PMC-CAN

Technische Unterlagen

Erstellt von: D. Dorsch
Datum: 2.04.2001
Überarbeitet: 21.05.2002

Dokumentationsnr.: 942.0144.04

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 den 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 wir 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 Unterlagen 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 nummeriert.
- Für bestimmte Begriffe verwenden wir Abkürzungen. Ein Abkürzungsverzeichnis finden Sie im 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:	PMC-CAN Version ab 1.0
Firmware	PicoBug (dms) v1.3

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 PMC-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**
- **Lokale Vorverarbeitung durch 32-Bit RISC-CPU**
- **Einfache Softwareanpassung durch Dualport-RAM Schnittstelle**
- **Lokale Bedienung über RS-232 Terminal Schnittstelle für einfachen Testbetrieb**
- **Einfacher CAN-Bus Monitor in Standart Firmware integriert**

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 Kollisionsfall 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-PCI -Interface

Das PMC-CAN Modul ist ein I/O-Modul nach dem PMC-Modul-Standard und ist mit 4 Philips SJA1000 CAN-Controllern aufgebaut. Die CAN-Chips werden von einer lokalen RISC-CPU bedient. Auf dieser CPU läuft die Firmware alle zeitkritischen Vorgänge übernimmt. Für den Benutzer stellt sie eine Datenschnittstelle im Dualport-RAM zur Verfügung. Dieses Dualport RAM wird über ein PCI-Businterface in den Adressraum der Host-CPU gemapt. Der Hauptrechner sieht also nur einen 128 KByte großen Speicherbereich, der von der lokalen

CPU mit Daten versorgt wird. Dies erleichtert die Softwareentwicklung und erlaubt den Einsatz des Moduls auch als Bus-Monitor für den CAN-Bus.

1.4. CAN Bustreiber

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

1.5. Lokale Intelligenz

Die Lokale CPU des PMC-CAN übernimmt folgende Aufgaben:

- ✍✍Das Initialisieren der CAN-Controller und Interface für die eingestellte Busgeschwindigkeit.
- ✍✍Das Empfangen von CAN-Botschaften unter Interrupt. Und das Abspeichern der Daten im Dual-Port-RAM.
- ✍✍Das einfache oder zyklische Senden von CAN-Telegrammen.
- ✍✍Die Bereitstellung einer lokalen Benutzerschnittstelle für ein VT-100 Terminal.

1.6. Softwareunterstützung

Die CAN-Controller der PMC-CAN werden von der eingebauten Firmware bedient. Für die Zugriffe auf das Dual-Port-RAM sind in den technischen Unterlagen entsprechende Programmbeispiele als ANSI C Source Code enthalten.
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
CPU-Clock	32 MHz
Lokale CPU	Motorola M-CORE 2107
Serielle Schnittstelle	RS-232, 9pol.SubD 19200 Baud, 8 Bit,NoParity
Abmessungen	149 x 74 x 10 mm
Stromaufnahme	5V / max. 400 mA
Betriebstemperatur	0 bis +50 °C
Lagertemperatur:	-25 bis +70 °C

1.8. Lieferumfang

Zum Lieferumfang der PMC-CAN gehört:

- Baugruppe PMC-CAN
- Technische Unterlagen PMC-CAN

1.9. Bestellbezeichnungen

PMC-Modul	
PMC-CAN	CAN Interface Modul mit 4 Kanälen je Kanal umschaltbar High-Speed-Treiber und Low-Speed-Treiber
PCI-Grundkarte	
PCI-PMC	PCI-Trägerkarte für ein PMC-Modul zum Einbau in PC's

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 PMC-CAN

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

2.1. Hardware-Einstellungen

Auf der PMC-CAN sind keine Einstellungen erforderlich.

2.2. Endmontage

Das PMC-CAN ist in den vorgesehenen Steckplatz zu stecken und auf der 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.

2.3. Software-Einstellungen

Es sind keine Softwareeinstellungen vorgesehen.

2.4. Lokaler Betrieb der Firmware mit Terminal

Das PMC-CAN Modul kann unabhängig vom Hostsystem ohne weitere Software in Betrieb genommen werden. Hierzu wird vom Hostsystem nur die 5V Betriebsspannung benötigt.
Nach dem Power-On startet die lokale CPU selbständig die CAN-Firmware. Über die RS-232 Schnittstelle kann das Modul nun mit einem ANSI VT100 Terminal bedient werden.
Als Terminal kann das Hyperterm unter Windows am PC benutzt werden.

2.4.1. Terminal Anschluss

Für den Anschluss ist ein PC-kompatibles Null-Modem Kabel erforderlich.

Pin	RS232
1	-
2	RxD input data
3	TxD output data
4	-
5	GND
6	-
7	-
8	-
9	-

Table 1: Pin layout Terminal-interface ST1

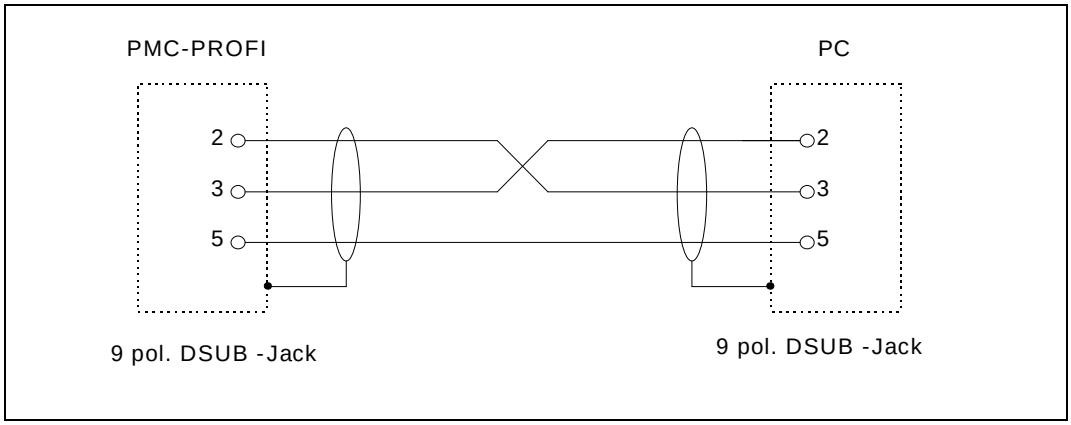

Figure 1: Interface cable PMC-PROFI ST1 - PC (DPKONFIG)

