Vernetzung intelligenter I/O-Module mit CAN

Dr. Claus Kühnel

Ethernet-TCP/IP hat sich für die Vernetzung von PCs seit langem durchgesetzt und ist auch zunehmend in der Automatisierungstechnik zu finden. Zur Vernetzung von intelligenten I/O-Modulen, Sensoren oder Aktoren sind die erforderlichen Aufwendungen aus dem resultierenden Overhead aber nicht gerechtfertigt. Häufig stehen außerdem die Ressourcen an dieser Stelle überhaupt nicht zur Verfügung.

Im folgenden Beitrag sollen unterschiedliche I/O-Module über das Controller Area Network (CAN) vernetzt werden. In die Vernetzung ist ein PC einbezogen, der dann seinerseits eine Ethernet-TCP/IP-Schnittstelle an ein übergeordnetes System zur Verfügung stellen kann. Der letztgenannte Aspekt soll hier aber keiner weiteren Vertiefung unterliegen.

Ziel des Beitrags ist es, mit dem noch zu betrachtenden Netzwerk zu experimentieren und die unterschiedlichen Herangehensweisen zu verdeutlichen. Der Schwerpunkt liegt dabei auf den Grundfunktionen, mit deren Beherrschung dann eine konkrete Applikation aufgebaut werden kann.

1. Einige Grundlagen zu CAN

Die Grundlagen zu CAN sind an vielen Stellen in Büchern und im Web zu finden und sollen hier nicht explizit wiederholt werden. Im Literaturverzeichnis sind Hinweise auf CAN-Grundlagen zu finden [1][2][3][4][5]. Einen sehr guten Überblick bietet der Beitrag in [1]. Einige wichtige Aspekte, die bei CAN unterschiedlich zu anderen Systemen der Datenübertragung sind, sollen aber herausgestellt werden.

Üblicherweise erfolgt der Datenaustausch zwischen zwei Teilnehmern. Ein Master adressiert einen Slave und zwischen diesen beiden Teilnehmern werden die Daten übermittelt.

Bei CAN erfolgt der Datenaustausch nach dem sogenannten "Producer-Consumer Prinzip". Eine von einem beliebigen Teilnehmer (Producer) gesendete Nachricht kann von allen anderen Teilnehmern (Consumer) übernommen werden. Die zu versendenden Nachrichten werden über einen eindeutigen Identifier gekennzeichnet.

Im CAN-Protokoll werden Nachrichten üblicherweise mit einem 11-Bit langen Identifier gekennzeichnet (Standardformat, Standard Frame). Für Sonderanwendungen ist jedoch auch die Verwendung von 29-Bit langen Identifiern (Erweitertes Format, Extended Frame) möglich. Abbildung 1 zeigt den Aufbau eines Standard-Frames mit den entsprechenden Erläuterungen.

BUS IDLE	SOF	ID	RTR	IDE	RO	DLC	DATA	CRC	ACK	EOF	INT	BUS IDLE

Name	Bits	Bedeutung
SOF	1	Start-of-Frame Bit
ID	11	Identifier
RTR	1	Remote-Transmission-Request Bit
DIE	1	Identifier-Extension Bit
RO	1	Reserved Bit
DLC	4	Data-Length Control
DATA	0...8x8	Data (0 to 8 Bytes)
CRC	15	Cyclic Redundancy Check
ACK	2	Acknowledge Bits
EOF	7	End-of-Frame (7 recessive Bits)
INT		Inter-Frame Space

Abbildung 1 CAN-Standard-Frame

Jeder Teilnehmer eines CAN-Netzwerkes (Knoten) muss die für ihn relevanten Nachrichten anhand der Identifier aus dem auf dem Bus verfügbaren Nachrichtenstrom herausfiltern.

Jeder Teilnehmer kann mit dem Senden einer Nachricht beginnen, sobald der Bus frei ist. Es kann vorkommen, dass gleichzeitig mehr als ein Knoten mit dem Versenden einer Nachricht beginnt. Das Arbitrierungsverfahren stellt sicher, dass schließlich nur ein Teilnehmer mit dem Senden seiner Nachricht fortfährt. Im Rahmen der Arbitrierung wird anhand des Identifiers die Nachricht mit der höchsten Priorität ermittelt. Je niedriger der Wert des Identifiers, desto höher ist die Priorität der betreffenden Nachricht.

Jeder Teilnehmer im Netzwerk kann einen Nachrichtentransfer initiieren, wodurch ein direkter Nachrichtenaustausch zwischen allen Teilnehmer des Netzwerks möglich ist.

2. Zu betrachtendes Netzwerk

Das zu betrachtende Netzwerk besteht aus einem PC, auf dem die Entwicklungstools einschließlich eines CAN-Interfaces laufen und verschiedenen I/O-Modulen. Die I/O-Module kommen von unterschiedlichen Herstellern und weisen unterschiedliche Möglichkeiten der Programmierung auf. Abbildung 2 zeigt ein Blockschema des zu betrachtenden Netzwerks. Folgende Module kommen darin zum Einsatz:

- CAN-Interface am PC (USB-CAN-Interface bzw. RS232-CAN-Interface)
- Digitales Eingabe-Modul CST-DI8-TTL
- Digitales Ausgabe-Modul CST-DO8-TTL
- CANDIP/M162 Starter Kit mit ATMEL ATmega162 und Activity Board ACB1
- MicroMod Evaluationboard mit PCAN MicroMod-Modul (MB90F497)
- STK500/501 Starterkit mit ATMEL AT90CAN128

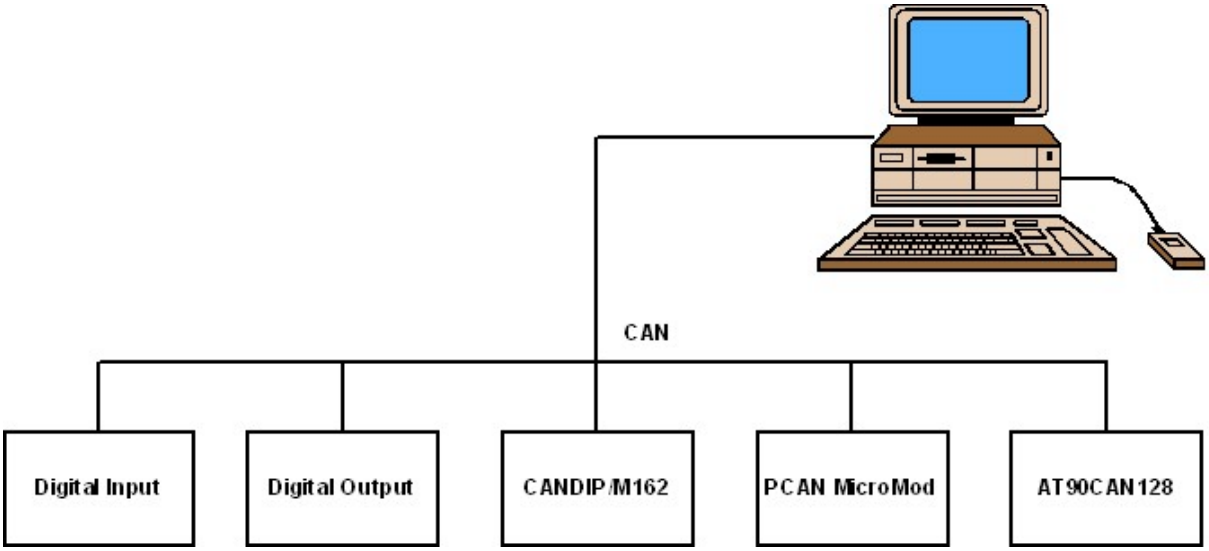


Abbildung 2 Vernetzte I/O-Module

Die Nachrichten (CAN-Mitteilungen) werden über ein zweiadriges geschirmtes "twisted pair" Kabel übertragen. Stromversorgung ist optional. Der Anschluss an den CAN-Bus erfolgt über 9-polige D-SUB-Stecker, deren Pinbelegung dem CiA-Normierungsvorschlag DS 102-1 entspricht. In Feldgeräten werden meist 5-polige M12-Steckverbinder verwendet.

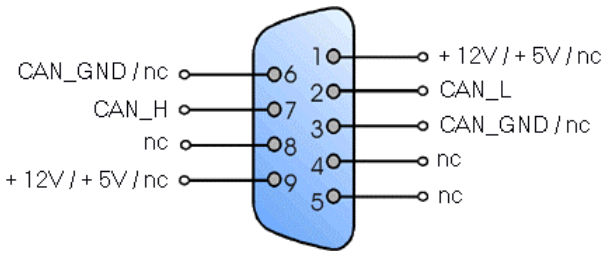


Abbildung 3 Pinbelegung gemäss DS102-1 (CiA)

Der CAN-Bus wird in Linienstruktur aufgebaut. Die Enden der Linien sind mit einem Abschlusswiderstand von 120 Ω zu versehen.

Die Leitungslänge kann bei einer Baudrate von 1 MBit/s 40 m, bei 100 kBit/s 500 m und bei 10 kBit/s 1000 m betragen.

3. Verwendete CAN-Tools

Zur Inbetriebnahme und Konfiguration der einzelnen CAN-Knoten (I/O-Module) verwendet man im allgemeinen spezielle Tools, die auf einem Entwicklungs-PC installiert werden. Der Entwicklungs-PC muss seinerseits auch mit einem CAN-Interface ausgestattet werden.

Für die vorzustellenden Experimente wurde auf den Einbau einer CAN-Interfacekarte in den Entwicklungs-PC verzichtet. An dessen Stelle wurden ein USB-CAN-Interface von PEAK-System Technik (D) bzw. ein RS232-CAN-Interface von LAWICEL (S) eingesetzt.

3.1. USB-CAN-Interface von PEAK-System Technik

Das "PC-USB zu CAN Interface" [www.peak-system.com/db/de/pcanusb.html] wurde für die einfache und kostengünstige Anbindung in CAN-Netzwerke mit einer maximalen Baudrate

von 1 MBit/s entwickelt. Das USB-CAN-Interface ist in einem Kunststoffgehäuse untergebracht und durch seine geringe Größe optimal für den Einsatz an Laptops und Notebooks geeignet. Abbildung 4 zeigt das USB-CAN-Interface von PEAK-System Technik.

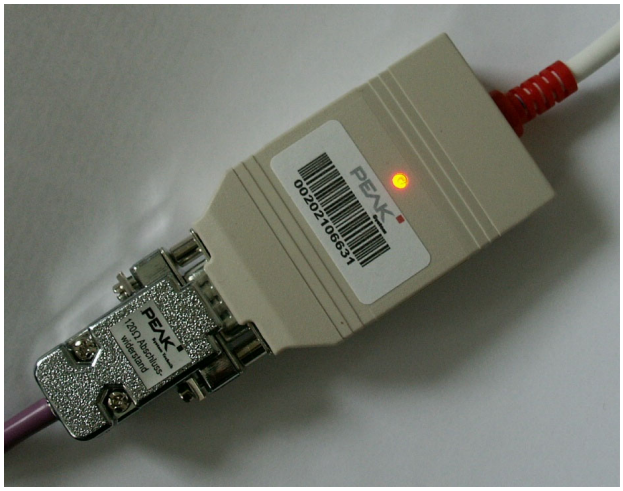


Abbildung 4 USB-CAN-Interface von PEAK-System Technik

Zusammen mit dem USB-CAN-Interface erhält man vom Hersteller das Programm PCANView, welches für die hier vorgenommenen Inbetriebnahmen vollkommen ausgereicht hat. Abbildung 5 zeigt einen Screenshot dieses Tools.

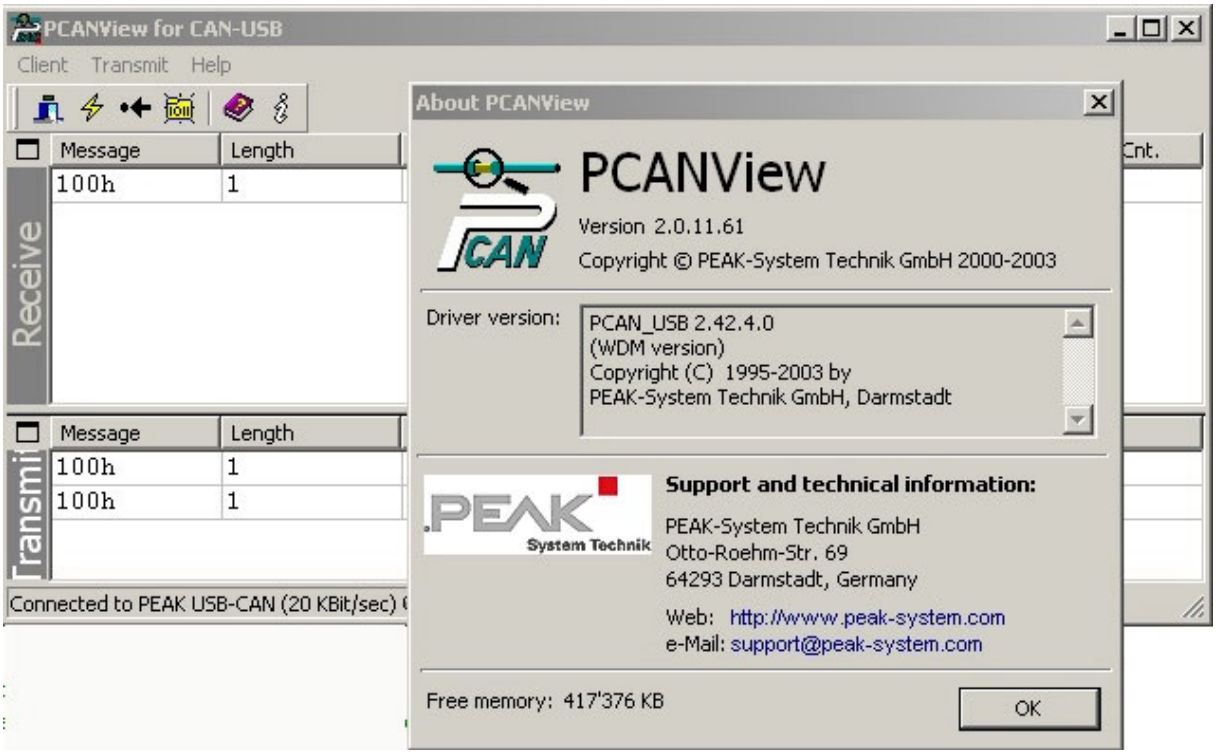


Abbildung 5 Screenshot PCANView

In Abschnitt 4.3.3 wird die Arbeit mit diesem Programm noch anhand eines konkreten Beispiels betrachtet, so das an dieser Stelle auf weitere Informationen verzichtet werden kann.

3.2. CAN232 von LAWICEL

CAN232 [www.can232.com] ist kleines Interface, welches an ein COM-Port eines PCs oder eine beliebige RS232-Schnittstelle eines Mikrocontrollers angeschlossen, Verbindung zu einem CAN-Netzwerk aufnimmt. Abbildung 6 zeigt das CAN232-Interface. Die übergeordnete Software muss nur eine RS232-Schnittstelle bedienen und kann auf spezielle Treiber verzichten. Die CAN-Mitteilungen werden in ASCII-Format zu übermittelt.

