbindung mit von DMS empfohlenen bzw. zugelassenen Fremdgeräten und -komponenten verwendet werden.
- Der einwandfreie und sichere Betrieb des Produktes setzt sachgemäßen Transport, sachgerechte Lagerung, Aufstellung und Montage sowie sorgfältige Bedienung und Instandhaltung voraus.

Inhaltsverzeichnis

1. Die IP-CAN im Überblick

- **Low Cost Sensor Aktor Bus**
- **4 unabhängige CAN Kanäle mit SJA-1000 Controller**
- **High- und Low-Speed CAN-Bustreiber softwaremäßig umschaltbar**
- **Physical Layer mit Mercedes-spezifischer Beschaltung**
- **BasicCAN und CAN 2.0B mit 11 oder 29 Bit Identifier**
- **CAN Protokoll ISO-Schicht 2 in Hardware realisiert**
- **bis 1 MBit/sec**
- **Einfache Softwareeinbindung**
- **Direkter Zugriff auf die CAN Chips**
- **Auf passiven oder intelligenten IP-Grundkarten einsetzbar**

1.1. Allgemeines

Der CAN-Bus ist ein einfaches, störsicheres und kostengünstiges Netzwerkkonzept, das für die Kommunikation von Steuergeräten mit ihren Sensoren und Aktoren in Fahrzeugen, entwickelt wurde. In der Automobiltechnik hat sich CAN bereits durchgesetzt und ist als ISO-Standard genormt.

1.2. CAN Eigenschaften

CAN ist multimasterfähig, das heißt alle Stationen können Master werden, es ist keine Leitstation nötig, auch bei Ausfall einiger Teilnehmer bleiben die anderen kommunikationsfähig (offene Busstruktur). CAN benötigt keine Startadressen, sondern ist nachrichtenorientiert, das bedeutet, eine Nachricht, wie zum Beispiel die Position eines Antriebes, kann gleichzeitig von allen interessierten Teilnehmern empfangen werden. Es sind 11-Bit oder 29-Bit Nachrichten-Identifier möglich. CAN führt eine verlustfreie Arbitrierung auf Bit-Ebene aus. Der Identifier entscheidet über die Priorität und von wichtigen Nachrichten wird kein Bit gestört, so daß sie auch im Kollissionsfall nicht wiederholt werden braucht.
Die Nutzlänge ist max. 8 Byte. Sie ist für die Übertragung von Sensordaten optimal und führt zu kurzen Reaktionszeiten für Prioritätsmeldungen. CAN leistet per Hardware umfangreiche Fehlerabdeckung und veranlaßt das Wiederholen gestörter Telegramme.

1.3. CAN-VMEbus -Interface

Das IP-CAN Modul ist ein I/O-Modul nach dem IP-Modul-Standard und ist mit 4 Philips SJA1000 CAN-Controllern aufgebaut. Die CAN-Chips sind über die IP-Grundkarte direkt vom VMEbus ansprechbar. Hiermit können alle CAN-Funktionen mit voller Geschwindigkeit von der VMEBus-CPU bearbeitet werden. Dies erleichtert die Softwareentwicklung und erlaubt den Einsatz des Moduls auch als Bus-Monitor für den CAN-Bus.
Das IP-CAN wurde auf der Green Spring VIPC-610 Grundkarte getestet. Sie bietet 4 IP-Modul Steckplätze und damit 16 CAN-Bus-Schnittstellen.

1.4. CAN Bustreiber

Das IP-CAN hat für jeden Kanal einen Highspeed und einen Lowspeed Treiber. Die Auswahl erfolgt über ein Kontrollregister per Software. Das IP-CAN ist damit sowohl für Anwendungen im KFZ-Antriebsbereich (Highspeed) als auch im KFZ-Innnenraumbereich (Lowspeed) einsetzbar. Der gesamte Physical Layer entspricht der Daimler- Benz-Spezifikation.

1.5. Lokale Intelligenz

IP-Module, wie das IP-CAN, können auf verschiedenen VMEbus-CPU-Baugruppen benutzt werden. Bei Motorola sind dies z.B. die MVME162 mit 68040 CPU oder die MVME172 mit 68060 CPU. Diese Baugruppen haben ein Dual-Port-Memory zum VMEbus für die Datenübergabe. Die CAN-Software kann lokal auf der Baugruppe mit den IP-CAN Modulen arbeiten. So kann aus einer MVME162 CPU und 4 IP-CAN Modulen ein 16 Kanal CAN-Inteface mit lokaler Vorverarbeitung oder ein CAN Gateway aufgebaut werden.