2.4.2. Com Schnittstellen Einstellungen

Die Einstellungen für die Schnittstelle lauten:

- 19200 Baud
- 8 Bit
- No Partity
- Kein Protokoll
- Terminalemulation ANSI VT100

2.4.3. Firmware Kommandos

Die Firmware Kommandos erlauben die Benutzung des PMC-CAN als einfachen CAN-Monitor und ermöglichen das einzelne oder zyklische Senden von Telegrammen.

2.4.3.1. B - Bus Bitrate

Auswahl der Übertragungsgeschwindigkeit am CAN-Bus. Gleichzeitig wird damit das richtige Physikalische Interface selektiert.

Syntax : B <Kanal > <Bitrate>

Kanal ist die Can-Kanal Nummer 1...4 zur Auswahl der Schnittstelle.
Bitrate ist eine Buchstabenkennung:

- L Lowspeed Interface 83,3 kBit/sec
- H Highspeed Interface 500 kBit/sec
- M Highspeed Interface 1 Mbit/sec

Beispiel : B 1L schaltet Kanal 1 auf Lowspeed 83,3 Kbit/sec

Es können so alle 4 Kanäle eingestellt werden. Die Einstellung wird mit dem folgenden Restart Kommando wirksam.

2.4.3.2. R – Restart

Löscht die Buffer und initialisiert die Can-Controller neu.

2.4.3.3. L – List Last Telegramme

Zeigt die letzten 20 empfangenen Telegramme an.

2.4.3.4. U – Update Übersichtsanzeige

Zeigt eine Empfangsübersicht nach ID-Nummern sortiert an

2.4.3.5. N – Next

Zeigt die nächste Seite der Empfangsübersicht an.

2.4.3.6. P – Previous

Zeigt die vorherige Seite der Empfangsübersicht an

2.4.3.7. S – Send Telegramm

Syntax S <Kanal> <Zyklus> <Identifier> <Byte> <Byte> ...

Kanal ist die Kanalnummer 1..4
Zyklus ist der Wiederholungszyklus in Millisekunden. 0 für einmaliges Senden.
Identifier ist die CAN-ID in Hexadezimal 0..7FF
Byte sind die Datenbytes. Maximal 8 Stück durch Leerzeichen getrennt.

Beispiel : S 2 50 100 55 AA
Auf Kanal 2 wird alle 50 ms die Botschaft 0x100 mit 2 Byte Länge und den Daten 0x55 und 0xAA gesendet.

Es können je Kanal bis zu 64 zyklische Sendetelegramme eingegeben werden.

2.4.3.8. Q – Quit

Can-Firmware beenden und picobug Debug-Monitor starten.
Picobug erlaubt das Debuggen von Software und die Kontrolle von Speicherinhalten.
Er kann zur Kontrolle der Daten im Dual-Port-RAM benutzt werden.
Mit ? kann eine Hilfe angezeigt werden.

2.5. Hostanbindung über PCI-Bus

Das PMC-CAN wird über eine 32-bit PCI-Bus Schnittstelle mit dem Hostsystem verbunden. Der PCI-Bus Interfacechip vom Typ PLX 9050 bindet das 128 kByte große DualPort-RAM des PMC-CAN in den Host-Memory-Adressraum ein.

2.6. Demontage

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

2.7. 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. 7.6) bei.

3. Funktionsgruppen der PMC-CAN

3.1. Allgemeines zum PMC-Modul

Die PMC-Module sind 149 * 74 mm große Steckbaugruppen. Über zwei Steckverbinder werden die Bus-Signale mit der Grundkarte verbunden. Durch die Anordnung der Schraubverbindung und der Steckverbinder entsteht eine stabile mechanische Einheit aus Grundkarte und Modul. Die Modul Frontplatte ragt durch die Frontplatte der Grundkarte und biedet Platz für die I/O Anschlüsse.
Der Bus von PMC-Modulen enspricht in seinen Signalen und in der Funktionslogik dem Standard PCI-Bus der PC's. Er ist nur mechanisch anders.

3.1.1. PMC-Modul-ID

Über den Ident-Bereich kann das Modul von der Software identifiziert werden.
Das PMC-CAN hat die Vendor-ID des Interface-Chips
PLX 9050 = 0x905010b5
und die Subvendor-ID von DMS
DMS PMC-CAN = 0x0002444D

3.1.2. Clock Signale

Das PMC-CAN hat einen eigenen Clockoszillator zur Erzeugung des 32-MHz CPU-Clocks und des 16 MHz CAN-Clocks. Aus diesem 16 MHz Signal erzeugt jeder SJA-1000 Can-Controller die Bit-Rate für seinen CAN-Bus Kanal.

3.1.3. PMC-Versorgungsspannungen

Das PMC-CAN wird mit +5V vom PCI-Bus versorgt. Die +12V und -12V Versorgungsspannungen sind nicht erforderlich.

3.1.4. CAN-Treiber

Die PMC-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. CAN-Anschluß

Das PMC-CAN hat eine 9-pol. D-Sub-Buchse an der Frontplatte für den CAN-Anschluß.
Die CAN-Anschlüsse haben keine Potentialtrennung !