Abbildung 6 CAN232 Dongle

Das CAN232-Interface kann von einem Terminalprogramm oder einem Anwenderprogramm aus konfiguriert und betrieben werden. Abbildung 7 zeigt einen Hyperterminal-Mitschnitt, der nachträglich kommentiert wurde.

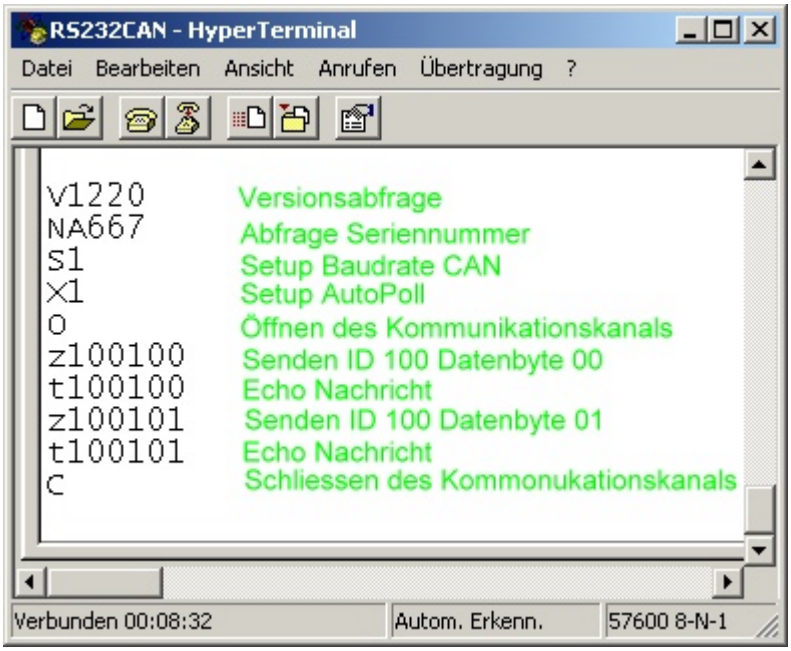


Abbildung 7 Konfiguration und Kommunikation mit Hyperterminal

Die Kommandos V und N dienen der Abfrage von Versions- und Seriennummer des CAN232-Interfaces. Das Kommando S1 setzt die Baudrate auf einen Wert von 20 kBit/s und mit X1 wird das AutoPolling nach empfangenen CAN-Mitteilungen eingeschaltet. Nach diesen Vorbereitungen kann der Kommunikationskanal geöffnet werden. Durch das Kommando t100100 wird eine CAN-Mitteilung mit dem Identifier 0x100 und einem Datenbyte

(DL=1) mit dem Wert 0x00 verschickt. Durch z wird das erfolgreiche Versenden gemeldet. Der Empfänger sendet hier die CAN-Mitteilung unverändert als Echo zurück, was in der Zeile unter dem ersten z ersichtlich wird. Durch das Kommando t100101 wird eine CAN-Mitteilung mit dem Identifier 0x100 und einem Datenbyte (DL=1) mit dem Wert 0x01 verschickt. Hier wiederholt sich das gleiche Spiel und schließlich wird durch das Kommando C der Kommunikationskanal geschlossen.

4. I/O-Module

Nach dem die verwendeten Tools bekannt gemacht sind, können wir uns die einzelnen I/O-Module und deren Konfiguration ansehen. Dort, wo es möglich ist, wollen wir überschaubare Anwendungsbeispiele auf das betreffende I/O-Modul laden.

4.1. CST - Modulbaureihe CAN im Stecker

Die CAN-Module der CST-Reihe wurden von der Fa. EMS Dr. Thomas Wünsche [www.ems-wuensche.com] als dezentrale I/O-Module für die Mess-, Steuerungs- und Automatisierungstechnik entwickelt. Die Elektronik ist komplett mit Epoxydharz umschlossen, so dass der gehäuselose Einsatz in rauer Umgebung möglich ist.

Die verschiedenen CST-Module decken einen breiten Anwendungsbereich ab. Erhältlich sind derzeit die folgenden I/O-Module (Tabelle 1).

CST-DI8-TTL	8 digitale Eingänge, TTL-Pegel
CST-DO8-TTL	8 digitale Ausgänge, TTL-Pegel
CST-DI8-24V	8 digitale Eingänge, 24 V
CST-DO8H-24V/500mA	8 digitale Ausgänge, 24 V/500 mA
CST-AI8-8-0/10V	8 analoge Eingänge, 0-10 V, 8 Bit Auflösung
CST-AI8-8-0/10V-RO	8 analoge Eingänge, 0-10 V, 8 Bit Auflösung
CST-AI4-12-0/10V	4 analoge Eingänge, 0-10 V, 12 Bit Auflösung, galvanische Trennung
CST-AI4-12-0/25mA	4 analoge Eingänge, 0-25m A, 12 Bit Auflösung, galvanische Trennung
CST-AO2-12-0/10V	2 analoge Ausgänge, 0-10 V, 12 Bit Auflösung, galvanische Trennung
CST-AO2-12-0/25mA	2 analoge Ausgänge, 0-25 mA, 12 Bit Auflösung, galvanische Trennung
CST-II2-TTL	2 inkrementelle Encoder, 24 Bit Auflösung, galvanische Trennung
CST-II1ED-TTL	1 inkrementeller Encoder, integrierte Fehlererkennung, 24 Bit Auflösung, galvanische Trennung
CST-MC1100	Digitaler Motorcontroller für synchrone, asynchrone and Schrittmotoren, galvanische Trennung

Tabelle 1 CST-Module der Fa. EMS Dr. Thomas Wünsche

Mit den ersten beiden CST-Modulen wollen wir uns in der Folge befassen. Abbildung 8 zeigt die beiden CST-Module zur digitalen Ein-/Ausgabe mit TTL-Pegeln.

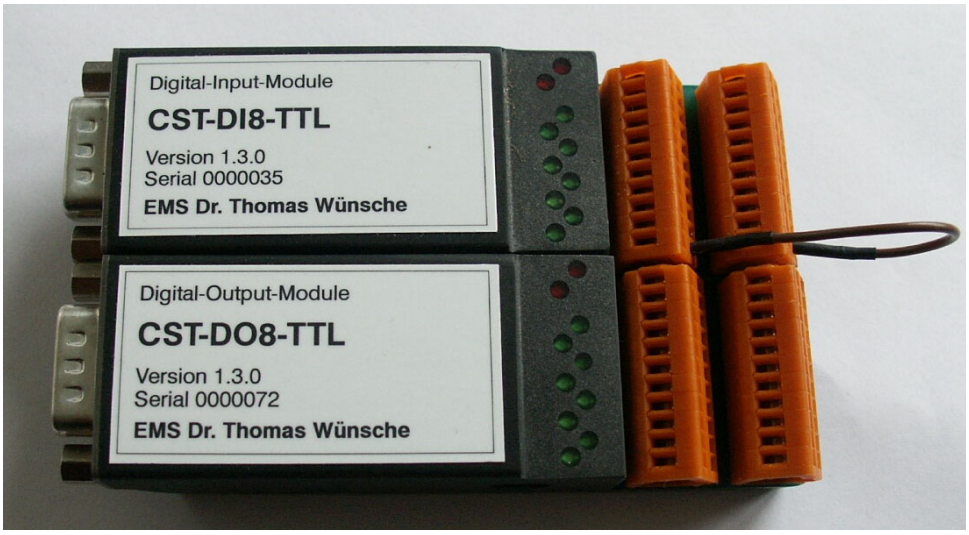


Abbildung 8 CST-Module für digitale Ein-/Ausgabe

In Abbildung 8 sind an der linken Seite die Steckverbinder (DSUB9) zum CAN-Bus zu finden. An der rechten Seite befinden sich zwei 10-polige Klemmleisten für die Verdrahtung. LEDs signalisieren die Schaltzustände der digitalen I/O und das Anliegen der Versorgungsspannung.

Auf die CST-Module wird über Schreib-, Lese- und Eventvariablen zugegriffen. Die Identifier für den Variablenzugriff werden während der Konfiguration festgelegt.

Die CST-Module haben einen Control-/Statusbereich, der unter der Variablen-Nummer 0 adressiert wird. Der Controlbereich wird durch Schreibvorgänge angesprochen und besteht aus mehreren 16-Bit-Registern, die durch eine 5 Byte lange CAN-Mitteilung gesetzt werden. Die Auswahl des Registers erfolgt über das erste Byte der Nachricht, die nächsten zwei Bytes enthalten den zu schreibenden Wert, die darauffolgenden zwei Bytes enthalten eine Maske, die einzelne Bits zur Veränderung freigibt. Nur in der Maske aktive Bits (logisch 1) werden im Controlregister beeinflusst.

Das Controlregister 0 enthält Flags, die das Verhalten des CST-Moduls am CAN-Bus beeinflussen. Folgende Bits sind (Modul-Version x.3.x) definiert (Tabelle 2).

Bit-Nr.	*Funktion (Bit gesetzt)*	*Erläuterung*
0	Auto-Bus-On	Das Modul reaktiviert seinen CAN-Controller, nachdem er aufgrund von Fehlern in den Bus-Off-Zustand übergegangen ist.
1	Auto-Baud	Das Modul folgt der Datenrate, wenn diese im Betrieb verändert wurde. Das Statusregister liefert beim Lesen eine 4 Byte lange Nachricht. Die ersten zwei Bytes enthalten den Bus-Status, die folgenden zwei Bytes den Modulstatus. Bei einzelnen Modulen können Veränderungen im Statusregister Events auslösen. Der Bus-Status gibt gegenwärtig ein Abbild des Controlregisters 0 zurück.

Tabelle 2 Flags im Controlregister 0

4.1.1. Kommunikation mit den CST-Modulen

Zur Unterscheidung zwischen Konfiguration und Normalbetrieb nehmen die CST-Module zwei Zustände ein, einen Konfigurations- und einen Operationsmode. Der Übergang zwischen den Betriebszuständen erfolgt durch Kommandos zur Modeumschaltung.

Unterschieden wird zwischen globaler oder selektiver Modeumschaltung. Bei selektiver Modeumschaltung wird das zu konfigurierende CST-Modul adressiert. Die globale Modeumschaltung in den Konfigurationsmode kann nur dann eingesetzt werden, wenn ein einziges CST-Modul am CAN-Bus angeschlossen ist. Die Umschaltung zurück in den OperationsMode kann auch bei mehreren CST-Modulen am Bus gleichzeitig durch die globale Modeumschaltung erfolgen.

Bei mehreren CST-Modulen am CAN-Bus ist deshalb von folgender Initialisierung auszugehen (Abbildung 9).

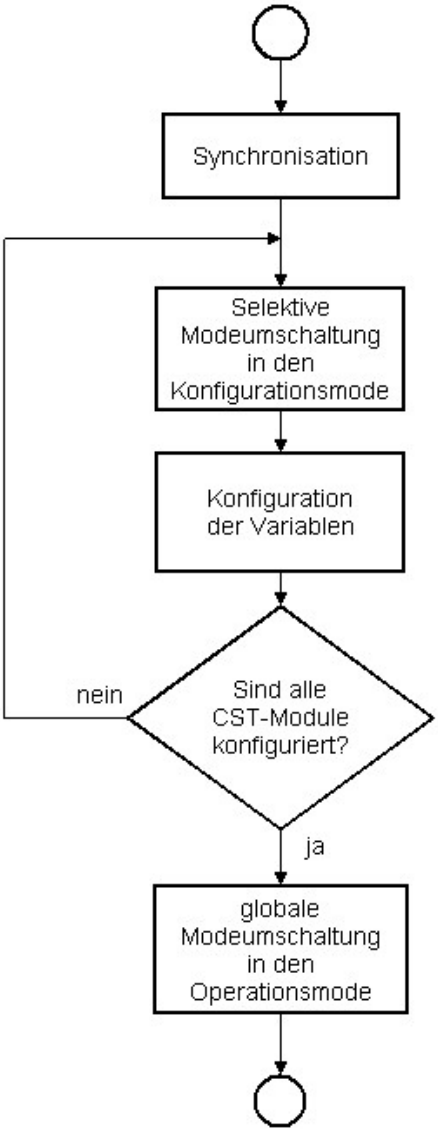


Abbildung 9 Initialisierung CST-Module

Im Konfigurationsmode werden Identifier (Communication Object Identifier - COB-ID) zum Lesen und Schreiben sowie für die Anzeige von Events für die einzelne Variablen festgelegt.

Die Variablen repräsentieren den Zustand der Ein-/Ausgänge. Bild 2 zeigt eine CAN-Mitteilung zur Zuweisung von COB-IDs.

COB-ID	*DL*	*CS*	*TY*	*VN*	*ID-Lo*	*ID-Hi*
0x7E5	5	80	0|1|2	0...9	Xx	xx

Tabelle 3 Zuweisung von COB-Identifiern

Der Identifier 0x7E5 ist reserviert für die Konfiguration der CST-Module. DL bezeichnet die Länge der CAN-Mitteilung, CS das Kommando (Command Specifier), TY den Zugriffstype (0 = Lesen, 1 = Schreiben, 2 = Event), VN die Nummer der Variablen und ID den zuzuordnenden Identifier.

Im Operationsmode erfolgen die Zugriffe über die im Konfigurationsmode zugewiesenen Identifier. Tabelle 4 zeigt das Schreiben des Bytes 0xAA mit einer CAN-Mitteilung, die den Identifier xx aufweist, während über eine CAN-Mitteilung mit dem Identifier yy die zugeordnete Variable über einen Remote-Frame abgefragt wird.

	COB-ID	*DL*	*Data*
Schreiben	xx	1	0xAA

	COB-ID	*DL*
Lesen	yy	5

Tabelle 4 Schreiben und Lesen von Variablen

Wie die Konfiguration und der Betrieb der CST-Module im einzelnen erfolgt, zeigen die folgenden Abschnitte. Eine detaillierte Beschreibung der Konfiguration ist den Unterlagen des Herstellers zu entnehmen [6].

4.1.2. Digitale Eingabe CST-DI8-TTL

Das Modul CST-DI8-TTL hat den Produktnamen CST00009 und weist die in Tabelle 5 angegebene Variablenzuordnung auf. Für unsere Experimente hier wollen wir uns auf die Variablen-Nr. 0 und 1 beschränken.