1.6. Softwareunterstützung

Die CAN-Controller der IP-CAN lassen sich VMEbus-seitig sehr einfach ansprechen. In der technischen Unterlage sind entsprechende Programmbeispiele als ANSI C Source Code enthalten. Das Programm ist für die Green Spring VIPC-610 Grundkarte und das OS-9 Betriebssystem geschrieben und kann leicht auf eine andere Umgebung portiert werden.
Bei Bedarf ist es auf einer DOS-Diskette lieferbar.

1.7. Technische Daten

CAN-Prozessor	pro Kanal ein SJA1000
Protokoll	BasicCAN und CAN 2.0B
Identifier	11 oder 29 Bit
Übertragungsrate	bis zu1 MBit/sec
Nutzdatenlänge	0...8 Byte
CAN-Schnittstellen	4 CAN-Schnittstellen mit je einem High- und Low-Speed Treiber
Potentialtrennung	keine
Interner-Clock	16 MHz für CAN-Controller
IP-Modul-Clock	8 MHz oder 32 MHz
Abmessungen	99,1 x 45,7 x 7,3 mm
Stromaufnahme	5V / max. 300mA
Betriebstemperatur	0 bis +50 °C
Lagertemperatur:	-25 bis +70 °C

1.8. Lieferumfang

Zum Lieferumfang der IP-CAN gehört:

- Baugruppe IP-CAN
- Technische Unterlagen IP-CAN

1.9. Bestellbezeichnungen

IP-Modul	
IP-CAN	CAN Interface Modul mit 4 Kanälen je Kanal umschaltbar High-Speed-Treiber und Low-Speed-Treiber
IP Grundkarte	
IP-VME4	Green Spring VIPC-610 Grundkarte VME 6 HE mit 4 Modulplätzen

1.10. Sicherheitshinweise

Vorsicht

- Die Baugruppe ist als Steckmodul für IP-Grundkarten vorgesehen.
- Die Baugruppe ist für den Betrieb in einem zwangsbelüfteten Gehäuse vorgesehen. Für eine ausreichende Lüftung ist zu sorgen.
- Die Baugruppe darf nur im eingebauten Zustand eingeschaltet werden.
- Vor dem Anschließen oder Trennen von Schnittstellen ist die Stromversorgung (VCC) der Baugruppe auszuschalten.
- Bei der Handhabung der Baugruppe sind die einschlägigen ESD-Schutzmaßnahmen zu befolgen.
- Diese Baugruppe erzeugt und verwendet Hochfrequenzsignale und kann sie ausstrahlen. Der Betrieb der Baugruppe kann durch starke Hochfrequenzsignale gestört werden. Beim Entwurf der Baugruppe wurde von DMS auf ein EMV-gerechtes Design geachtet z.B. galvanisch getrennte Schnittstellen mit EMV-Schutzbeschaltung, Multilayertechnik usw. Beim Einsatz der Baugruppe hat der Anwender auf Einhaltung der gültigen EMV-Bestimmungen zu achten.

2. Inbetriebnahme der IP-CAN

Warnung
Das System muß bei allen Hardwareveränderungen im oder am System abgeschaltet werden!

2.1. System Konfigurationen

Die IP-CAN ist für den Einsatz auf einer IP-Grundkarte (Green Spring VIPC-610) vorgesehen. Für jedes Modul ist neben dem Ident-Bereich ein 1 KB Memory-Bereich erforderlich. der I/O-Bereich wird nicht genutzt.

2.1.1. Einstellung der VIPC-610

Das Beispiel ergibt folgende Einstellung :

VMEbusadresse: Standard (A24) 600000 bis 67FFFF
und Short (A16). 6000 bis 63FF

Folgende Jumper müssen gesteckt werden:

Basisadresse: 0x600000
E1
E7-7
E7-4
E7-3
E7-2
E7-1

IP-Size 128 KB
E2-3 ✍ E2-4
E2-5 ✍ E2-6

Mem Size 512 KB
E5-1 ✍ E8-1
E5-2 ✍ E8-2
E5-3 ✍ E8-3
E5-4 ✍ E8-4
E5-5 ✍ E8-5
E5-6 ✍ E8-6

2.2. Hardware-Einstellungen

Auf der IP-CAN sind keine Einstellungen erforderlich.