Signal	Kanal 4	Kanal 3	Kanal 2	Kanal 1
Can_L	2	4	6	8
Can_H	1	3	5	7
GND	9	9	9	9

Tabelle 1: Pinbelegung der IO-Signale

Die Verbindung des GND Anschlusses mit der Fahrzeugmasse ist für die Lowspeed-Treiber wichtig !

3.2. Adressierung des PMC-CAN

Wie am PCI-Bus üblich, wird die Basisadresse des PMC-CAN Moduls in der System-Initialisierungsphase vom Betriebssystem des Hostrechners dynamisch zugewiesen (Plug & Play).
Das Anwenderprogramm erhält durch einen Systemaufruf die Adresse des Dual-Port-RAM's der PMC-CAN. Hierzu werden Vendor-ID, SubVendor-ID und Speichergröße übergeben.
Beispiel Microsoft Windows :

```
                /* ---------- Vendor ------ 9050 PLX -- CAN  DMS -- 128kB ---*/
p = (DPR_Typ*)  MapPciDevice( 0x905010b5, 0x0002444D, 0x20000) ;
```

4. Informationen für Systemprogrammierer

Das PMC-Profi kommt mit einer vorinstallierten Firmware von DMS. Wenn andere Funktionen benötigt werden, besteht jedoch auch die Möglichkeit eigene Software für die lokale M-Core CPU zu entwickeln.

4.1. Entwicklungsumgebung

Für die Motorola M-Core CPU steht von Motorola eine kostenlose GNU C Entwicklungsumgebung zur Verfügung. Mehr Komfort bietet die Metrowerks CodeWarrior Umgebung (ca. 6000.-DM).
Zum Download und zur Flashprogrammierung ist das Motorola „Enhanced Background Debug Interface“ erforderlich. (ca. 1400.-DM).
Die Firmware Version 1.2 ermöglicht ein Download eigener Programme über die serielle Schnittstelle.

4.2. Erstellung eigener Programme

Auch wenn eigene Programme die CAN-Monitor Firmware von DMS ersetzen sollen, ist es sinnvoll den PicoBug Monitor auf dem PMC-CAN zu belassen. Mit seiner Hilfe können Programme getestet und ins Flash-EPROM des Prozessors programmiert werden. Außerdem übernimmt er die Grundinitialisierung der Hardware sowie die Verwaltung der Intetrupts und die Ein/Ausgabe über die RS-232 Schnittstelle.
Die Kommunikation zwischen Anwendung und PicoBug geschieht über einen festen RAM-Bereich (ComRAM) auf der Adresse 0x8081FD00 im DPR.
An einfachsten ist es vom DMS Source-Code CanMon auszugehen. Er besteht im wesentlichen aus 3 Dateien:
Startm.asm ist ein C-startfile und sollte nicht geändert werden.
MinIO.c ist das Interface zum PicoBug und einige C-Funktionen für die Ein/Ausgabe und stellt die Interruptverwaltung zur Verfügung. Möglichst auch nicht ändern.
CanMon.c ist die eigentliche Anwendung. Sie kann nach Erfordernis geändert werden.

4.3. Update des Programms im Flash-EPROM

Um ein neues Programm ins Flash-Eprom des M-Cor-Prozessors zu programmieren, sind folgende Schritte nötig.

- ✍✍Der Programmier-Jumper J40 muss gesteckt sein. Die 5V Betriebsspannung muss ausreichend hoch sein (5,15 bis 5,25 V) da sie als Vpp benutzt wird. Wenn die Systemspannung nicht ausreicht, kannn das PMC-CAN außerhalb des Systems programmiert werden. Hierzu die Betriebsspannung von 5,25 V am ST3 einspeisen. Pin 1 + und Pin 7 GND.
- ✍✍Schließen Sie ein Terminal an (19200 Baud, 8 Bit, No Parity, kein Protokoll) und schalten Sie dann den PMC-CAN ein. Drücken Sie innerhalb von 2 Sekunden ein A, um den Autostart des CANMON zu unterbrechen. Der PicoBug meldet sich mit einem > Zeichen.
- ✍✍Löschen Sie die alte Firmware mit dem Kommando UE1 <Return>
- ✍✍Schließen Sie das HyperTerm-Fenster am PC

✍✍Erzeugen Sie aus Ihrer S-Record Datei eine Binär Datei Ihres Programms (zum Beispiel mit dem Programm exbin.exe, das in einer DOS-Box gestartet wird (exbin –r canmon.s conmon.rom)

✍✍Benutzen Sie nun pcuProg.exe, um die Binärdatei (canmon.rom) in den PMC-CAN zu übertragen. Zunächst im Menü Com die Schnittstelle Com1 wählen. Dann im Menü Bereich Programm 1 wählen. Dann die Taste Programm anklicken und die Datei im File-Dialog wählen.

✍✍Wenn die Übertragung beendet ist, schließen Sie PCUprog und öffnen Sie wieder Ihr HyperTerm, um dem PicoBug das Kommando run zu geben.

4.4. Update des Programms via Dualport-RAM

Ab der Firmware Version 1.3 liegen die Komunikationsbuffer für Serielle Ein-Ausgabe der Firmware im DPR. Damit können alle Funktionen der RS-232 Schnittstelle nun auch von der Host-CPU über das DPR gesteuert werden. Das Demo-Programm PMC_CAN.exe wurde um eine Funktion zum Programmdownload erweitert und kann anstelle der RS-232 Programmierung (Kapitel 4.3) benutzt werden (Taste PROG, Datei canmon.rom).
Der Souce-Code im Anhang kann zur Portierung auf ander Hostsysteme benuzt werden.

Achtung DPR Adressbelegung ab Firmware 1.3 geändert ! (Kapitel 4.5)
An eigenen Programmen sind folgende Änderungen nötig:
Die Definition von ComRAMbase wird 0x8081FD00.
Im startm.c wird nach dem jsri main ein bkpt Befehl eingefügt.
Das Programm muss mit KEYDOWN() prüfen ob Zeichen im Inputbuffer sind
Und mit Q beendet werden (return aus main).