Variable	*Nr.*	*Länge*	*Zugriff*	*Bedeutung*
Control	0	5	Schreiben	Schreiben des Steuerwortes
Status	0	4	Lesen	Auslesen des Status
All_Chan	1	1	Lesen/Event	Lesen aller Kanäle – bei Event Änderung mindestens bei einem Kanal
Chan_0	2	1	Lesen/Event	Lesen/Änderung Kanal 0
Chan_1	3	1	Lesen/Event	Lesen/Änderung Kanal 1
Chan_2	4	1	Lesen/Event	Lesen/Änderung Kanal 2
Chan_3	5	1	Lesen/Event	Lesen/Änderung Kanal 3
Chan_4	6	1	Lesen/Event	Lesen/Änderung Kanal 4
Chan_5	7	1	Lesen/Event	Lesen/Änderung Kanal 5
Chan_6	8	1	Lesen/Event	Lesen/Änderung Kanal 6
Chan_7	9	1	Lesen/Event	Lesen/Änderung Kanal 7

Tabelle 5 Variablenverzeichnis CST00009

Zur Kommunikation mit dem CST-Modul dient der bereits erwähnte USB-CAN-Adapter der Fa. PEAK-System Technik. Abbildung 10 zeigt im Transmit-Fenster die für die Konfiguration benötigten und im Receive-Fenster die vom CST-Modul versendeten CAN-Mitteilungen. Durch Doppelklick auf eine Meldung im Transmit-Fenster kann diese verschickt (manuell getriggert) werden.

Tabelle 6 zeigt diese CAN-Mitteilungen in kommentierter Form. *Die Antworten vom CST-Modul sind kursiv gesetzt.*

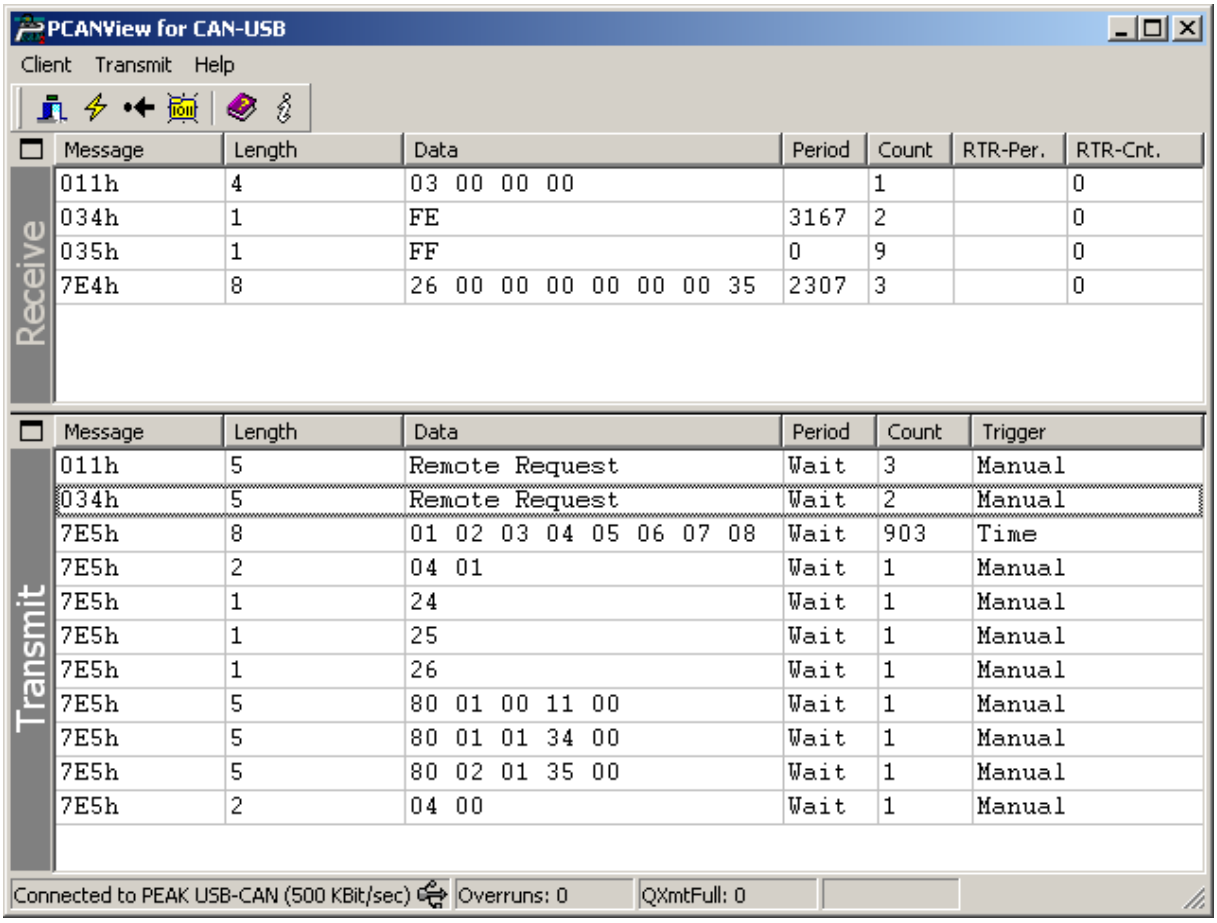


Abbildung 10 Konfiguration und Betrieb des Moduls CST-DI8-TTL

Aktion	*CAN-Message*	*Bemerkung*
Synchronisation	0x7E5 0x08 0x01 0x02 ... 0x07 0x08	mehrfach im Abstand von 10 ms
Konfigurationsmode	0x7E5 0x02 0x04 0x01	
Abfrage Hersteller	0x7E5 0x01 0x24 *0x7E4 0x08 0x24 0x45 0x4D 0x53 0x5F 0x54 0x5F 0x57*	EMS_T_W
Abfrage Produktname	0x7E5 0x01 0x25 *0x7E4 0x08 0x25 0x43 0x53 0x54 0x30 0x30 0x30 0x39*	CST0009
Abfrage Seriennummer	0x7E5 0x01 0x26 *0x7E4 0x08 0x26 0x00 0x00 0x00 0x00 0x00 0x00 0x35*	0000035
Konfiguration Control	0x7E5 0x05 0x80 0x00 0x00 0x10 0x00	COB 10: Konfiguration schreiben
Konfiguration Status	0x7E5 0x05 0x80 0x01 0x00 0x11 0x00	COB 11: Status lesen
Konfiguration Var1	0x7E5 0x05 0x80 0x01 0x01 0x34 0x00	COB 34: Var1 lesen
	0x7E5 0x05 0x80 0x02 0x01 0x35 0x00	COB 35: Var1 Event
Operationsmode	0x7E5 0x02 0x04 0x00	
Abfrage Modulstatus	0x11 0x05 *0x11 0x04 0x03 0x00 0x00 0x00*	RTR setzen
Alle I/Os abfragen	0x34 0x05 *0x34 0x01 0xFF*	RTR setzen
Bei I/O Event	*0x35 0xFE*	Bit0 gegen GND

Tabelle 6 Konfiguration und Betrieb des Moduls CST-DI8-TTL

Nach dem Zuschalten der Betriebsspannung ist das CST-Modul auf die durch den USB-CAN-Adapter vorgegebene Baudrate (von 500 kBit/s) zu synchronisieren. Durch mehrfaches Versenden der Synchronisationsmitteilung kann sich das CST-Modul mit Hilfe seiner AutoBaud-Funktion einstellen.

Da hier nur das eine CST-Modul am CAN-Bus angeschaltet ist, kann das Umschalten in den Konfigurationsmode vorerst global erfolgen.

Die nun folgende Parameterabfrage (Hersteller, Produktname, Seriennummer) wird durch das CST-Modul mit den Tabelle 6 zu entnehmenden Antworten bedient. Genau diese Parameter sind auch für eine selektive Modeumschaltung zu verwenden.

Schließlich werden noch drei COB-IDs konfiguriert. Über die COB-ID 0x11 kann das Statusbyte (Variable Nr. 0) gelesen werden. Über die COB-ID 0x34 können alle acht Eingänge gelesen werden und COB-ID 0x35 wird einem I/O-Event zugeordnet. Sobald sich einer der Eingänge ändert, versendet das CST-Modul eine CAN-Mitteilung mit dem COB-ID 0x35.

In Abbildung 10 war zum Zeitpunkt der manuellen Abfrage (COB-ID 0x34) der dem I/O-Bit 0 zugeordnete Eingang mit GND verbunden und die vom CST-Modul zurückgemeldete Eingangsbelegung musste folglich 0xFE lauten. Nach dem Lösen dieser Verbindung (I/O Event) versendete das Modul eine CAN-Mitteilung mit dem COB-ID 0x35 und einer Eingangsbelegung von 0xFF.

Der Vollständigkeit halber ist in Abbildung 11 noch das selektive Umschalten in den Konfigurationsmode dokumentiert (CAN-Mitteilungen im Rahmen). Bedingung für das selektive Umschalten ist die Kenntnis der Parameter (Hersteller, Produktname, Seriennummer) der im Netzwerk zu konfigurierenden Module.

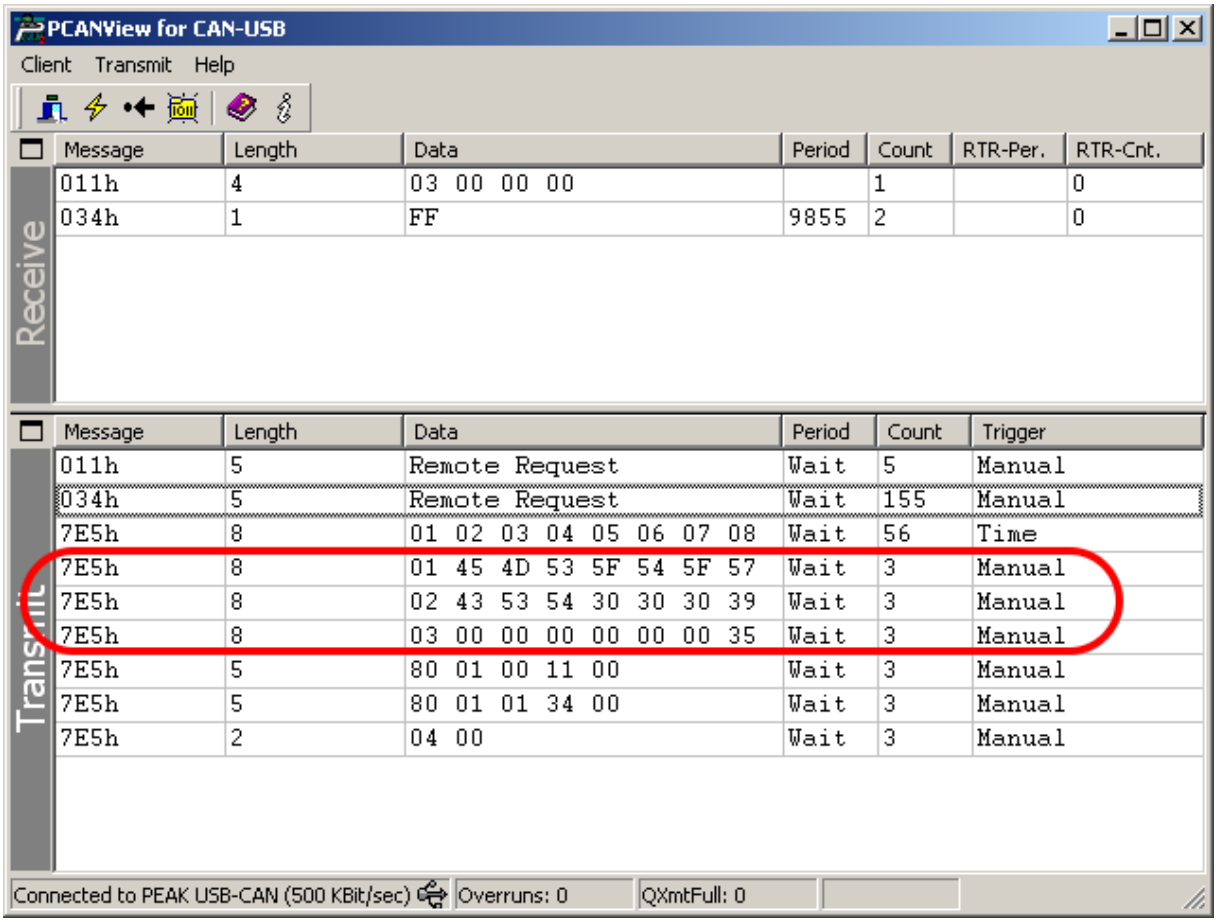


Abbildung 11 Selektives Umschalten in den Konfigurationsmode

4.1.3. Digitale Ausgabe CST-DO8-TTL

Das Modul CST-DO8-TTL hat den Produktnamen CST00010 und weist die in Tabelle 7 angegebene Variablenzuordnung auf. Für unsere Experimente hier wollen wir uns wieder auf die Variablen Nr. 0 und 1 beschränken.

Variable	*Nr.*	*Länge*	*Zugriff*	*Bedeutung*
Control	0	5	Schreiben	Schreiben des Steuerwortes
Status	0	4	Lesen/Event	Auslesen/Änderung des Status
All_Chan	1	1	Schreiben/Event	Ausgeben/Rücklesen aller Kanäle
Chan_0	2	1	Schreiben/Event	Ausgeben/Rücklesen Kanal 0
Chan_1	3	1	Schreiben/Event	Ausgeben/Rücklesen Kanal 1
Chan_2	4	1	Schreiben/Event	Ausgeben/Rücklesen Kanal 2
Chan_3	5	1	Schreiben/Event	Ausgeben/Rücklesen Kanal 3
Chan_4	6	1	Schreiben/Event	Ausgeben/Rücklesen Kanal 4
Chan_5	7	1	Schreiben/Event	Ausgeben/Rücklesen Kanal 5
Chan_6	8	1	Schreiben/Event	Ausgeben/Rücklesen Kanal 6
Chan_7	9	1	Schreiben/Event	Ausgeben/Rücklesen Kanal 7

Tabelle 7 Variablenverzeichnis CST00010

Tabelle 8 zeigt diese CAN-Mitteilungen für Konfiguration und Betrieb in kommentierter Form. *Die Antworten vom CST-Modul sind kursiv gesetzt.*

Aktion	*CAN-Message*	*Bemerkung*
Synchronisation	0x7E5 0x08 0x01 0x02 ... 0x07 0x08	mehrfach im Abstand von 10 ms
Konfigurationsmode	0x7E5 0x02 0x04 0x01	
Abfrage Hersteller	0x7E5 0x01 0x24	
	0x7E4 0x08 0x24 0x45 0x4D 0x53 0x5F 0x54 0x5F 0x57	EMS_T_W
Abfrage Produktname	0x7E5 0x01 0x25	
	0x7E4 0x08 0x25 0x43 0x53 0x54 0x30 0x30 0x31 0x30	CST0010
Abfrage Seriennummer	0x7E5 0x01 0x26	
	0x7E4 0x08 0x26 0x00 0x00 0x00 0x00 0x00 0x00 0x72	0000072
Konfiguration Control	0x7E5 0x05 0x80 0x00 0x00 0x10 0x00	COB 10: Konfiguration schreiben
Konfiguration Status	0x7E5 0x05 0x80 0x01 0x00 0x11 0x00	COB 11: Status lesen
Konfiguration Var1	0x7E5 0x05 0x80 0x00 0x01 0x33 0x00	COB 33: Var1 schreiben
	0x7E5 0x05 0x80 0x01 0x01 0x34 0x00	COB 34: Var1 lesen
Operationsmode	0x7E5 0x02 0x04 0x00	
Abfrage Modulstatus	0x11 0x05	RTR setzen
	0x11 0x04 0x03 0x00 0x00 0x00	
Alle I/Os setzen	0x33 0x01 0xFF	
	0x34 0x05	RTR setzen
	0x34 0x01 0xFF	
Alle I/Os löschen	0x33 0x01 0x00	
	0x34 0x05	RTR setzen
	0x34 0x01 0x00	

Tabelle 8 Konfiguration und Betrieb des Moduls CST-DO8-TTL

Nach dem Zuschalten der Betriebsspannung ist das CST-Modul wieder auf die durch den USB-CAN-Adapter vorgegebene Baudrate (von 500 kBit/s) zu synchronisieren.