2.3. Endmontage

Das IP-CAN ist in den vorgesehenen Steckplatz zu stecken und auf der IP-Grundkarte festzuschrauben. Danach ist die Grundkarte ins System einzusetzen und mit den Halsschrauben an der Frontplatte am System festzuschrauben. Nur bei einer korrekt eingesetzten und festgeschraubten Karte ist eine einwandfreie Funktion der Karte gewährleistet.

2.4. Software-Einstellungen

Es sind keine Softwareeinstellungen vorgesehen.

2.5. Demontage

Zur Demontage des IP-CAN ist die Grundträgerkarte auszubauen, die Befestigungsschrauben auf der Lötseite der Grundträgerkarte zu entfernen und das IP-CAN aus der Steckfassung herauszuziehen.

2.6. Verpackung und Transport

Verpacken Sie die Baugruppe sorgfältig (ESD-geschützt), am besten in der Originalverpackung.
Falls Sie die Baugruppe zur Reparatur an DMS einschicken, legen Sie bitte das ausgefüllte Formblatt "DMS-Service Anlage" (siehe Kap. 6.4) bei.

3. Funktionsgruppen der IP-CAN

3.1. Allgemeines zum IP-Modul

Die IP-Module sind 91 * 48 mm große Steckbaugruppen. Über zwei Steckverbinder werden Bus- und I/O-Signale mit der Grundkarte verbunden. Durch die Anordnung der Schraubverbindung und der Steckverbinder entsteht eine stabile mechanische Einheit aus Grundkarte und Modul.

3.1.1. IP-Modul-ID

Über den Ident-Bereich kann das Modul von der Software identifiziert werden.

3.1.2. Clock Signale

Das IP-CAN hat einen eigenen 16 MHz Clockoszillator zur Erzeugung der CAN-Bit-Rate im SAJ-1000 Controller. Es kann deshalb ohne Softwareänderung sowohl auf 8 MHz- als auch auf 32 MHz-Grundkarten betrieben werden.

3.1.3. IP-Versorgungsspannungen

Das SM-CAN-Mxx wird mit +5V und +12V* versorgt. Die -12V Versorgungsspannung ist nicht erforderlich.

?? die +12V wird als Pull-Up Spannung für den Low-Speed Treiber benötigt.

3.1.4. CAN-Treiber

Die IP-CAN Module haben für jeden Kanal zwei unterschiedliche CAN-Treiber:

A: Highspeed-Treiber (82C250)

B: Lowspeed Treiber (TJA1053)

Die Auswahl geschieht durch ein Kontroll-Register.

3.1.5. I/O-Anschluß

Jedes IP-Modul hat 50 I/O Anschlüsse. Normalerweise sind diese am Modul, auf der Grundkarte und am Bandkabel gleich bezeichnet.

Signal	**Kanal 1**	**Kanal 2**	**Kanal 3**	**Kanal 4**
Can_L	50	44	38	32
Can_H	49	43	37	31
BAT	48	42	36	30
Wake_Up	47	41	35	29
GND	46 , 45	40, 39	34, 33	28, 27

Tabelle 1: Pinbelegung der IO-Signale

3.2. Adressenübersicht der IP-CAN

Der Zugriff erfolgt im Memory-Bereich byteweise über die ungeraden Adressen.

Modul Basisadressen der VIPC-610 bei der beschriebenen Einstellung VMEbus Standard 0x600000.

Register	Modul A	Modul B	Modul C	Modul D
Kanal 1 SAJ-1000	60 00 01	61 00 01	62 00 01	63 00 01
Kanal 1 Phy-Reg.	60 00 FF	61 00 FF	62 00 FF	63 00 FF
Kanal 2 SAJ-1000	60 01 01	61 01 01	62 01 01	63 01 01
Kanal 2 Phy-Reg.	60 01 FF	61 01 FF	62 01 FF	63 01 FF
Kanal 3 SAJ-1000	60 02 01	61 02 01	62 02 01	63 02 01
Kanal 3 Phy-Reg.	60 02 FF	61 02 FF	62 02 FF	63 02 FF
Kanal 4 SAJ-1000	60 03 01	61 03 01	62 03 01	63 03 01
Kanal 5 Phy-Reg.	60 03 FF	61 03 FF	62 03 FF	63 03 FF
IRQ-Vector	60 00 FD	61 00 FD	62 00 FD	63 00 FD

3.3. Informationen für Systemprogrammierer

Die Programmierung des SJA 1000 Controllers ist dem Datenblatt zu entnehmen. Es sind nur Bytezugriffe erlaubt. Alle Register liegen auf den ungeraden Adressen. Also Contol auf 0x600001 und Command auf 0x600003 u.s.w..