4.5. Interne Adressen aus Sicht der M-Core CPU

0x 0000 0000	0x 0000 7FFF	Flash EPROM	32 KB	PicoBug
0x 0000 8000	0x 0001 FFFF	Flash EPROM	96 KB	Anwendung
0x 0080 0000	0x 0080 0FFF	Internes SRAM	4 KB	PicoBug
0x 0080 1000	0x0080 12FF	Internes SRAM	768 Byte	**Com-RAM bis V1.2 - jetzt frei**
0x 0089 1300	0x0080 1FFF	Internes SRAM	3,25 KB	Anwender RAM
0x 00C0 0000	0x 00DF FFFF	M-Core Register		
0x 8080 0000	0x 8081 FCFF	Dual-Port-RAM	127 KB	
0x 8081 FD00	**0x 8081 FEFF**	**DPR ComRAM**	**0,5 kB**	**Serial I/O**
0x 8081 FF00	**0x 8081 FF03**	**DPR Tick**		
0x 8081 FF04	0x 8081 FFFB	DPR Reserviert	0,25 kB	
0x 8081 FFFC	0x 8081 FFFF	DPR IRQ-Flags		
0x 8100 0000	0x 8100 01FF	Can-Controller	SAJ-1000	Kanal 1
0x 8100 0200	0x 8100 03FF	Can-Controller	SAJ-1000	Kanal 2
0x 8100 0400	0x 8100 05FF	Can-Controller	SAJ-1000	Kanal 3
0x 8100 0600	0x 8100 07FF	Can-Controller	SAJ-1000	Kanal 4

4.5.1. CAN-Controller

Die Programmierung des SJA 1000 Controllers ist dem Datenblatt zu entnehmen. Es sind nur Bytezugriffe erlaubt. Alle Register liegen auf den ungeraden Adressen. Also Control auf 0x81000001 und Command auf 0x81000003 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 !

4.5.2. Physical Layer Umschaltung

Das CAN-Interface kann über Port-Register der M-Core-CPU zwischen Low und High-Speed Physik umgeschaltet werden:
Portadresse 0x00CF991D
Kanal 1 Bit 0
Kanal 2 Bit 1
Kanal 3 Bit 2
Kanal 4 Bit 3
1 = Lowspeed 0 = Highspeed ;

4.5.3. Interrupts zum Hostsystem

Das PMC-CAN kann über den PCI-Bus ein Interrupt an das Host-System auslösen. Hierzu müssen die Register im 9050-PCI-Interface entsprechend den Host Erfordernissen programmiert werden.
Das Interrupt wird dann von der M-Core CPU durch Schreiben eines 16-Bit Wortes auf die Duaport-RAM Adresse 0x1FFFC gesetzt. Die Host-CPU kann es durch Lesen der Dualport RAM Adresse 0x1FFFC wieder zurücksetzen.

4.5.4. Interrupts vom Hostsystem

Das Hostsystem kann für die M-Core CPU des PMC-CANi ein Interrupt auslösen. Voraussetzung dafür ist, dass die Firmware im PMC-CAN den IRQ-5 freigibt und eine Interruptbearbeitungs-Routine installiert. Als Beispiel können hierfür die IRQ1-4 Routinen der Can-Controller dienen.
Das Hostsystem kann dann durch Schreiben der Dualport-RAM Adresse 0x1FFFE das Interrupt auslösen. Die M-Core-CPU muss es in der Interruptserviceroutine durch Lesen der Dualport-RAM Adresse 0x1FFFE wieder rücksetzen.

5. DMS CAN-Monitor

5.1. Dualport-RAM Struktur

Das Anwenderprogramm steuert alle Funktionen des PMC-CAN über Datenstrukturen im Dualport RAM.
Es gibt 4 Bereiche:

- Kontrollworte für Steuerfunktionen und Bitrateeinstellung
- Die Empfangs Übersicht nach ID-Nummern geordnet.
- Das Empfangs-FiFo
- Die Sende-Listen für jeden Kanal

Listing siehe Anhang

5.1.1. Daten Konsistenz

Empfangs-FIFO:
Die Firmware schreibt zuerst die Daten ins DP-RAM und stellt danach den Pointer weiter. Die Leseroutine des Hostsystems sieht dadurch immer konsistente Daten.

Sende-Listen:
Beim eintragen neuer Sendeaufträge sucht der Host zunächst einen freien Platz in der Liste (Flag = 0) dann trägt er die Daten ein. Zuletzt setzt er das Flag auf Sendebereit (Flag=1). Die Firmware beginnt danach mit dem Senden. Beim einmaligen Senden setzt die Firmware das Flag nach dem Senden wieder auf 0 und gibt so den Buffer wieder frei. Beim zyklischen Senden muss der Host den Auftrag durch Löschen des Flags beenden. Der Buffer darf frühestens 1 ms danach vom Host mit neuen Daten belegt werden.

Empfangs-Übersicht:
Die Daten in der Empfangsübersicht werden durch die Interruptroutinen in der Firmware ständig aktualisiert. Die Aktualisierung hat immer Vorrang. Der Host muss daher prüfen, ob während seines Auslesevorgangs die Daten aktualisiert wurden. Dazu dient das Sema Flag.
Der Ablauf ist folgender:
Der Host liest Sema in eine Temp-Variable ein und prüft dann ob im Temp das niederwertigste Bit gesetzt ist. (Temp ungeradzahlig). Ist dies der Fall, so läuft gerade eine Aktualisierung. Der Host muss den Vorgang wiederholen bis Temp geradzahlig ist.
Nun kann der die Daten kopieren. Nach dem Kopieren der Daten vergleicht er ob Temp immer noch gleich Sema ist. Wenn ja, wurden gültige und konsistente Daten übernommen. Wenn nicht, hat eine Aktualisierung während des Kopierens stattgefunden. Die Daten sind ungültig und der gesamte Vorgang muss wiederholt werden.
Die Wahrscheinlichkeit, dass eine Wiederholung nötig ist, hängt vom Verhältnis zwischen Übertragungszyklus der CAN-Botschaft und der Dauer des Kopierens ab. Das Kopieren dauert typisch weniger als 5 Mikrosekunden. Für eine Botschaft, die z.B. alle 10 ms gesendet wird, ist sie also 1 : 2000. Es ist maximal eine Wiederholung nötig.