Neben der Initialisierung werden noch drei COB-IDs konfiguriert. Über die COB-ID 0x11 kann das Statusbyte (Variable Nr. 0) und über COB-ID 0x34 Variable Var1 gelesen werden. Über die COB-ID 0x33 können alle acht Eingänge geschrieben werden.

In Abbildung 12 wurden nach der Initialisierung alle Ausgänge gesetzt (COB-ID 0x33) und anschliessend durch einen Remote-Zugriff (COB-ID 0x34) die Variable zurückgelesen.

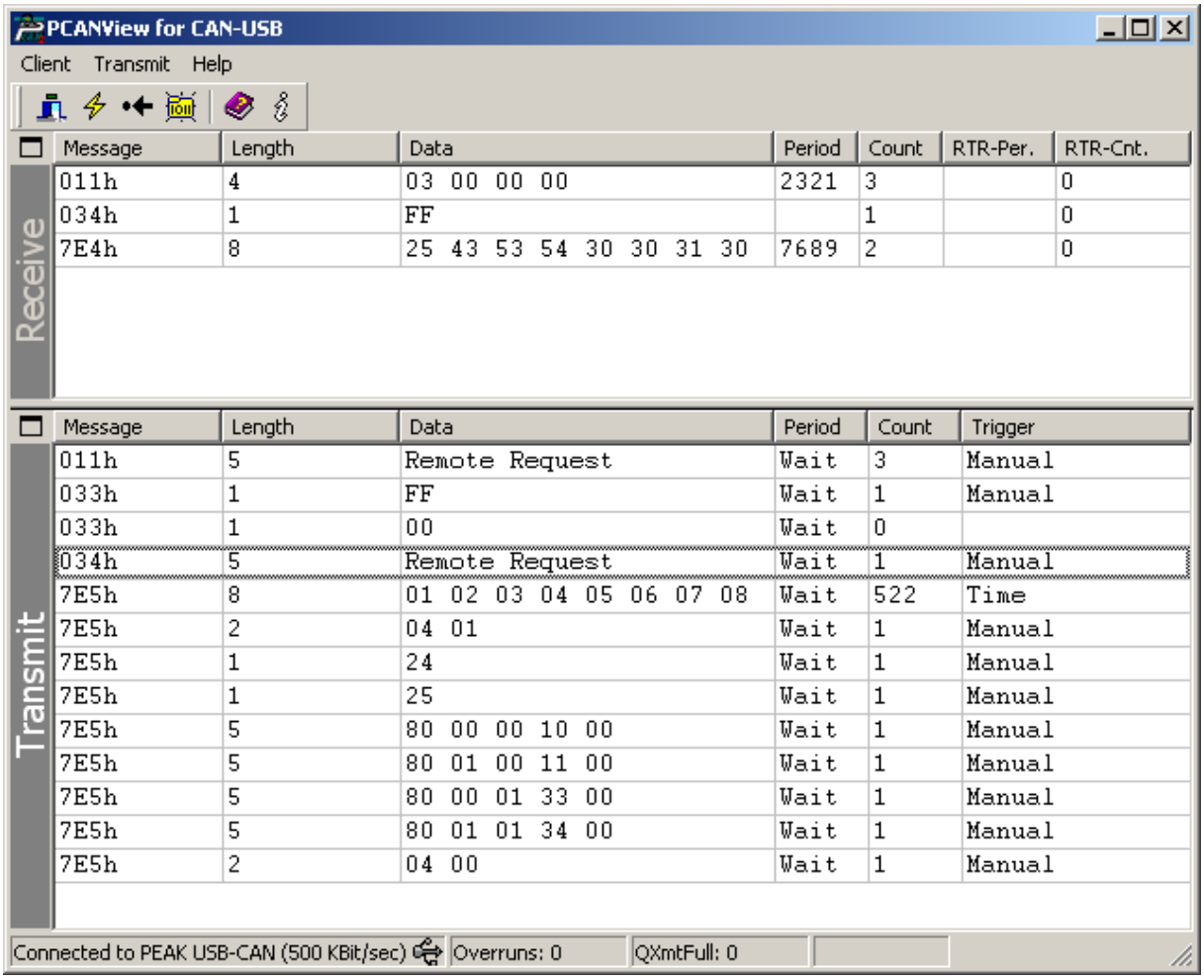


Abbildung 12 Konfiguration und Betrieb des Moduls CST-DO8-TTL

4.2. CANDIP/M162

4.2.1. Hardware & WiCAN Library

CANDIP/M162 ist ein auf dem ATmega162 von ATMEL aufbauendes Mikrocontrollermodul im Standard-28-Pin-DIP-Format. Die am Sockel des CANDIP/M162 verfügbaren Anschlüsse können aus dem in Abbildung 13 gezeigten Blockschema entnommen werden

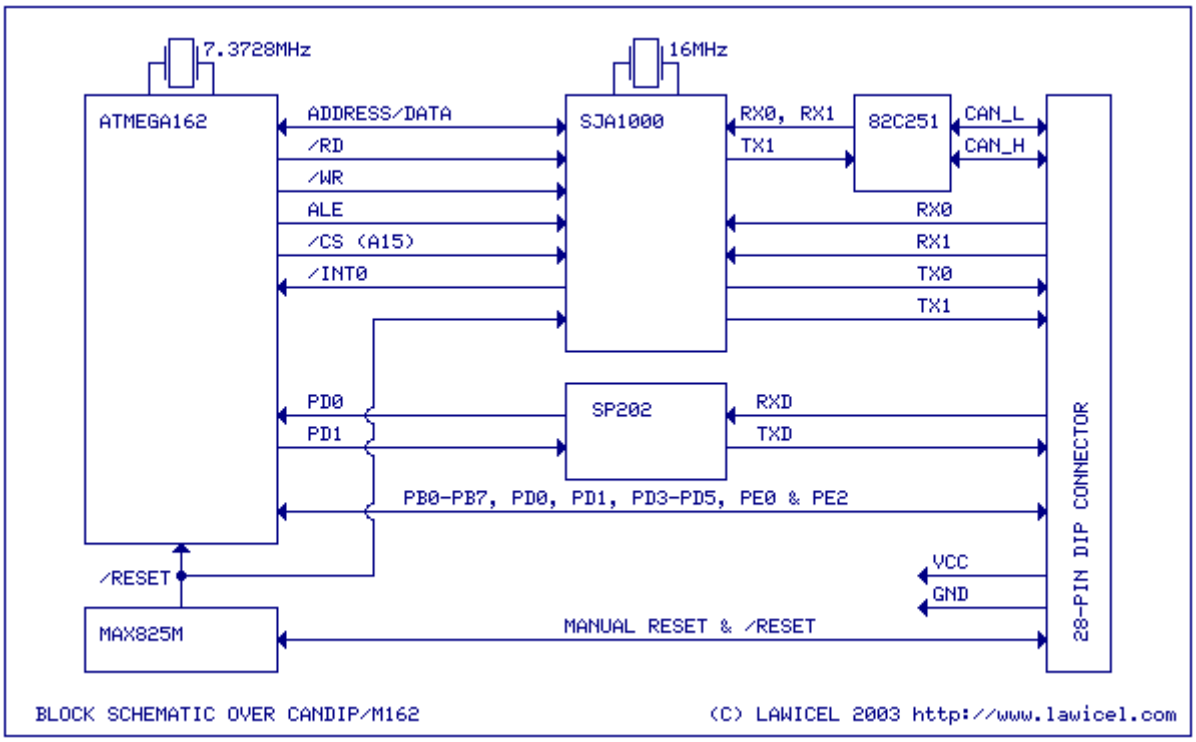


Abbildung 13 CANDIP/M162 Blockschema

Abbildung 14 zeigt die kompakte Bauweise des CANDIP/M162 und Abbildung 15 das zugehörige Evaluationsboard.

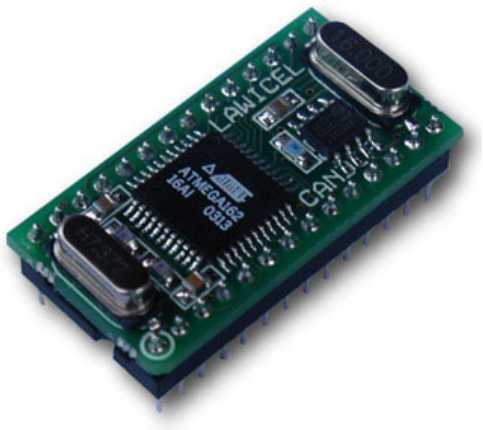


Abbildung 14 CANDIP/M162

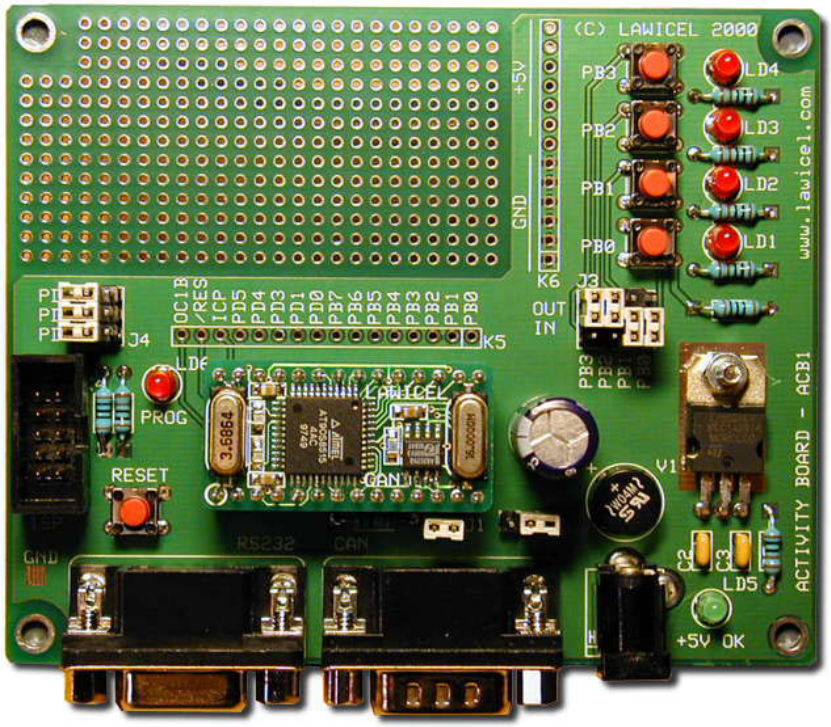


Abbildung 15 Acitivity Board ACB1

Das CANDIP/M162 kommt nicht mit einer vorprogrammierten Firmware oder einem Interpreter daher, sondern muss frei programmiert werden.

LAWICEL bietet hierzu derzeit die Library WiCAN Layer 2 CANlib für den C-Compiler ICCAVR von ImageCraft an. Von der ImageCraft-Website kann eine 45-Tage-Version des C-Compilers ICCAVR heruntergeladen werden [www.imagecraft.com/software/demos.html]. Versionen der Library für CodeVisionAVR von HPInfotech und WinAVR (AVR GCC) sind in Vorbereitung.

Die WiCAN Library unterstützt sowohl die (Standard) 11-Bit-Identifier als auch die (Extended) 29-Bit-Identifier. Die WiCAN Library ist so konzipiert, dass die Ressourcen des Mikrocontrollers geschont werden. Gegenwärtig verbraucht sie weniger als 2 KByte von den 16 KByte Flash, die beim ATmega162 zur Verfügung stehen.

Es gibt gegenwärtig drei Versionen dieser Library. Die freie, hier verwendete Version, ist auf eine CAN-Baudrate von 20 kBits/s und FIFO-Queues für vier CAN-Mitteilungen festgelegt. Die Version CLIB/1 kommt in Objektform und kann nicht verändert werden. Es gibt keine Einschränkungen bei der CAN-Baudrate mehr und die FIFO-Queues sind auf acht CAN-Mitteilungen erweitert. Die Version CLIB/2 stellt den komplett kommentierten Sourcecode als Site Licence zur Verfügung.

4.2.2. Anwendungsbeispiel

Das folgende Programmbeispiel soll die Verwendung der Library WiCAN Layer 2 CANlib für den C-Compiler ICCAVR verdeutlichen. Das CANDIP/M61 Mikrocontrollermodul wird auf das zum Starterkit von LAWICEL gehörende Activityboard ACB1 aufgesteckt.

Eine CAN-Mitteilung mit dem Identifier 0x100 soll das als Ausgang konfigurierte Pin PB1 umschalten. Pin PB0 blinkt durch einen Timerinterrupt angesteuert und dient als Lebenszeichen.