Nach dem Reset ist der SJA 1000 im BasicCAN Modus. Hier sind 32 Register vorhanden.

Durch Schreiben von Bit 7 des CDR Registers auf 1 kann in den PeliCAN Mode mit 128 Register umgeschaltet werden. Beide Modes haben eine unterschiedliche Adreßbelegung !

3.4. Register für Lowspeed Physical Layer (PhyReg)

Für jeden CAN Kanal ist ein Steuerregister für den Physical Layer vorhanden. Mit ihm wird von Highspeed auf Lowspeed Mode umgeschaltet.

Es hat folgende Bitbelegung:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Enable	STBn	INHIB	IRQL	IRQLEN	IRQC	0	LowSel
r/w	r/w	r	r	r/w	r	r	r/w

Die Bit's Enable, STBn, INHIB und IRQL entsprechen den Pin's des Lowspeed-Treibers TJA-1053. IRQL wird gespeichert solange IRQLEN gesetzt ist.
IRQLEN ist das Interrupt Freigabesignal für den IRQ des Lowspeed-Treibers.
IRQC zeigt an ob ein IRQ vom Controller SJA1000 ansteht.
Mit dem Bit LowSel wird auf den Lowspeed-Treiber umgeschaltet.

Also für High-Speed : PhyReg = 0x00 ;
und für Low-Speed: PhyReg = 0xC1 ;

4. Software

4.1. Programmbeispiel

Das Beispiel-Programm benutzt ein IP-CAN auf dem Modul Platz A einer VIPC-610 Grundkarte. Alle 4 Kanäle werden an einem gemeinsamen CAN Bus angeschlossen. Nach der Initialisierung sendet jeweils ein Kanal. Das Telegramm muß von den anderen 3 Kanälen empfangen werden. Der Sender wechselt zyklisch.
Im Normalfall wird der CAN-Bus mit 500 Kbit/sec und den Highspeed-Treibern benutzt.
Mit der Option –l wird der Lowspeed-Teiber mit 83,3 Kbit/sec eingestellt. (Aufruf : ipcan –l).

Das Beispiel benutzt den BasicCAN Mode. Die Struktur CAN200_typ gilt nur für diesen Mode.

5. Wartung und Instandhaltung

Es ist keine Wartung der IP-CAN erforderlich.

Im Falle eines Fehlers sind die Einstellungen der Baugruppe (Schnittstelleneinstellung) zu überprüfen. Läßt sich der Fehler nicht beseitigen, dann ist die Baugruppe sorgfältig zu verpacken (ESD-geschützt), mit einer Fehlerbeschreibung zu versehen und zur Reparatur an DMS einzuschicken. Im Anhang befindet sich dazu das Formular "DMS-Service Anlage".

6. Pläne und Anhang

6.1. Liste der verwendeten Abkürzungen

ASCII	American Standard Code for Information Interchange 7 Bit Zeichencode
CPU	Central Processing Unit
IP	Modul Standard VITA 4-1995
DMS	DMS Dorsch Mikrosystem GmbH
DP	PROFIBUS, Dezentrale Peripherie
DP-RAM	Dualported RAM
ID	Identifier
KB	Größenangabe für den Speicherplatz, 1 KB = 1.024 Bytes
MB	Größenangabe für den Speicherplatz, 1 MB = 1.048.576 Bytes
OS-9	Multiuser- Multitasking- Betriebssystem von Microware
VMEbus	Versa Module Eurocard Bussystem

6.2. Literaturhinweise

[1] DMS Kataolg, Dorsch Mikrosystem GmbH
[2] Datenblatt SJA1000 Stand alone CAN Controller, Philips

6.3. Korrekturblatt

DMS Dorsch Mikrosystem GmbH
Abt.: Qualitätsmanagement
Holmlück 13-15

24972 Steinbergkirche

Absender (bitte ausfüllen)

Name/Abt.: _____

Firma: _____

Anschrift: _____

Telefon: _____

Sie sind beim Lesen dieser technischen Unterlage auf Druckfehler gestoßen? Bitte teilen Sie uns die Fehler durch diesen Vordruck mit.