5.2. Programmbeispiel für Windows

Das Beispielprogramm zeigt das Auslesen des Empfangs-FIFO's unter Windows. Das Programm wurde mit dem Borland C++-Bilder erstellt.

Listing siehe Anhang

5.3. Empfang der CAN-Botschaften

Alle CAN-Botschaften werden ungefiltert empfangen und von der Empfangsinterruptroutine in die Empfangsübersicht eingetragen. Wenn das Meldebit im Kanal-Byte gesetzt ist,wird die Botschaft auch in das Empfangs-FIFO eingetragen. Nach der Initialisierung sind alle Meldebits bei allen Identifiern gesetzt, es werden also alle Botschaften ins FIFO eingetragen. Die Host Software kann bei unerwünschten ID's die Meldebits löschen und so eine Filterung für das Empfangs-FIFO einstellen.
Jede Empfangsbotschaft erhält einen Zeitstempel (RxTime) in ms.
Eigene Sendungen werden nicht empfangen.

5.4. Senden von CAN-Botschaften

Für jeden Kanal gibt es eine eigene Sendeliste mit 64 Plätzen. Platz 0 hat die höchste, Platz 64 die niedrigste Priorität. Einmal pro Millisekunde wird jede Liste durchsucht und die Botschaft gesendet, die unter den sendebereiten die höchste Priorität hat. Dadurch werden mehrere zyklische Botschaften auf verschiedene Millisekunden verteilt, auch wenn sie die gleiche Zykluszeit haben.
Je Kanal wird maximal eine Botschaft je Millisekunde gesendet.

6. Wartung und Instandhaltung

Es ist keine Wartung der PMC-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".

7. Pläne und Anhang

7.1. Liste der verwendeten Abkürzungen

ASCII	American Standard Code for Information Interchange 7 Bit Zeichencode
CPU	Central Processing Unit
PMC	Modul Standard IEEE P1386
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

7.2. Literaturhinweise

[1] DMS Kataolg, Dorsch Mikrosystem GmbH
[2] Datenblatt SJA1000 Stand alone CAN Controller, Philips
[3] Datenblatt PLX 9050 PCI-Interface, PLX
[4] M-Core MMC 2107 Advance Information, Motorola

7.3. Listing PMC_CAN.h

```
/* ------------------------------------------------------
   PMC-CAN Dual-Port-RAM
   =====================
   Version 1.0
   ------------------------------------------------------*/

/* ------- DMS Typen ---------------------- */
#ifndef dmstyps
#define dmstyps
typedef unsigned char  byte ;   /*  8-Bit */
typedef unsigned short wort ;   /* 16-Bit */
typedef unsigned int   lwort ;    /* 32-Bit */
#endif


/* Für jeden Kanal eine eigene Sendeliste mit 64 Einträgen
   ------------------------------------------------------- */

#define maxSL 64

typedef
volatile struct {   wort   Id ;             /* CAN Id */
                    wort   Flag ;           /* Flag 0 = frei, 1 = sendebereit*/
                    wort   TxCyc ;          /* 0 = einmalig, n = Zyklus in ms */
                    byte   Status ;         /* 0 = kein Fehler */
                    byte   ln ;             /* Anzahl der Datenbyte's */
                    byte   data[8] ;        /* Daten */
                    lwort  TxTime ;         /* Nächster Sendezeitpunkt wird vom PMC gesetzt */
                } SendList_Typ ;            /* 20 Byte */



/* Ein Empfangs FIFO mit 2500 Plätzen
   ---------------------------------- */

#define maxFifo 2500

/* Empfangene Telegramme deren Melde-Flag = 1 ist werden ins
   Empfangs-FIFO eingetragen.
   Das Empfangs FIFO ist ein Ring-Buffer mit 2500 Plätzen */

typedef struct {    wort   Id ;             /* CAN Id */
                    byte   Kanal ;          /* Kanal Nr 1 .. 4 */
                    byte   ln ;             /* Anzahl der Datenbyte's */
                    byte   data[8] ;        /* Daten */
                    lwort  RxTime ;         /* Empfangszeitpunkt in ms */
                } RxBlock_Typ ;              /* 16 Byte */
```

```
typedef struct {    wort   Size ;          /* Max Anzahl der Einträge (PMC) */
                    wort   OutPnt ;        /* Lese-Index (USER) */
                    wort   InPnt ;         /* Schreib-Index (PMC) */
                    byte   Status ;        /* Status 0 = OK, n = Overrun (PMC) */
                    byte   ElementSize ;   /* Grösse eines RxBlocks in Byte (PMC) */
                    RxBlock_Typ Data[maxFifo] ;
                } RxFiFo_Typ ;             /* 40008 Byte */


/* Eine Empfangs Übersicht mit je einem Eintrag für jeden Identifier (0..2047)
   ------------------------------------------------------------------------- */

typedef struct {
                    byte   Sema ;          /* Semafore für konsistentes Lesen (PMC) */
                    byte   Kanal ;         /* Bit 4..7 Kanal Flag, Bit 0 .. 3 Melde-Flag's */
                    int    RxTime ;        /* Empfangszeit in ms (PMC) */
                    int    RxCnt ;         /* Anzahl der empfangenen Telegramme (PMC) */
                    byte   Status ;        /* Controller Status 0 = OK (PMC) */
                    byte   ln ;            /* Daten Länge (PMC)*/
                    byte   Daten[8] ;      /* Daten (PMC)*/
                    } Telegram_Typ;        /* 20 Byte */



/* Dual Port RAM  Aufteilung PMC-CAN (max 128 kB)
   ---------------------------------------------- */

typedef struct {
                   wort     Modul_Status ;          /* 1 = Run 2 = Reinit */
                    wort    Firmware_Version ;      /* 0x0010 = 1.0 */
                    char    Bus_Speed_1 ;           /* L=83,3k , H=500k ; M=1MBit/sec */
                    char    Bus_Speed_2 ;           /* L=83,3k , H=500k ; M=1MBit/sec */
                    char    Bus_Speed_3 ;           /* L=83,3k , H=500k ; M=1MBit/sec */
                    char    Bus_Speed_4 ;           /* L=83,3k , H=500k ; M=1MBit/sec */
                    wort    CW[12] ;                /* reserve */
                    Telegram_Typ  Tel[2048] ;       /* Empfangs Übersicht  40960 Byte */
                    SendList_Typ  SL1[maxSL] ;      /* Sendeliste Kanal 1   1280 Byte */
                    SendList_Typ  SL2[maxSL] ;      /* Sendeliste Kanal 1   1280 Byte */
                    SendList_Typ  SL3[maxSL] ;      /* Sendeliste Kanal 1   1280 Byte */
                    SendList_Typ  SL4[maxSL] ;      /* Sendeliste Kanal 1   1280 Byte */
                    RxFiFo_Typ    RxFiFo ;          /* Empfangs FiFo      40008 Byte */
                } DPR_Typ ;
```