ICCAVR bietet einen Application Builder, der die Initialisierung der verwendeten Peripherie sehr einfach macht. Nachdem im Application Builder die Vorgaben für CPU, Taktfrequenz,

Port- und Timerkonfiguration gemacht wurden, stehen bereits diese Quelltextteile zum Einfügen in das von LAWICEL bereitgestellte Demoprogramm WiCANDemo_M162.C zur Verfügung. Bleiben nun noch die anwendungsspezifischen Modifikationen, die in Listing 1 fett hervorgehoben wurden.

```
/*
**  File:       WiCANDemo_M162.C
**
**  Purpose:    Demonstrate WiCAN LIB for CANDIP/M162 on Activity Board ACB1
**
**  Chip:       ATMega162 running at 7.3728 MHz
**
**  Version:    1.0.6, 1:st of July 2003
**
**  Author:     Lars Wictorsson
**              LAWICEL / SWEDEN
**              http://www.lawicel.com   lars@lawicel.com
**
**  Modified:   Claus Kuehnel
**
**  Copyright:  The copyright to the computer program(s) herein is the
**              property of LAWICEL HB, Sweden. The program(s) may be used
**              and/or copied only with the written permission of LAWICEL HB
**              in accordance with the terms and conditions stipulated in
**              the agreement/contract under which the program(s) have been
**              supplied.
**
**  Remarks:    This program is tested with ICCAVR version 6.31A.
**
**  History:    2000-05-18  1.0.0   Created (LWI)
**              2001-05-06  1.0.1   Modified to test 29bit (LWI)
**              2001-05-14  1.0.2   Modified to test RTR (LWI)
**              2002-01-23  1.0.3   Re-Compiled with latest ICCAVR (LWI)
**              2002-08-16  1.0.4   Re-Compiled with latest ICCAVR & WiCAN 1.0.6 (LWI)
**         2003-07-01  1.0.5.  Re-Compiled with ICCAVR 6.28, ATMega162 & WiCAN 1.0.7 (LWI)
**         2004-09-16 1.0.6  Modified an re-compiled with ICCAVR 6.31A & WiCAN 1.0.7 (CK)
*/

#include <iom162v.h>
#include <stdio.h>
#include <macros.h>
#include "WiCAN.h"

#define   SWITCH_ID     0x100              // COB-ID to change output

void port_init(void)
{
 PORTA = 0x00;
 DDRA  = 0x00;
 PORTB = 0x00;
 DDRB  = 0x03;                            // PB0 and PB1 output
 PORTC = 0x00;
 DDRC  = 0x00;
 PORTD = 0x00;
 DDRD  = 0x00;
 PORTE = 0x00;
 DDRE  = 0x00;
}

//TIMER1 initialize - prescale:256
```

```
// WGM: 0) Normal, TOP=0xFFFF
// desired value: 100mSec
// actual value: 99.965mSec (0.0%)
void timer1_init(void)
{
 TCCR1B= 0x00; //stop
 TCNT1H= 0xF4; //setup
 TCNT1L= 0xC1;
 OCR1AH= 0x0B;
 OCR1AL= 0x3F;
 OCR1BH= 0x0B;
 OCR1BL= 0x3F;
 TCCR1A= 0x00;
 TCCR1B= 0x04; //start Timer
}

#pragma interrupt_handler timer1_ovf_isr:16
void timer1_ovf_isr(void)
{
 //TIMER1 has overflowed
 TCNT1H= 0xE3; //reload counter high value
 TCNT1L= 0xE1; //reload counter low value
 PORTB = PORTB ^1;                          // Toggle PB0
}

//call this routine to initialize all peripherals
void init_devices(void)
{
 //stop errant interrupts until set up
 CLI();              //disable all interrupts
 port_init();
 timer1_init();

 MCUCR= 0x00;
 EMCUCR = 0x00;
// GIMSK= 0x00;
 TIMSK= 0x80;        //timer interrupt sources
 ETIMSK=0x00;
 GICR= 0x00;
 PCMSK0=0x00;
 PCMSK1=0x00;
 SEI();              //re-enable interrupts
 //all peripherals are now initialized
}

WiCAN_Object MyFrame;

void main( void)
{
    UBRR0 = 7;                              // 57600 baud for 7.3728 MHz clock
    UCSR0B  = 0x18;                         // Enable RXEN & TXEN

    init_devices();                         // initialize Timer1 and PortB
/*
**  Setup CAN controller
*/
    if ((WiCAN_Init(WiCAN_SPEED_20K, 0, 0, 0xDE)) == WiCAN_OK) {
        WiCAN_Start();

        MyFrame.id      = 0x500;            // Set up a standard 11-bit ID Frame with
        MyFrame.len     = 1;                // DLC = 1 (i.e. 1 byte)
```

```
        MyFrame.byte[0] = 0x01;           // The first byte to '0x01'
        WiCAN_SendFrame(&MyFrame);        // Send the frame

        MyFrame.id      = 0x510;          // Set new ID
        MyFrame.flags   = WiCAN_29;       // Change the frame type to Extended
        MyFrame.byte[0]++;                // Increase the value of byte 1
        WiCAN_SendFrame(&MyFrame);        // Send the frame again

        MyFrame.id      = 0x520;          // Set new ID
        MyFrame.flags   = WiCAN_RTR;      // Change the STD frame type and set RTR bit
        WiCAN_SendFrame(&MyFrame);        // Send the frame again (notice DLC is set to
zero when sending)

        while(1) {                        // Do a forever loop
            if (WiCAN_GetRxQueueSize() >= 1) {    // Wait until input FIFO has 1 message(s)
or more
                while (WiCAN_GetFrame(&MyFrame) == WiCAN_OK)
                {
                     if (MyFrame.id == SWITCH_ID)       // Filtering COB-ID
                     {
                        if (MyFrame.byte[0] == 0)
                            PORTB &= 0x0D;
                        else
                            PORTB |= 0x02;
                     }
                     WiCAN_SendFrame(&MyFrame);  // Send all of them back as they came in.
                }
            }
        }
    }
}
```

Listing 1 Programmbeispiel WiCANDemo_M162.C

Nach dem Programmieren des CANDIP/M162 kann das Programm gestartet werden. Es soll vorausgesetzt werden, dass die Kommunikation zwischen Entwicklungs-PC und CANDIP/M162 vorbereitet ist. Mit Hilfe des CAN232-Interfaces kann das gemäss Abschnitt 3.2 vorgenommen werden. Dann kann, wie in Abschnitt 3.2 schon gezeigt, durch das Versenden der CAN-Mitteilungen [100100] bzw. [1001xx] (xx bedeutet verschieden von 00) die an Pin PB1 angeschlossene LED ein- bzw. ausgeschaltet werden. Die CAN-Mitteilung wird als Echo zurückgeschickt. CAN-Mitteilungen mit anderen Identifiern werden nicht berücksichtigt und nur zurückgeschickt.

Drückt man zwischendurch den Restknopf des Activityboards ACB1, dann kann man auch die zu Beginn des Programms vom CANDIP/M162 versendeten CAN-Mitteilungen sehen. Abbildung 16 sind die letzten drei CAN-Mitteilungen durch Reset provoziert worden. Im einzelnen sind das ein Standard Frame [500101], ein Extended Frame [00000510102] und ein Remote Frame [520].

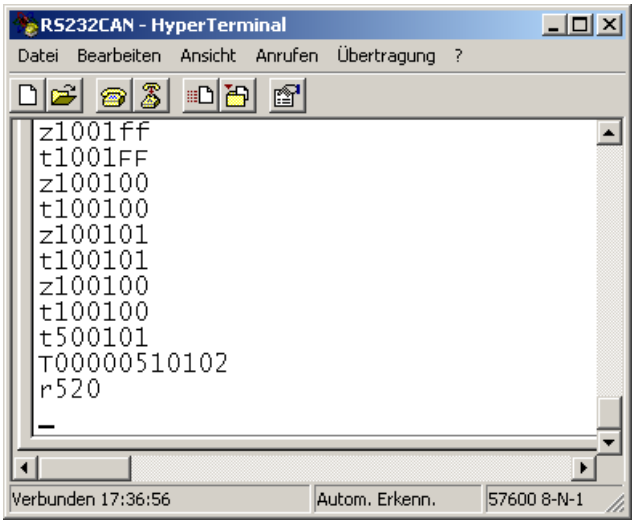


Abbildung 16 Kommunikation mit CANDIP/M162

Dem Quelltext (Listing 1) ist das Erstellen dieser CAN-Mitteilungen gut kommentiert zu entnehmen.

4.3. PCAN MicroMod

4.3.1. Hardware

Beim PCAN MicroMod handelt es sich um ein universelles Ein-/Ausgabemodul mit CAN-Interface.

Kern ist der Controller MB90F497 von Fujitsu. Mit seinem integriertem CAN-Bus Controller und den analogen und digitalen Ein-/Ausgängen ist er eine preiswerte Lösung für kleine intelligente CAN-Knoten.

Die integrierte Firmware im PCAN MicroMod erlaubt es dem Anwender die Hardware des Mikrocontrollerboards über ein als Windows-Programm vorliegendes MicroMod Konfiguration Tool zu konfigurieren. Dadurch sind keinerlei Programmierkenntnisse erforderlich.

Die Konfigurationsdaten werden über CAN an das PCAN MicroMod gesendet. Jedes Modul kann einzeln über den Bus angesprochen und parametriert werden.

Es ist selbstverständlich auch möglich, eigene Programme zu schreiben. Der von Fujitsu frei erhältliche C-Compiler Softune Workbench erlaubt das Erstellen eigener Software. Über die serielle Schnittstelle kann die Software auf das PCAN MicroMod geladen werden.

Erweiterbar mit verschiedenen Grundplatinen ist der Einsatz im Geräte- und Anlagenbau oder in der KFZ-Industrie möglich. Das optional erhältliches MicroMod Evaluation Board erleichtert den Einstieg und die Entwicklung eigener Grundplatinen.

Durch die mitgelieferte Software für Windows® stehen folgende Möglichkeiten zur Verfügung:

- Die digitalen Eingangspegel können periodisch oder auf Pegelwechsel hin gesendet werden
- Digitale Eingänge können logisch verknüpft werden
- Anpassungen von Analoggrößen über Kennlinien
- Direktes Umsetzen von analogen Eingängen auf CAN-IDs

- Direkte Unterstützung von Rotary-Encodern (Drehschalter, KFZ)

Abbildung 17 zeigt das MicroMod Evaluation Board mit aufgestecktem PCAN MicroMod. Das PCAN MicroMod bietet folgende Ein-/Ausgänge:

- 8 digitale Eingänge (TTL)
- 8 digitale Ausgänge (TTL)
- 8 analoge Eingänge 10 Bit Auflösung mit Referenzspannung 5V
- 4 PWM- oder 2 Frequenzausgänge im Bereich von 1 Hz bis 10 kHz
- CAN-Anschluss mit 82C251 Transceiver

Über eine Stiftleiste/Lötjumper kann die Modul-ID gesetzt werden.

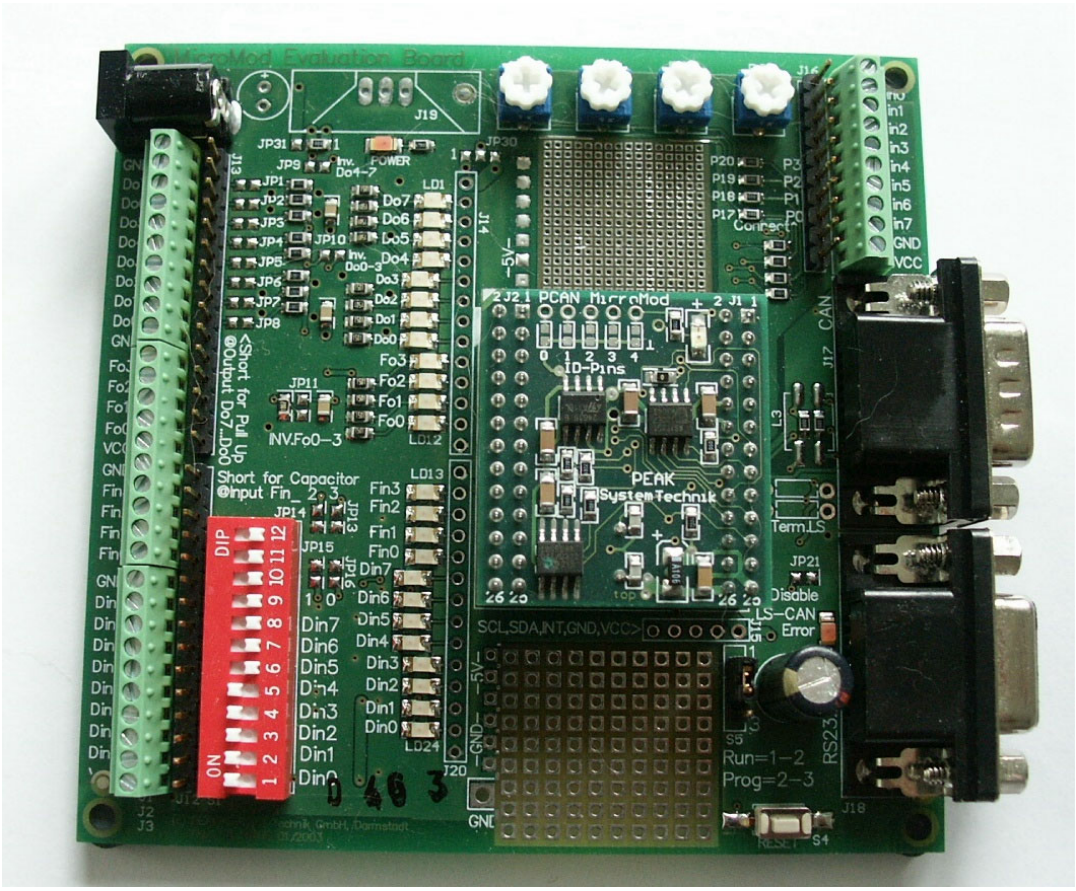


Abbildung 17 MicroMod Evaluation Board mit aufgestecktem PCAN MicroMod

4.3.2. MicroMod Configuration Tool

Das MicroMod Configuration Tool von PEAK-System Technik ist ein komfortables, an die Funktionalität des PCAN MicroMod angepasstes Werkzeug. Abbildung 18 zeigt die Oberfläche des Programms mit Fenstern für die einzelnen zu konfigurierenden Funktionen.

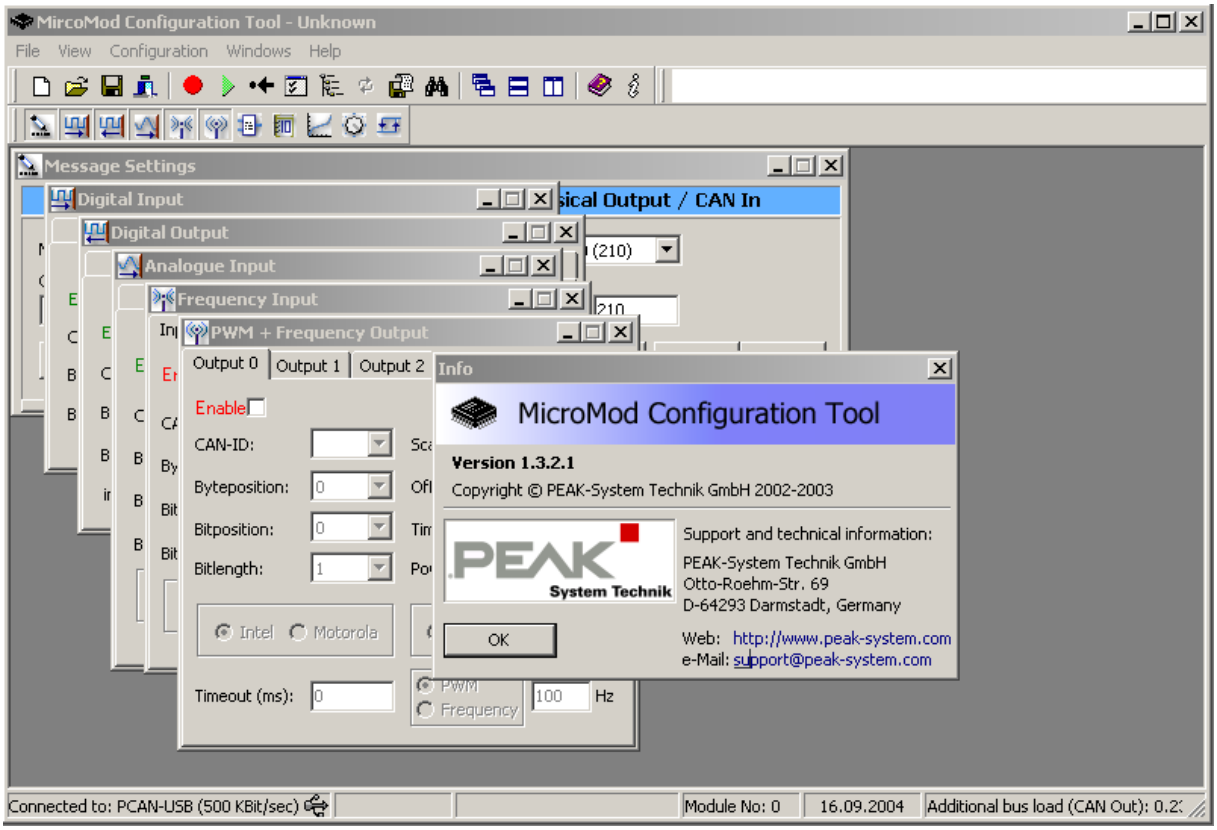


Abbildung 18 Screenshot MicroMod Configuration Tool

Das auf dem MicroMod Evaluation Board befindliche PCAN MicroMod soll folgendermaßen konfiguriert werden:

CAN-ID	*Funktion*	*DL*	*Trigger*
0x200	Lesen der Eingänge Din7-Din4	1	Zyklisch 100 ms
0x201	Lesen der Eingänge Din3-Din0	1	Bei Änderung eines der Eingänge
0x210	Schreiben der Ausgänge Do7-Do0	1	Manuell
0x220	Lesen des Eingang Ain0	2	Zyklisch 200 ms

Vom PCAN MicroMod wird alle 100 ms eine CAN-Mitteilung mit der ID 0x200 und einem Datenbyte verschickt, das im oberen Halbbyte die Werte der Eingänge Din7-Din4 aufweist.