Auch für Ihre Anregungen und Verbesserungsvorschläge sind wir dankbar.

Dokumentationsnr.: _____ Datum: _____

6.4. DMS-Service Anlage

Sollten Sie eine DMS-Baugruppe einschicken,bitten wir Sie, den nachfolgenden Bericht ausgefüllt beizulegen.

Hinweis: Die Rücklieferung an DMS geschieht auf Kosten und Risiko des Kunden, auch bei Mängelrügen und Gewährleistungspflichten. Die Kosten für die Rücksendung an den Kunden bei Garantieansprüchen übernimmt DMS.

Name der Baugruppe	Serien-Nr	Datum

Ansprechpartner für technische Rückfragen

Firma : _____

Abteilung : _____ Herr / Frau: _____

Straße : _____ Tel.-Nr. : _____/_____

Ort : __ _____ _____

Gewünschte Bearbeitung

[ ] Reparatur
[ ] Überprüfung der Baugruppe und bei Fehler Reparatur

Die Baugruppe soll auf die neueste Version

Hardware:
[ ] hochgerüstet werden (Normalfall)
[ ] nicht hochgerüstet werden .

Software:
[ ] Update
[ ] kein Update (Normalfall)

Auftreten des Fehlers

[ ] von Anfang an
[ ] nach anfänglich einwandfreiem Betrieb
[ ] _____

Häufigkeit des Fehlers

[ ] sporadisch und selten
[ ] etwa ____ Minuten nach dem Einschalten
[ ] im Abstand von etwa ____ Minuten
[ ] bei einer Temperatur von _____ Grad Celsius
[ ] sehr häufig
[ ] ständig
[ ] _____

Vergleich mit anderen Baugruppen des gleichen Typs

[ ] das System läuft einwandfrei mit einer Baugruppe gleichen Typs
[ ] Baugruppe läuft in einem anderem System einwandfrei
[ ] kein Vergleich möglich