7.4. Listing "CAN-Demo" für PMC-CAN

```
   //---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "CAN.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
// ---------------------------
#include <dos.h>
#include <stdio.h>
#include <time.h>
#include <errno.h>
#include "..\defs\pmc_can.h"

typedef unsigned char byte ;
typedef unsigned short wort ;
typedef unsigned int lwort ;

typedef volatile struct { short anz ;
                short in  ;
                short out ;
                short res ;
                char  data[248] ;
              } BBuffer ;


typedef volatile struct {
     BBuffer InchBuf ;
     BBuffer OutBuf ;
     lwort   tick   ;
     } ComRAM_Typ ;



/* ---------- Globale Variablen ------------------- */
char DispStr[256];
char Fehler[256] ;
char TmpStr[256];
volatile DPR_Typ* DPR ;
volatile ComRAM_Typ* ComRAM ;

// --- Aus Unit PCI.cpp ---
ULONG         MapPciDevice(ULONG DeviceVendor,ULONG SubVendor,ULONG MemSize) ;
long  PCI_DeInit(void);
```

```
// --- Unterprogramme ---------------------------- */
wort SwapW (wort w)
{
  return ( ((w & 0x00FF)<<8)
         |((w & 0xFF00)>>8) ) ;
}

lwort SwapL (lwort w)
{
  return ( ((w & 0x000000FF)<<24)
         |((w & 0x0000FF00)<<8)
         |((w & 0x00FF0000)>>8)
         |((w & 0xFF000000)>>24) ) ;
}

void WaitMS (int Time) /* Wartet Time Millisecunden */
{
          Sleep(Time);
}


short BBufRdy (BBuffer *B)
{
short a ;
    a = SwapW(B->in) - SwapW(B->out) ;
    if (a<0) a= a + SwapW(B->anz) ;
    return (a) ;
}

char BBufOut (BBuffer *B)
{
char z ;
short n ;
    if (B->out == B->in) return (0) ;
    n = SwapW(B->out) ;
    z = B->data[n] ;
    n++ ;
    if (n == SwapW(B->anz)) B->out = 0 ;
                    else  B->out = SwapW(n);
    return (z) ;
}

void BBufIn (BBuffer *B , char d)
{
short n ;
    n = SwapW(B->in) ;
    B->data[n] = d;
    n++ ;
    if (n == SwapW(B->anz)) B->in = 0 ;
                    else  B->in = SwapW(n);
```

```
}

void OUTCH(char Zeichen)  /* Output 1 Zeichen ans PMC-Can */
{
  ComRAM->OutBuf.res = 0xFFFF ; /* RS-232 Ausgabe IRQ ausschalten ! */
  while  (BBufRdy(&(ComRAM->InchBuf)) > 246) WaitMS(10);
  BBufIn(&(ComRAM->InchBuf),Zeichen);
}


short KEYDOWN (void) /* Anzahl der Zeichen im Input Buffer */
{
        return (BBufRdy(&(ComRAM->OutBuf))) ;
}

char INCHNW (void) /* Input 1 Zeichen vom PMC-Can  */
{
   while ( BBufRdy(&(ComRAM->OutBuf))==0 )  WaitMS(10);
   return (BBufOut(&(ComRAM->OutBuf))) ;
}

void OUTS (const char *TextString)
{
 while ( *TextString >0 )  {
   OUTCH (*TextString) ;
   TextString++ ;
   } ;
}

char INCHB (void)
{
  return INCHNW() ;
}



wort CalcParity(wort *addr,lwort size)
{
    wort    parity;
    lwort   i,cnt;

    cnt = size / 2;
    parity = 0;
    for (i=0;i<cnt;i++)
    {
        parity  ^= addr[i];
    }

    if (size & 1) parity ^= ((byte*) addr)[size-1];
    return(parity);
}
```

```
int WaitforKey(long msTimeOut)
{
        long    TimeOut;

        if (KEYDOWN())return(1);
        else
        {
                TimeOut = 0;
                msTimeOut/=20;
                while (!KEYDOWN())
                {
                        WaitMS(20);
                        TimeOut++;
                        if (TimeOut >= msTimeOut) return(0);
                }
        }
   return(1);
}


int WaitforSync(void)
{
        char Zeichen,ende=0;

        while (!ende)
        {
                if (WaitforKey(5000))
                {
                        Zeichen =  INCHB();
                        if (Zeichen == 0) ende = 1;
                }
                else
                {
                        return(0);
                }
        }
        return(1);
}

void OUTBuffer(char *buffer,int cnt)
{
  while (cnt)
    {
      OUTCH (*buffer++) ;
      cnt-- ;
    } ;
}

void OUTBIN(byte Z)
```

```
{
  OUTCH(Z) ;
}



int SendBin(byte *buffer,int cnt)
{
        int     i,rcnt;
        char    found;
        int             maxtrans;
   wort *saveadr,parity,checkparity;

   rcnt = cnt;
   saveadr = (wort*) buffer;

        OUTBIN(0);
        if (!WaitforSync()) return(1);
        while(cnt > 0)
        {
                found = 0;
                if (cnt < 3) i = cnt;
                else