Wird an einem der Eingänge Din3-Din0 eine beliebige Pegeländerung festgestellt, dann wird vom PCAN MicroMod eine CAN-Mitteilung mit der ID 0x201 und einem Datenbyte verschickt, das im unteren Halbbyte die Werte der Eingänge Din3-Din0 aufweist.

Empfängt das PCAN MicroMod eine CAN-Mitteilung mit der ID 0x210, dann überträgt es den Inhalt des ersten Bytes an die Ausgänge Do7-Do0.

Letztlich wird vom PCAN MicroMod alle 200 ms eine CAN-Mitteilung mit der ID 0x220 und zwei Datenbytes verschickt, die das 10-Bit Ergebnis des mit Ain0 verbundenen AD-Umsetzers repräsentieren.

Die folgenden Bilder zeigen die Schritte der Konfiguration des PCAN MicroMod auf. Abbildung 19 zeigt die Verbindungsaufnahme zum angeschlossenen PCAN MicroMod. Sind mehrere gleichartige Module am CAN-Bus angeschlossen, dann werden diese über eine Modul-ID (Modul No), die über Jumper eingestellt werden kann, selektiert.

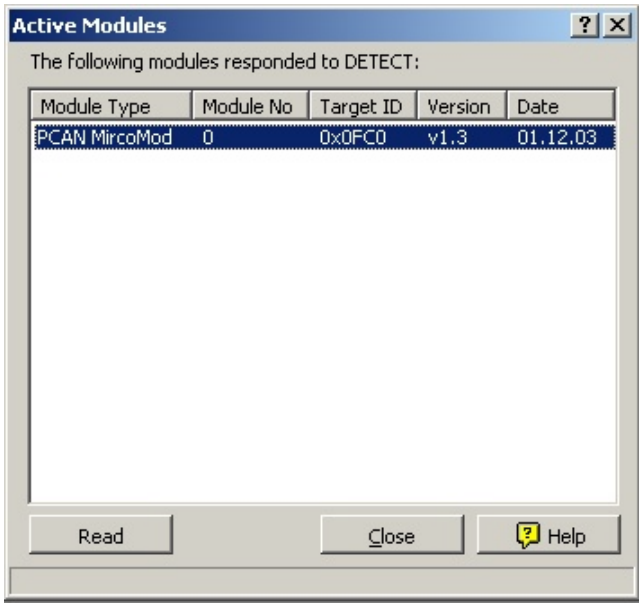


Abbildung 19 Verbindungsaufbau zu PCAN MicroMod

Wie Abbildung 20 zeigt, werden die CAN-IDs getrennt nach Ein- und Ausgabe vergeben.

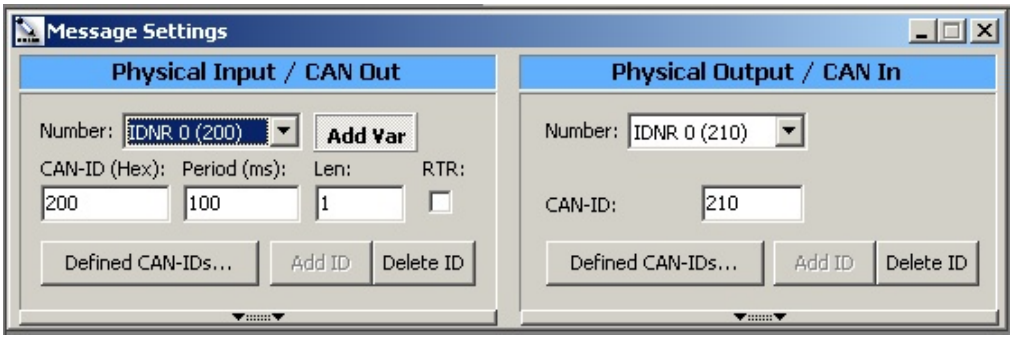


Abbildung 20 Vergabe der CAN-Identifier

Die Zuordnung der digitalen Eingänge zum betreffenden Identifier erfolgt in separaten Fenstern. Abbildung 21 zeigt die Zuordnung von Din0 zur Bit 0 von Byte 0. Als Trigger zum Versenden dieser CAN-Mitteilung wurde eine beliebige Änderung gewählt.

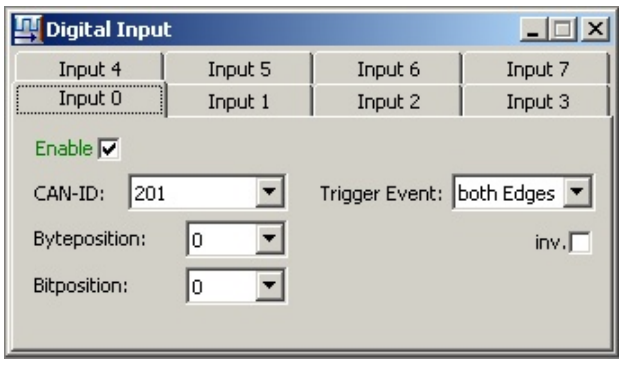


Abbildung 21 Zuordnung der digitalen Eingänge

Abbildung 22 zeigt die Zuordnung eines digitalen Ausgangs zum betreffenden CAN-Identifier.

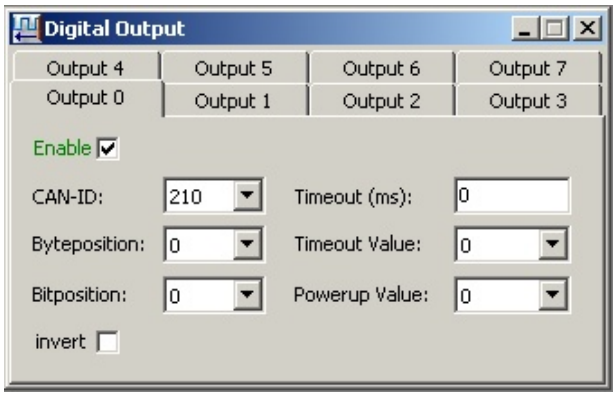


Abbildung 22 Zuordnung der digitalen Ausgänge

Die Zuordnung der analogen Eingänge kann sehr flexibel erfolgen. Abbildung 23 zeigt die Zuordnung von Eingang Ain0 zu betreffenden Identifier. Das 10-Bit umfassende Ergebnis des AD-Umsetzers wird rechtsbündig (LSB auf Bit 0 von Byte 0) und ohne Vorzeichen ausgegeben. Dem Analogeingang ist ein Filter mit einer Zeitkonstanten von 20 ms vorgeschaltet.

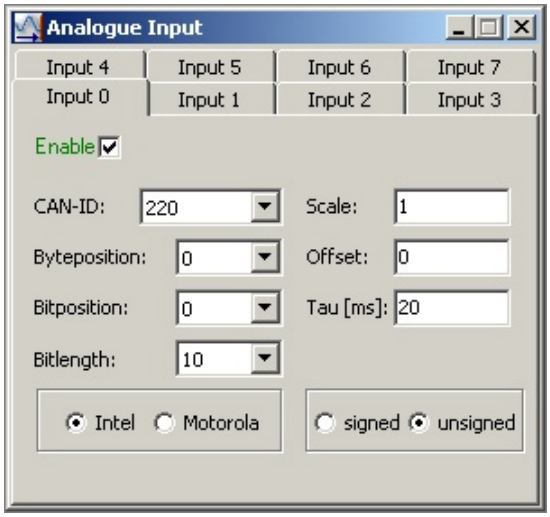
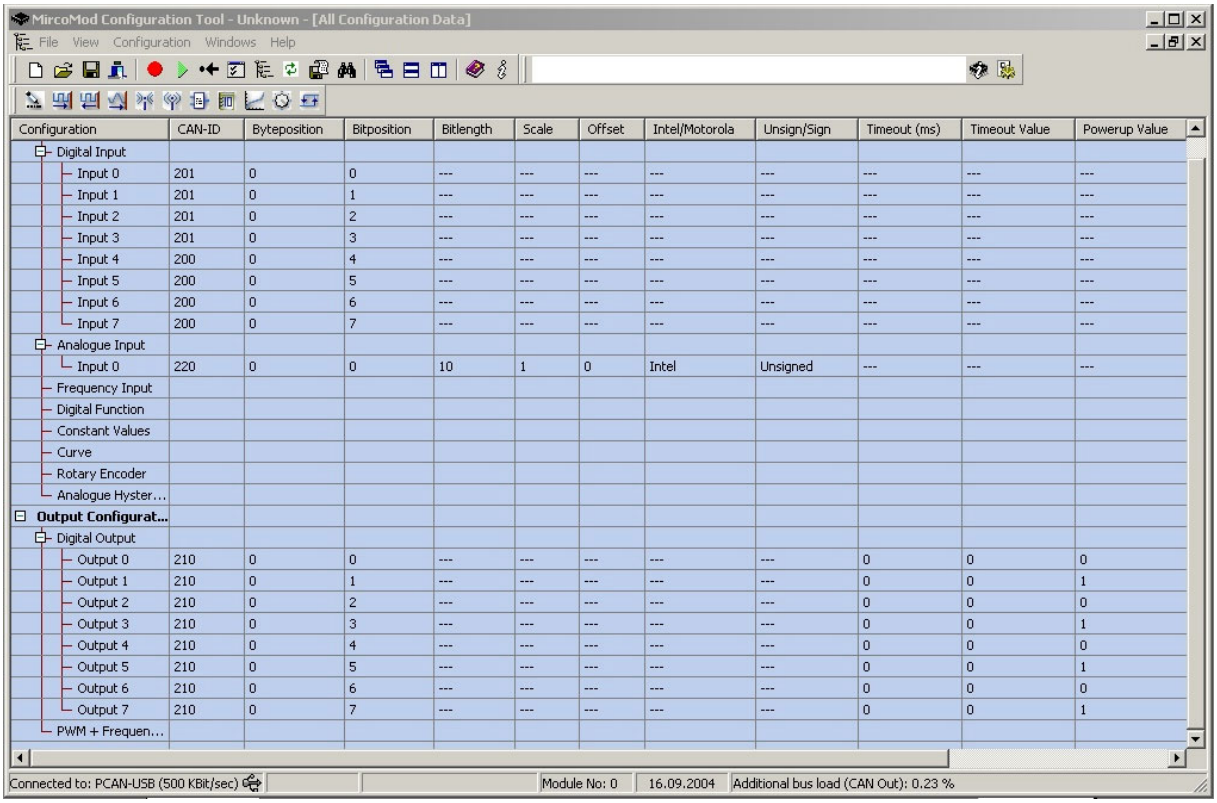


Abbildung 23 Zuordnung der analogen Eingänge

Nachdem nun alle Konfigurationsschritte absolviert sind, kann die Konfiguration im Überblick betrachtet werden. Abbildung 24 zeigt diese Zusammenstellung.



Configuration	CAN-ID	Byteposition	Bitposition	Bitlength	Scale	Offset	Intel/Motorola	Unsign/Sign	Timeout (ms)	Timeout Value	Powerup Value
Digital Input											
Input 0	201	0	0	---	---	---	---	---	---	---	---
Input 1	201	0	1	---	---	---	---	---	---	---	---
Input 2	201	0	2	---	---	---	---	---	---	---	---
Input 3	201	0	3	---	---	---	---	---	---	---	---
Input 4	200	0	4	---	---	---	---	---	---	---	---
Input 5	200	0	5	---	---	---	---	---	---	---	---
Input 6	200	0	6	---	---	---	---	---	---	---	---
Input 7	200	0	7	---	---	---	---	---	---	---	---
Analogue Input											
Input 0	220	0	0	10	1	0	Intel	Unsigned	---	---	---
Frequency Input											
Digital Function											
Constant Values											
Curve											
Rotary Encoder											
Analogue Hyster...											
Output Configurat...											
Digital Output											
Output 0	210	0	0	---	---	---	---	---	0	0	0
Output 1	210	0	1	---	---	---	---	---	0	0	1
Output 2	210	0	2	---	---	---	---	---	0	0	0
Output 3	210	0	3	---	---	---	---	---	0	0	1
Output 4	210	0	4	---	---	---	---	---	0	0	0
Output 5	210	0	5	---	---	---	---	---	0	0	1
Output 6	210	0	6	---	---	---	---	---	0	0	0
Output 7	210	0	7	---	---	---	---	---	0	0	1
PWM + Frequen...											

Abbildung 24 Überblick über die erfolgte Konfiguration

Entspricht die Konfiguration der Aufgabenstellung, dann kann diese zum PCAN MicroMod herunter geladen werden und mit dem Test der Anwendung begonnen werden.

4.3.3. PCANView

PCANView ist ein leicht zu handhabendes Monitorprogramm der Fa. PEAK-System Technik und hilft, den Datenfluss in einem CAN-Netzwerk transparent zu machen. PCANView passt zum USB-CAN-Interface und weist die folgenden Merkmale auf:

- Anzeige jeder empfangenen CAN-Mitteilung mit ID, Datenlänge und Daten in einer Liste sowie zusätzlichen Angaben
- Anzeige empfangener Remote Frames
- Eintragen zu versendender CAN-Mitteilungen in eine Liste. Das Versenden kann zyklisch in festen Intervallen, manuell (durch Doppelklick) oder als Antwort auf einen Remote Frame erfolgen.