Einsatzbedingungen

[ ] DMS-System : _____
[ ] DMS-CPU : _____

[ ] Fremdsystem : _____
[ ] Fremd-CPU : _____

Einstellungen , Konfiguration , verwendete Software

_____

_____

_____

Problembeschreibung

_____

_____

_____

6.5. Listing "CAN-Demo" für IP-CAN

```
/*   Can Demo fuer IP-CAN
     ---------------------

v1.0  19.09.98  neu erstellt       do
*/

#include <dmstyp.h>
#include <module.h>
#include <types.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <csub.h>


typedef struct { byte dumy1  ; byte Cont ;
                 byte dumy2  ; byte Comm ;
                 byte dumy3  ; byte Stat ;
                 byte dumy4  ; byte Irq  ;
                 byte dumy5  ; byte Accept ;
                 byte dumy6  ; byte Mask ;
                 byte dumy7  ; byte Tim0 ;
                 byte dumy8  ; byte Tim1 ;
                 byte dumy9  ; byte OutC ;
                 byte dumy10 ; byte Test ;
                 byte dumy11 ; byte TxDes1 ;
                 byte dumy12 ; byte TxDes2 ;
                 byte dumy13 ; byte TxD1 ;
                 byte dumy14 ; byte TxD2 ;
                 byte dumy15 ; byte TxD3 ;
                 byte dumy16 ; byte TxD4 ;
                 byte dumy17 ; byte TxD5 ;
                 byte dumy18 ; byte TxD6 ;
                 byte dumy19 ; byte TxD7 ;
                 byte dumy20 ; byte TxD8 ;
                 byte dumy21 ; byte RxDes1 ;
                 byte dumy22 ; byte RxDes2 ;
                 byte dumy23 ; byte RxD1 ;
                 byte dumy24 ; byte RxD2 ;
                 byte dumy25 ; byte RxD3 ;
                 byte dumy26 ; byte RxD4 ;
                 byte dumy27 ; byte RxD5 ;
                 byte dumy28 ; byte RxD6 ;
                 byte dumy29 ; byte RxD7 ;
                 byte dumy30 ; byte RxD8 ;
                 byte dumy31 ; byte Clk ;

               } Can200_Typ ;
```

```
/* ------- Prototypes ---------------------- */


/* ------- Glabale Variablen ------------- */
volatile byte dumy ;
volatile byte  *p ;
volatile byte *IDR ;        /* IP-Modul Ident Register */
volatile Can200_Typ *CC1 ;  /* C200 Can Controler */
volatile Can200_Typ *CC2 ;  /* C200 Can Controler */
volatile Can200_Typ *CC3 ;  /* C200 Can Controler */
volatile Can200_Typ *CC4 ;  /* C200 Can Controler */
char LowSpeed ;

/* ------- Unterporgramme ---------------- */

DisplayID (volatile byte * IDR)
{
 int  n ;
 byte b ;
 byte crc ;

 printf ("\n Pointer ..... : %08X",IDR) ;
 printf ("\n Modul Laenge ..... : %3d",IDR[10*2+1]) ;

 if (IDR[10*2+1]== 12)
  {
     printf ("\n Modul Code ..... : ") ;
           for (n=0;n<4;n++) printf("%c",IDR[n*2+1]) ;
           printf ("\n Manufacture ID ..... :        %3d",IDR[4*2+1]) ;
           printf ("\n Model No ..... : %3d",IDR[5*2+1]) ;
           printf ("\n Revision .....  :  %02X",IDR[6*2+1]) ;
  }
 else printf ("\n *** Fehler ID-Laenge ") ;
 printf ("\n") ;
}



void DisplayStatus (byte Stat)
{
   if (Stat & 0x82)
    {
           if (Stat & 0x80) printf ("Bus-Off ") ;
                   else printf ("Bus-On  ") ;

           if (Stat & 0x40) printf ("Error ") ;
                   else printf ("O.K.  ") ;

           if (Stat & 0x20) printf ("Transmit ") ;
                   else printf ("Tx-idle  ") ;

           if (Stat & 0x10) printf ("Receive ") ;
```

```
                else printf ("Rx-idle ") ;

            if (Stat & 0x08) printf ("Tx-Comp ") ;
                else printf ("Tx-inco ") ;

            if (Stat & 0x04) printf ("Tx-relesasd ") ;
                else printf ("tx-locked   ") ;

            if (Stat & 0x02) printf ("Rx-overrun ") ;
                else printf ("Rx-absent  ") ;

            if (Stat & 0x01) printf ("Rx-full  \n") ;
                else printf ("Rx-empty \n") ;
        }
}



/* ------- Init Can Modul fuer 500 K-Bit ------------ */

void CanInit (volatile Can200_Typ * CC)
{
        int n ;
        volatile byte * PyReg ;
        n = (int) CC ;
   PyReg = (volatile byte *) ( n + 255) ;

        if (LowSpeed)
    *PyReg = 0xC1 ; /* Enable Low-Speed-Treiber */
        else
    *PyReg = 0x00 ; /* Enable High-Speed-Treiber */

        printf (" Init Kanal %d\n", ((n & 0x300) >> 8)+1 ) ;

        WaitMS (10) ;
        CC->Cont   = 0x01 ; /* Reset Request */
        CC->Accept = 0xFF ; /* Beispiel */
        CC->Mask   = 0xFF ; /* Accept all */
        if (LowSpeed)
          {
           CC->Tim0   = 11 ;   /* 16 MHz / (11+1) = 1,33 MHz */
           CC->Tim1   = 0x58 ; /* 16 Clk's = 1 Bit -> 1,33/16 = 83,3 KBit/sec */