                {
                        if (cnt > 127) maxtrans = 127;
                        else                    maxtrans = cnt;

                        for (i=0;i<maxtrans-2;i++)
                        {
                                if (buffer[i] == buffer[i+1])
                                {
                                        if (buffer[i+1] == buffer[i+2])
                                        {
                                                found = 1;
                                                break;
                                        }
                                        else i++;
                                }
                        }/* for */
                }
                /*  Ungleichen Senden */
                if (i)
                {
                        OUTBIN((char)i);
                        OUTBuffer((char*) buffer,i);
                        buffer+=i;
                        cnt-=i;
                        if (!(WaitforKey(5000) && (INCHB() == (char) 0x80)))
                        {
                                return(1);
                        }
```

```
                }
                if (found)
                {/* Anzahl gleicher Bytes ermitteln */

                i = 3;/* In jedem Falle gleich */

                        if (cnt > 127) maxtrans = 127;
                        else maxtrans = cnt;

                        while ((buffer[0] == buffer[i])&& i<maxtrans)i++;
                        OUTBIN(i+0X80);
                        OUTBIN(buffer[0]);
                        buffer+=i;
                        cnt-=i;
                        if (!(WaitforKey(5000) && (INCHB() == (char) 0x80)))
                        {
                                return(1);
                        }
                }
        }
        OUTBIN(0);

   parity = CalcParity(saveadr,rcnt);
        if (!(WaitforKey(5000))) return(1);
   checkparity = INCHB();
        if (!(WaitforKey(5000))) return(1);
        checkparity += ((wort) INCHB())<< 8;
   if (checkparity != parity)
    {
      sprintf (DispStr,"Patity %04X %04X %d", checkparity, parity, rcnt) ;
      Form1->Display->Lines->Add(DispStr) ;
      return(2);
    };
        return(0);
}


//---------------------------------------------------------------------------

int RxGet (RxBlock_Typ* RxB)
{
volatile RxBlock_Typ* Fifo ;
wort n,m ;

   n = SwapW(DPR->RxFiFo.OutPnt) ;
   if (n == SwapW(DPR->RxFiFo.InPnt)) return 0 ;
   Fifo = &(DPR->RxFiFo.Data[n]) ;
   RxB->Id    = SwapW(Fifo->Id) ;
   RxB->Kanal = Fifo->Kanal ;
   RxB->ln    = Fifo->ln ;
   for (m=0;m<RxB->ln;m++) RxB->data[m] = Fifo->data[m] ;
```

```
    RxB->RxTime = SwapL(Fifo->RxTime) ;
    if (n<SwapW(DPR->RxFiFo.Size)) n++ ;
                    else n = 0 ;
    if (n == SwapW(DPR->RxFiFo.InPnt)) DPR->RxFiFo.Status = 0 ;
    DPR->RxFiFo.OutPnt = SwapW(n) ;
    return 1 ;
}


int CanSend ( int Kanal, int ID, int RepMs, int Ln, byte * Data)
{
 SendList_Typ * SL ;
 int n,m;
   switch (Kanal) {
     case 1 : SL = &(DPR->SL1[0]) ;
              break ;
     case 2 : SL = &(DPR->SL2[0]) ;
              break ;
     case 3 : SL = &(DPR->SL3[0]) ;
              break ;
     case 4 : SL = &(DPR->SL4[0]) ;
              break ;
     default : return (-1) ;
   } ;
  n=0;
  while (SL[n].Flag) { n++ ; if (n == maxSL) return (-1 ); };
  SL[n].Id = SwapW(ID) ;
  SL[n].TxCyc = SwapW(RepMs) ;
  SL[n].ln = Ln ;
  for (m=0;m<Ln;m++) SL[n].data[m] = Data[m] ;
  SL[n].Flag = SwapW(1) ;
  return n ;
}



//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
long dummy;
byte *Base ;
  sprintf( DispStr, "Test für PMC-CAN v1.0") ;
  Form1->Caption = DispStr;
// ---------- Ventor ------ 9050 PLX -- CAN  DMS -- 128kB ---
     dummy = MapPciDevice( 0x905010b5, 0x0002444D, 0x20000) ;
     if (dummy)
       {
        Base = (byte*) dummy ;
        sprintf ( DispStr, " BasisAdresse .....: %X", Base );
        Display->Lines->Add(DispStr) ;
```

```
       DPR = (volatile DPR_Typ*) Base ;
       ComRAM = (volatile ComRAM_Typ*) (Base + 0x1FD00) ;
      }
    else
     {
      sprintf (DispStr,"Device nicht gefunden") ;
      Display->Lines->Add(DispStr) ;
     } ;
}