Abbildung 25 zeigt die Verbindungsaufnahme zum angeschlossenen und vorgängig konfigurierten PCAN MicroMod und die gegebenen Selektionsmöglichkeiten (Baudrate, Messagefilter).

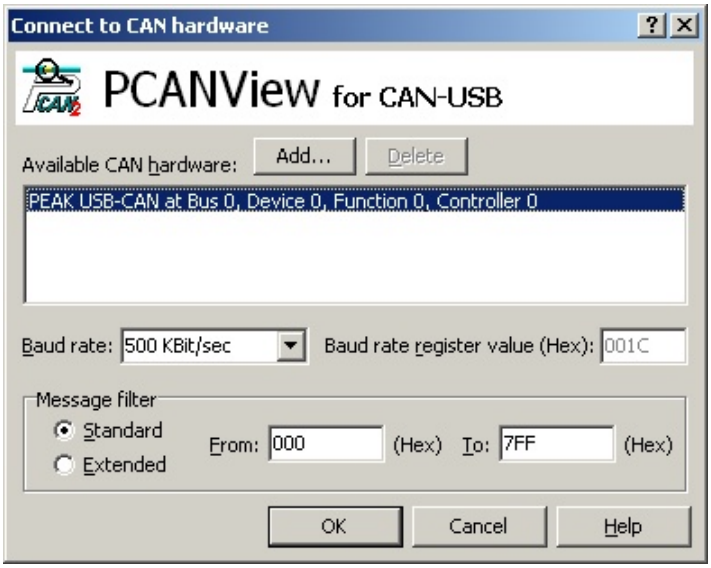


Abbildung 25 Aufnahme der Kommunikation

Ist die Kommunikation aufgenommen, dann sind im Receivefenster die vom PCAN MicroMod zyklisch versendeten CAN-Mitteilungen mit den IDs 0x200 und 0x220 zu sehen. Die in Abbildung 26 zusätzlich gelistete CAN-Mitteilung 0x201 erscheint erst nach einer Pegeländerung an einem der Pins Din3-Din0.

Bevor die CAN-Mitteilung mit dem ID 0x210 zum PCAN MicroMod geschickt werden kann, muss diese in die Transmitlliste eingetragen werden. In Abbildung 26 sind zwei bis auf den Dateninhalt identische CAN-Mitteilungen mit dem ID 0x210 eingetragen worden. Durch Doppelklick auf die eine oder die andere Zeile wird die betreffende CAN-Mitteilung (mit Datenbyte 0xAA oder 0x55) zum CAN-Bus geschickt.

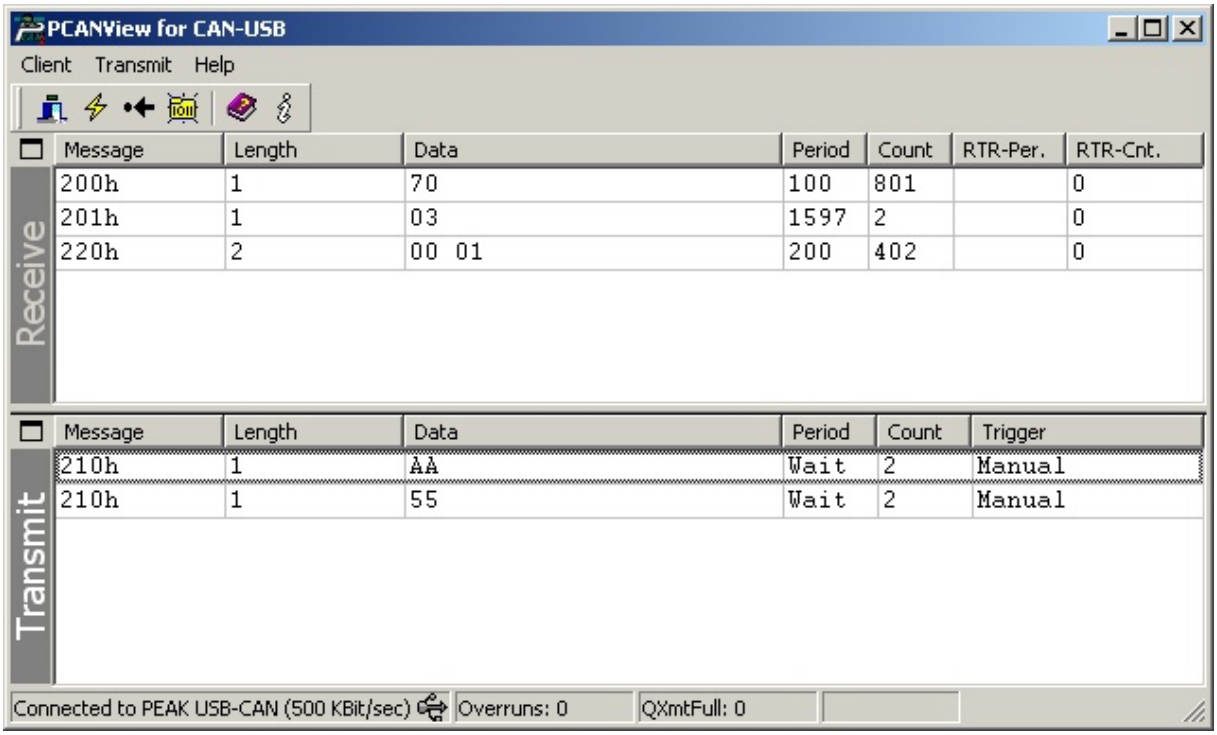


Abbildung 26 Monitoring der Verbindung zum PCAN MicroMod

4.4. STK500/501 mit AT90CAN128

4.4.1. Mikrocontroller AT90CAN128

Der AT90CAN128 ist ein Mikrocontrollers aus Atmels AVR-Familie, der neben einer umfangreichen Peripherie auch einen internen CAN-Controller aufweist. Dieser fullCAN-Controller stellt die gesamte Hardware für bequeme Akzeptanzfilterung und Messagehandling zur Verfügung.

Sogenannte Messageobjects (Mob's) enthalten alle benötigten Informationen (Identifier, Datenbyte etc.), die eine zu empfangende oder zu versendende CAN-Mitteilung beschreiben.

Während der Initialisierung der Peripherie des Mikrocontrollers definiert das Anwendungsprogramm, welche Mitteilungen gesendet bzw. empfangen werden sollen. Empfängt der CAN-Controller eine Mitteilung, deren Identifier mit einem Empfangs-Mob übereinstimmt, dann erfolgt ein Interrupt und das Anwendungsprogramm kann die empfangene CAN-Mitteilung bearbeiten. Empfangene Remote-Frames werden vom CAN-Controller direkt bearbeitet, ohne Ressourcen der CPU in Anspruch zu nehmen.

Der CAN-Controller im AT90CAN128 unterstützt CAN V2.0B Active (behandelt 11 Bit und 29 Bit Indentifier). Abbildung 27 zeigt ein Blockschema des fullCAN-Controllers im AT90CAN128.

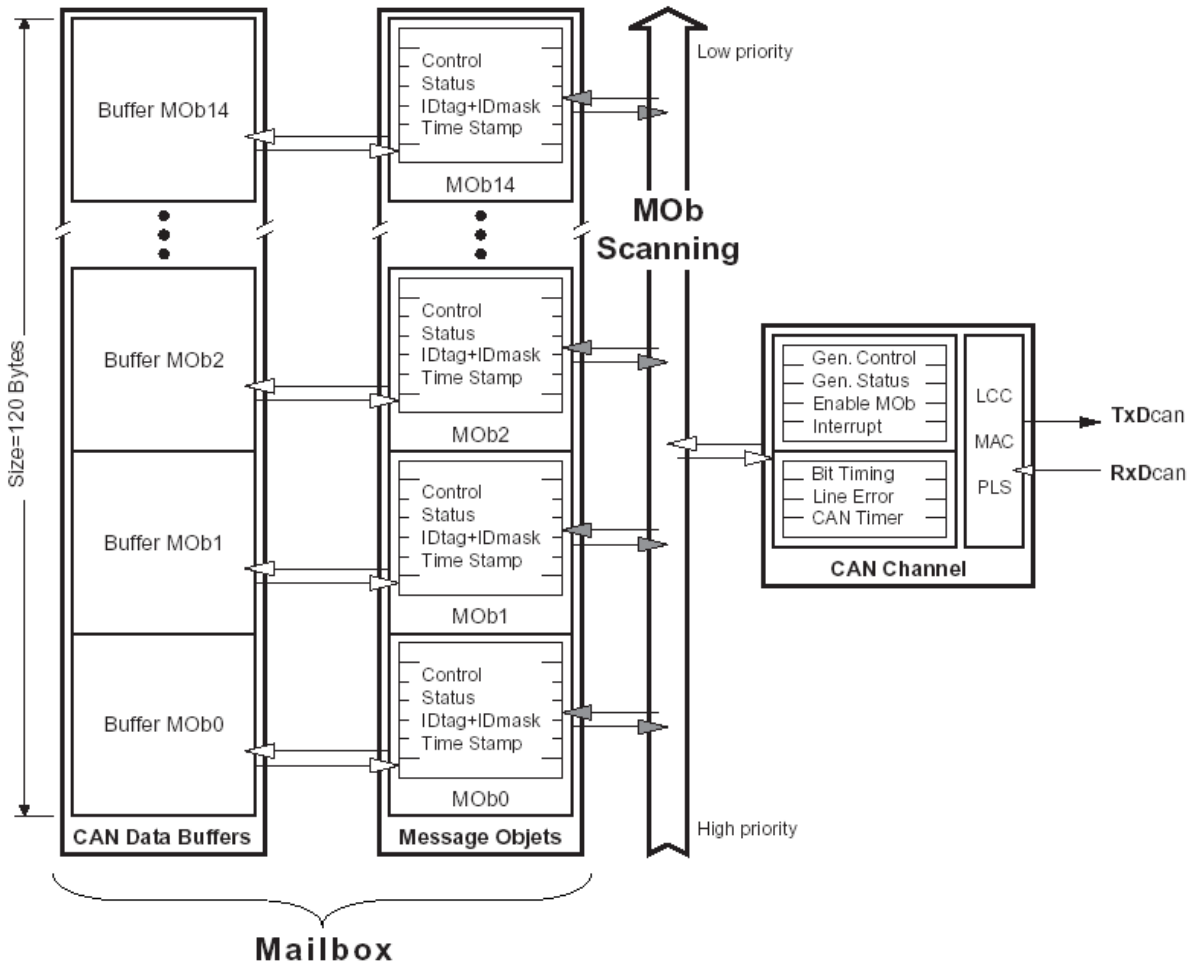


Abbildung 27 Blockschema des fullCAN-Controllers im AT90CAN128

4.4.2. Hardware & Library CAN_LIB.C, V 1.4

Der Mikrocontroller AT90CAN128 kann zu Evaluationszwecken am einfachsten im STK500/501 Entwicklungsboard von Atmel betrieben werden.

Das STK501 ist ein Aufsteckmodul für das STK500, um die Kontaktierung von Bausteinen im TQFP-Gehäuse über einen ZIF-Sockel (Zero Insertion Force) zu ermöglichen. Das ATADAPCAN01-STK501 CAN Add-On stellt noch den CAN-Treiberbaustein zur Verfügung.

Abbildung 28 zeigt die komplette Hardware, bestehend aus STK500, STK 5001 und dem ATADAPCAN01-STK501 CAN Add-On im Hintergrund.

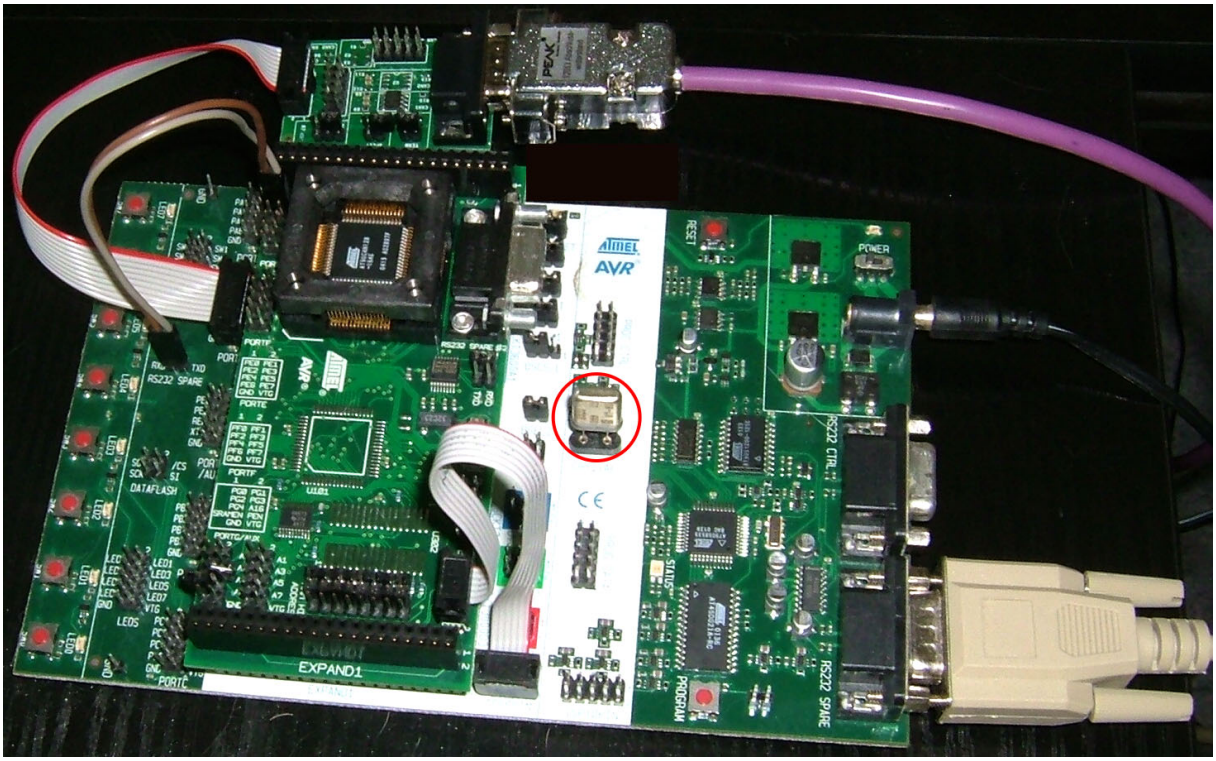


Abbildung 28 Hardware der AT90CAN128-Entwicklungsumgebung (STK500/5001 & ATADAPCAN01-STK501 CAN Add-On)

Am ATADAPCAN01-STK501 CAN Add-On wir der CAN-Bus angeschlossen und am STK500 stehen eine Terminalschnittstelle (unterer DSUB) und eine Programmierschnittstelle (oberer DSUB) zur Verfügung.