          }
        else
          {
           CC->Tim0   = 1 ;    /* 1 -> 16 MHz / 2 = 8 MHz */
           CC->Tim1   = 0x58 ; /* 16 Clk's = 1 Bit -> 8/16 = 500 KBit/sec */
          } ;
        CC->OutC   = 0xCB ; /* Float-Pull, Normal, TX1 = clk */
        CC->Cont   = 0x00 ; /* Enable fast - no IRQ */
        WaitMS (10) ;
   DisplayStatus (CC->Stat) ;
```

```
}

/* ------- Test Empfang ----------------------------- */

void CanRx (volatile Can200_Typ * CC)
{
  byte RxDaten[8] ;
  short Anzahl ;
  wort  Id ;
        int n ;

   if (CC->Stat & 0x01)
             {
                    Id = CC->RxDes1 << 3;
               Id = Id | (CC->RxDes2 >> 5) ;
                        Anzahl = CC->RxDes2 & 0x0F ;
                        RxDaten[0] = CC->RxD1 ;
                        RxDaten[1] = CC->RxD2 ;
                        RxDaten[2] = CC->RxD3 ;
                        RxDaten[3] = CC->RxD4 ;
                        RxDaten[4] = CC->RxD5 ;
                        RxDaten[5] = CC->RxD6 ;
                        RxDaten[6] = CC->RxD7 ;
                        RxDaten[7] = CC->RxD8 ;
                        CC->Comm = 0x04 ; /* release receive buffer */

          n = (int) CC ;
          printf (" Kanal %d - ", ((n & 0x300) >> 8)+1 ) ;
           printf (" Empfang ID: %04X mit %d Byte:",Id,Anzahl) ;
                        for (n=0;n<Anzahl;n++) printf (" %02X",RxDaten[n]) ;
                        printf ("\n") ;
           DisplayStatus (CC->Stat) ;
                    };
 }


/* ------- CAN Senden ------------------------------ */

byte CanTx (volatile Can200_Typ * CC,
          int Id, short Anzahl, byte * Daten)
{
  int n ;
  volatile byte * p ;

     n = (int) CC ;
     printf ("\n Kanal %d - ", ((n & 0x300) >> 8)+1 ) ;
     if (!(CC->Stat & 0x04)) return (4) ;
           printf (" Send ID: %d mit %d Byte's\n",Id,Anzahl);
           CC->TxDes1 = Id >> 3 ;
           CC->TxDes2 = ((Id << 5) & 0xE0) | Anzahl ;
           p = &CC->TxD1 ;
           for (n=0;n<Anzahl;n++)
             {
```

```
                *p++ = *Daten++ ;
                    p++ ;
                };
    CC->Comm = 0x01 ;
    DisplayStatus (CC->Stat) ;
    return 0 ;
}



/* ------ Hauptprogram ------------------------------- */

int main (int argc, char **argv)
{
int error ;
int Modul ;
int tim ;
int count ;

   LowSpeed = 0 ;
        if (argc == 2)
          if (!strcmp(argv[1],"-l")) LowSpeed = 1 ;

   printf ("\n") ;
   printf (" CAN Demo fuer IP-CAN v1.0 \n") ;
        printf (" ------------------------ \n") ;
        if (LowSpeed) printf(" --- LOW SPEED ----");

        Modul = 0 ; /* 0 = Modul A */

        p   = (byte*) 0x1600000 ;   /* VMEbus Standart Adressbereich */
        permit ( (void*)p, 0x80000);/* Freigabe fuer 512 KB */

        CC1 = (Can200_Typ*) (p + (Modul * 0x20000) + 0x000) ; /* Modul 128 K je Modul */
        CC2 = (Can200_Typ*) (p + (Modul * 0x20000) + 0x100) ; /* Modul 128 K je Modul */
        CC3 = (Can200_Typ*) (p + (Modul * 0x20000) + 0x200) ; /* Modul 128 K je Modul */
        CC4 = (Can200_Typ*) (p + (Modul * 0x20000) + 0x300) ; /* Modul 128 K je Modul */

        p   = (byte*) 0x1F06000 ;   /* VMEbus Short I/0 Bereich */
        permit ( (void*)p, 0x400 ); /* Freigabe fuer 1KB */

        IDR = p + (Modul * 0x100) + 0x80 ; /* 0x80 = Offset zun ID-Bereich */


        DisplayID ( IDR ) ;

        CanInit (CC1) ;
        CanInit (CC2) ;
        CanInit (CC3) ;
        CanInit (CC4) ;

        tim = 0 ;
```

```
        count = 0 ;

   do
    {
          WaitMS ( 20 ) ;
          CanRx (CC1) ;
          CanRx (CC2) ;
          CanRx (CC3) ;
          CanRx (CC4) ;
          tim++ ;

     if (tim == 100)
           {
                 count++ ;
       if (CanTx (CC1, 1000, 4, (byte*)&count ))
         printf (" *** Send Error \n") ;
                };

     if (tim == 200)
           {
                 count++ ;
       if (CanTx (CC2, 1000, 4, (byte*)&count ))
         printf (" *** Send Error \n") ;
                };

     if (tim == 300)
           {
                 count++ ;
       if (CanTx (CC3, 1000, 4, (byte*)&count ))
         printf (" *** Send Error \n") ;
                };

     if (tim == 400)
           {
                 tim = 0 ;
                 count++ ;
       if (CanTx (CC4, 1000, 4, (byte*)&count ))
         printf (" *** Send Error \n") ;
                };


        } while ( !KEYDOWN() ) ;


return 0 ;
} /* END MAIN  */
```