void Display_Antwort(char w)
{
int n ;
char z ;
  DispStr[0] = 0 ;
  n = 0 ;
  do
    {
    NextIn:
      z = INCHNW() ;
      if (z == 0x0D)
         {
           DispStr[n] = 0 ;
           Form1->Display->Lines->Add(DispStr) ;
           n = 0 ;
           goto NextIn ;
         } ;
      if (z == 0x0A) goto NextIn ;
      DispStr[n] = z ;
      n++ ;
      if ((w==0) && (KEYDOWN()==0)) break ;
    } while (z != w) ;
  DispStr[n] = 0 ;
  Form1->Display->Lines->Add(DispStr) ;
}

//---------------------------------------------------------------------------
//   Programmieren
//---------------------------------------------------------------------------
void __fastcall TForm1::ProgramClick(TObject *Sender)
{
#define maxln 114688
 byte Dbuf[maxln] ;
 char fn[80] ;
 FILE *fp ;
 int ln ;
 int err ;

       sprintf ( DispStr, " Call Monitor " );
       Display->Lines->Add(DispStr) ;
       OUTS("Q") ; OUTCH(0x0D) ;
```

```
WaitMS (300) ;
OUTS("A") ; OUTCH(0x0D) ;
WaitMS (300) ;
Display_Antwort(0) ;

sprintf ( DispStr, " Löschen Flasch " );
Display->Lines->Add(DispStr) ;
OUTS("ue1") ; OUTCH(0x0D) ;
WaitMS (300) ;
Display_Antwort('>') ;

strcpy(fn,"C:\CanMon.rom") ;
sprintf ( DispStr, " Laden File %s ",fn );
Display->Lines->Add(DispStr) ;
fp = fopen (fn,"rb");
ln = 0 ;
if (fp)
 {
   while (!feof(fp))
    {
     Dbuf[ln] = fgetc(fp) ;
     ln++ ;
     if (ln>maxln)
       {
         sprintf ( DispStr, " *** File zu lang " );
         Display->Lines->Add(DispStr) ;
         break ;
       } ;
    };
   ln-- ;
   fclose(fp) ;
  } ;

sprintf ( DispStr, " File mit %d Byte geladen ",ln );
Display->Lines->Add(DispStr) ;
if (ln)
 {
   sprintf ( DispStr, " Programmieren Flasch " );
   Display->Lines->Add(DispStr) ;
   OUTS("uu1") ; OUTCH(0x0D) ;
   if (ln & 1) ln++ ;
   err = SendBin(&Dbuf[0],ln);
   sprintf ( DispStr, " Error Code = %d ",err );
   Display->Lines->Add(DispStr) ;
   OUTCH(0x0D) ;
   WaitMS (300) ;
   Display_Antwort('>') ;
   if (err == 0)
    {
     OUTS("run") ; OUTCH(0x0D) ;
     ComRAM->OutBuf.res = 0x0 ;   /* Nun wieder RS-232 Ausgabe */
```

```
          } ;
        } ;
}


//---------------------------------------------------------------------------

void __fastcall TForm1::StartClick(TObject *Sender)
{
int n,m ;
char temp[10] ;
RxBlock_Typ RxBuf ;
byte SendDaten[] = {0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88} ;

     DPR->Bus_Speed_1 = 'L' ;
     DPR->Bus_Speed_2 = 'L' ;
     DPR->Bus_Speed_3 = 'L' ;
     DPR->Bus_Speed_4 = 'L' ;
     DPR->Modul_Status = SwapW(2) ;
     WaitMS (1000) ;

    CanSend(1,101,0,8,&SendDaten[0]) ;
    CanSend(2,102,0,8,&SendDaten[0]) ;
    CanSend(3,103,0,8,&SendDaten[0]) ;
    CanSend(4,104,0,8,&SendDaten[0]) ;
    WaitMS (100) ;
    n = 0 ;
    while (RxGet(&RxBuf))
     {
       sprintf (DispStr,"%8d  Kanal %1d ID: %04X mit %1d Byte :",
             RxBuf.RxTime, RxBuf.Kanal, RxBuf.Id, RxBuf.ln) ;
       for (m=0;m<RxBuf.ln;m++)
        {
          sprintf(temp," %02X", RxBuf.data[m]) ;
          strcat(DispStr,temp) ;
        } ;
       Display->Lines->Add(DispStr) ;
       n++ ;
     } ;
     if (n==12)
     { Display->Lines->Add("+++ LowSpeed O.K. +++");

     DPR->Bus_Speed_1 = 'M' ;
     DPR->Bus_Speed_2 = 'M' ;
     DPR->Bus_Speed_3 = 'M' ;
     DPR->Bus_Speed_4 = 'M' ;
     DPR->Modul_Status = SwapW(2) ;
     WaitMS (1000) ;

    CanSend(1,101,0,8,&SendDaten[0]) ;
    CanSend(2,102,0,8,&SendDaten[0]) ;
```

```
    CanSend(3,103,0,8,&SendDaten[0]) ;
    CanSend(4,104,0,8,&SendDaten[0]) ;
    WaitMS (100) ;
    n = 0 ;
    while (RxGet(&RxBuf))
     {
      sprintf (DispStr,"%8d  Kanal %1d ID: %04X mit %1d Byte :",
            RxBuf.RxTime, RxBuf.Kanal, RxBuf.Id, RxBuf.ln) ;
      for (m=0;m<RxBuf.ln;m++)
       {
         sprintf(temp," %02X", RxBuf.data[m]) ;
         strcat(DispStr,temp) ;
       } ;
      Display->Lines->Add(DispStr) ;
      n++ ;
     } ;
     if (n==12) Display->Lines->Add("+++ HighSpeed O.K. +++");
     } ;


}
//---------------------------------------------------------------------------
```

7.5. 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: _____

7.6. 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

_____

_____

_____