Um das kritische CAN-Bus-Timing einhalten zu können, wird hier mit einem externen Quarz (in Abbildung 28 markiert) gearbeitet. Die Fuses des AT90CAN128 sind darauf abzustimmen. Abbildung 29 zeigt die mit dem AVRStudio vorgenommenen Einstellungen für einen externen Quarz mit 8 MHz.

Für die Zugriffe auf den CAN-Controller stellt Atmel die Bibliothek CAN_LIB.C aktuell in der Version V 1.4 für die Embedded Workbench von IAR zur Verfügung. Im folgenden Anwendungsbeispiel wurde diese Bibliothek verwendet. Das Projekt befindet sich komplett auf der CD der Ergänzungslieferung.

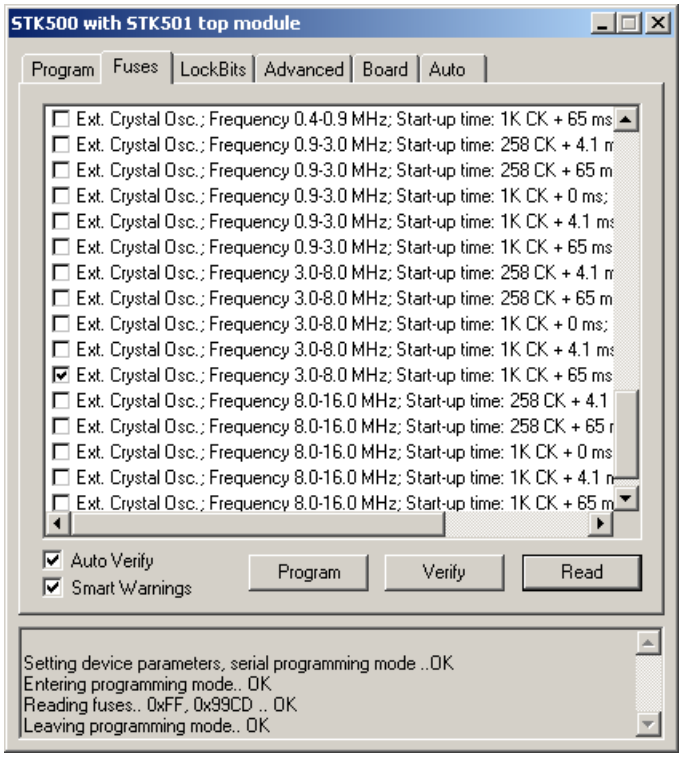


Abbildung 29 Fuses für externen Quarz 8 MHz

4.4.3. Anwendungsbeispiel

Als Anwendungsbeispiel soll ein einfaches Demoprogramm TRANSMIT-RECEIVE.C dienen, welches von Atmel zur Verfügung gestellt und hier nur leicht modifiziert wurde.

Wie es der Dateiname bereits verrät, empfängt der AT90CAN128 Mitteilungen vom CAN-Bus und stellt den Dateninhalt von Mitteilungen mit akzeptierten Identifiern am Terminalausgang zur Verfügung. Ein an die serielle Schnittstelle des AT90CAN128 angeschlossenes Terminal (z.B. Hyperterminal auf dem PC) zeigt diese Dateninhalte dann an.

Listing 2 zeigt den Quelltext des Programms TRANSMIT-RECEIVE.C. Das Projekt ist mit dem C Compiler von IAR zu übersetzen. Eine Demoversion der Embedded Workbench kann von www.iar.com heruntergeladen werden. Wichtige Stellen sind im Programm wieder fett markiert.

```
/****************************************************************************
* File Name: Transmit-Receive.c
*-----------------------------------------------------------------------------
* Copyright (c) 2002 Atmel.
*-----------------------------------------------------------------------------
* RELEASE:      Name:
* REVISION:     Revision: 1.0
*-----------------------------------------------------------------------------
* PURPOSE:
* ATmega128CAN11 Demo
****************************************************************************/

/*_____ I N C L U D E S _____________________________________________________*/
#include "config.h"
#include "can_lib.h"
#include "uart_lib.h"
#include "stdio.h"
```

```
/*_____ M A C R O S ________________________________________________*/

int putchar(int c)
{
  return uart_putchar(c);
}

/*_____ D E F I N I T I O N S ____________________________________________*/

void can_init(void);

/*_____ D E C L A R A T I O N S __________________________________________*/

Uchar f_recept_can_message=0;
Uchar compteurligne=0;
can_msg_t msgrx;
Uchar datarx [8];


/*F****************************************************************************
* NAME: main
*-------------------------------------------------------------------------------
* PARAMS:   none
* return:   none
*-------------------------------------------------------------------------------
* PURPOSE:
*-------------------------------------------------------------------------------
* EXAMPLE:
*-------------------------------------------------------------------------------
* NOTE:
*-------------------------------------------------------------------------------
* REQUIREMENTS:
*******************************************************************************/
void main (void)
{

Uchar i;
/*------- Enable AVR Init ------------*/
SREG=0x80;

/*------- Disable CLKK Prescaler ------------*/
CLKPR=0x80; CLKPR=0x00;

/*------- Init Uart ----------------*/
uart_init();

/*------- Init CAN ----------------*/
can_init();


//*************************
pt_st_can_rx=&msgrx;
pt_st_can_rx->pt_donne=&datarx[0];
printf ("Ident    DIR    Dlc    Data\r\n");

do
  {
    if  (f_recept_can_message)
      {
       f_recept_can_message=0;

       CAN_IT_DISABLE;

       if ((pt_st_can_rx->ctrl & CONF_IDE)==CONF_IDE)  //if True = extanded IDE
        {
          printf(" %x      Rx      %x     ",pt_st_can_rx->id.ext, (pt_st_can_rx->ctrl & 0x0F));
        }
```

```
       else
        {
          printf(" %x      Rx      %x     ",pt_st_can_rx->id.std, (pt_st_can_rx->ctrl & 0x0F));
        }

      for (i=0; i<(pt_st_can_rx->ctrl & 0x0F); i++)
        {
         if (datarx[i]==0)
           {
           printf("00 ");
           }
         else
           {
            printf("%x ",datarx[i]);
           }
        }

      printf(" \r\n");

     if (compteurligne<10)
      {
        compteurligne++;
      }
     else
      {
      printf ("Ident    DIR    Dlc    Data\r\n");
      compteurligne=1;
      }


      CAN_IT_ENABLE;

      }
  }
while(1);
 }

/*F**************************************************************************
* NAME: can_init
*-----------------------------------------------------------------------------
* PARAMS:   none
* return:   none
*-----------------------------------------------------------------------------
* PURPOSE:
*-----------------------------------------------------------------------------
* EXAMPLE:
*-----------------------------------------------------------------------------
* NOTE:
*-----------------------------------------------------------------------------
* REQUIREMENTS:
**************************************************************************/
void can_init(void)
{

// Init the CAN
int mob;

CAN_CONTROLLER_RESET;     // Reset CAN Controller
           //BRP=3, PRS=5, PHS1=4, PHS2=7 =>CAN baudrate=100K with clok =8MHz
           //BRP=3, PRS=2, PHS1=1, PHS2=1 =>CAN baudrate=250K with clok =8MHz
CanSetBRP (3);
CanSetPRS (2);
CanSetPHS1 (1);
CanSetPHS2 (1);
RazAllMailbox();               // Clear  Mailbox
CAN_RX_IT_ENABLE;              // Enable Receive Interrupt
CAN_BUF_IT_ENABLE;             // Enable Frame Buffer Interrupt
CAN_ERG_IT_ENABLE;             // Enable Global Errors Interrupt
```

```
CAN_ERCH_IT_ENABLE;             // Enable MOb Errors Interrupt
CAN_IT_ENABLE;                  // Enable CAN Interrupt
conf_rx = CONF_NOMSK_IDE | CONF_NORTR | CONF_NOMSK_RTR | CONF_BUFFER;  // 0 | 0 | 0 | 0

can_rx_filt.std = 0x100;        // ID 100 - 10F will be accepted
can_rx_msk.std  = 0x3F0;

//can_rx_filt.ext = 0x5555;       // ID 5554 - 5555 will be accepted
//can_rx_msk.ext  = 0xFFFE;

for (mob=0;mob<=14;mob++)
  {
  CAN_SET_CHANNEL(mob);         // Select MOb
  ConfChannel_Rx();             // Configure for Receiving
  CAN_CHANNEL_IT_ENABLE(mob);   // MOb Interrupt Enabled
  }
CAN_CONTROLLER_ENABLE;          // Enable CAN Controller

}
...
```

Listing 2 Programmbeispiel Transmit_Receive.C

Beginnen wir mit der Initialisierung des CAN-Controllers durch die Funktion can_init(). Zur Einstellung der gewünschten Baudrate sind die Register BRP, PRS, PHS1, PHS2 zu setzen. Wichtig ist hier, mit einem Prozessortakt zu arbeiten, der eine vernünftige Baudrate einstellen lässt. Leider taugen die üblichen Frequenzen, die eine genaue Baudrate des UART ermöglichen dafür nicht unbedingt. Hier wurde deshalb mit einem externen Quarz von 8 MHz gearbeitet.

Nach Freigabe diverser CAN-Interrupts werden die zu akzeptierenden Identifier gesetzt. Durch die Vorgabe von can_rx_filt.std = 0x100 und can_rx_msk.std = 0x3F0 werden die Identifier 0x100 bis 0x10F von unserem Knoten akzeptiert**.**

can_rx_filt.std	0	1	0	0	0	0	0	0	0	0
can_rx_msk.std	1	1	1	1	1	1	0	0	0	0
ID-Bereich	0	1	0	0	0	0	X	X	X	X

Beim nachfolgenden Initialisieren der Messageobjects erhält durch ConfChannel_Rx() jedes MOb den gleichen Inhalt.

Dass das nicht zwangsläufig so sein muss, zeigt der folgende Auszug (für MOb8) aus einem Programmbeispiel, was ebenfalls auf der mitgelieferten CD enthalten ist (Listing 3).

```
//      MOb   ID        IDE   Conf  DLC   Data
//      8     08        0     Tx    8     80 81 82 83 84 85 86 87
...
  j=0x80; for (i=0; i<8; i++) can_data8[i]= (j+i);
  can_tx_MOb8.pt_donne = can_data8;
  standart_ID=0x8; can_tx_MOb8.id.ext = standart_ID;
  can_tx_MOb8.ctrl = CONF_NOIDE | CONF_DLC_8;
...
```

Listing 3 Initialisierung MessageObject im AT90CAN128 (Auszug)

Nach diesen Initialisierung kann das Programm in seine Endlosschleife eintreten. Zuerst wird abgefragt, ob eine CAN-Mitteilung empfangen wurde (f_recept_can_message). Danach folgt die Überprüfung, ob es sich um ein Standard Frame oder ein Extended Frame handelt (pt_st_can_rx->ctrl & CONF_IDE)==CONF_IDE).

Nach Dekodierung der Mitteilung können Identifier und Länge der Mitteilung (DLC) ausgegeben werden (printf(" %x Rx %x ",pt_st_can_rx->id.std, (pt_st_can_rx->ctrl & 0x0F))), bevor sich die byteweise Ausgabe des Dateninhalts anschließt.

Abbildung 30 zeigt das zyklische Versenden von CAN-Mittelungen mit den Identifiern 0xFF, 0x100, 0x10F, 0x110 und 0x5000 im PCANView-Fenster. Der Hyperterminal-Mitschnitt zeigt, dass nur die Identifier 0x100 und 0x10F herausgefiltert und bearbeitet, alle anderen Identifier aber ignoriert wurden.

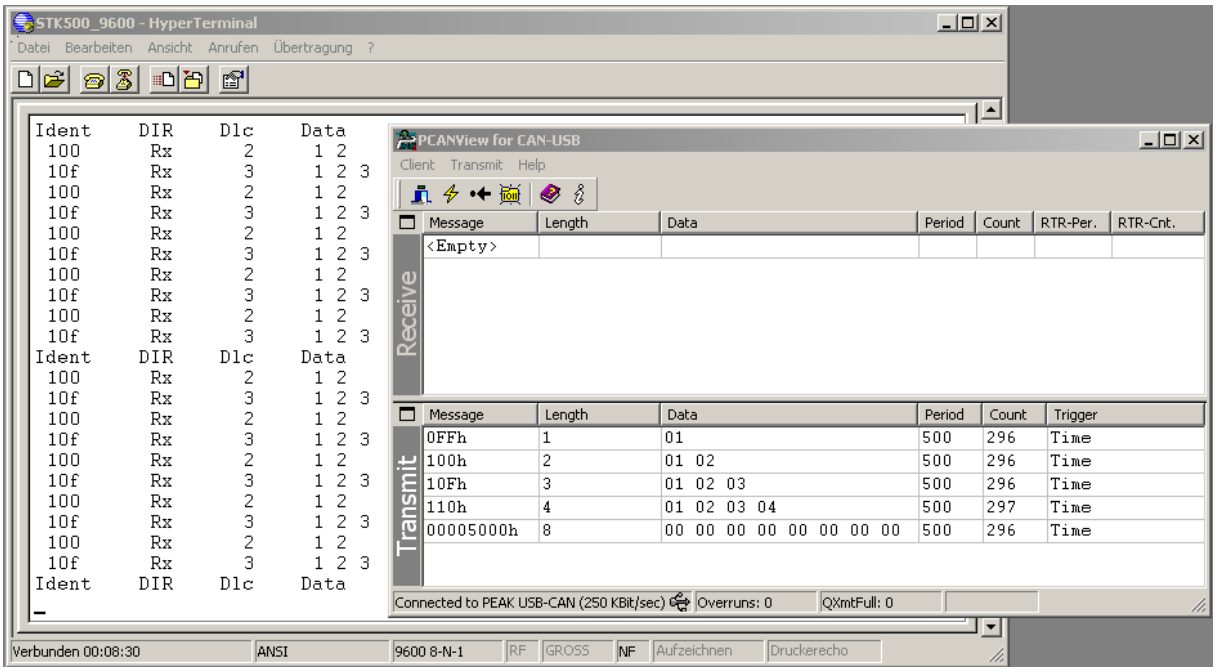


Abbildung 30 Filterung von CAN-Mitteilungen

5. Referenzen

[1] Marsh, D.:
CANBus Networks Break into Mainstream Use.
EDN, August 22, 2002, p.53-58

[2] Etschberger, K.:
Controller-Area-Network.
Fachbuchverlag, 2002
ISBN 3-446-21776-2

[3] Lawrenz, W.:
CAN Controller Area Network. Grundlagen und Praxis
Hüthig Verlag, 200
ISBN 3-7785-2780-0

[4] Pfeiffer, O.; Ayre, A.; Keydel, Chr.:
Embedded Networking with CAN and CANopen.
Annabooks/RTC Books, 2003
ISBN 0-9293-9278-7

[5] Controller Area Network (CAN)
www.can-cia.org/can/

[6] Modulbaureihe CST - Anwender-Handbuch
Modulbaureihe CST - Technische Daten
EMS Dr. Thomas Wünsche
www.ems-wuensche.com