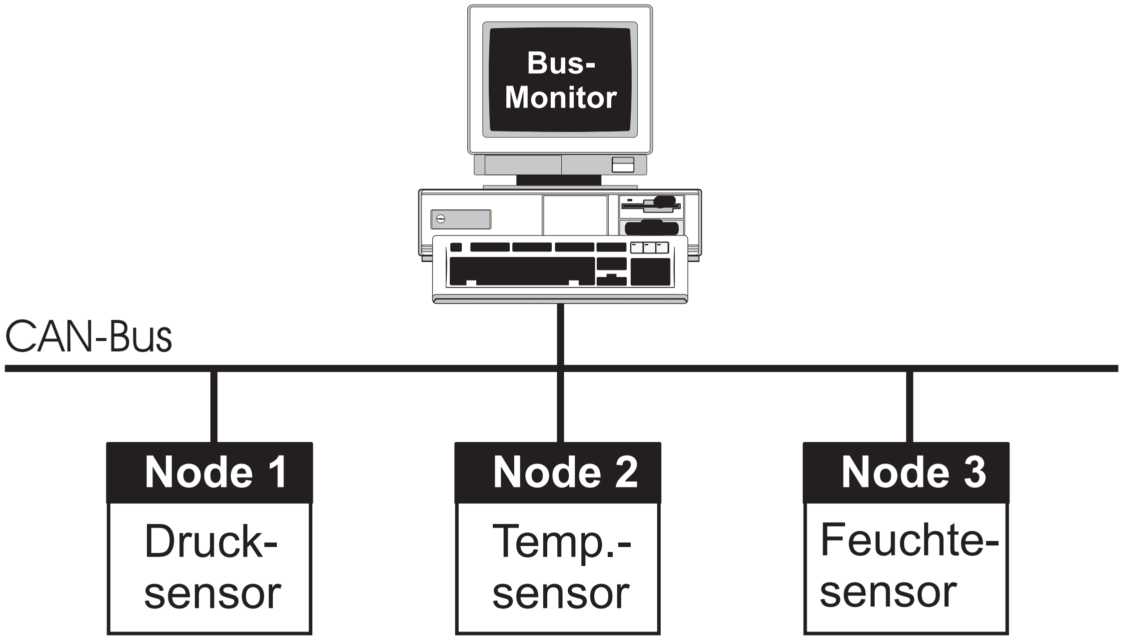
Aufbau und Analyse eines CAN-Bussystems in Anwendungen der Kfz- und Automatisierungstechnik



Holger Müller

Diplomarbeit

DA 1430

Technische Universität Darmstadt

Institut für Elektromechanische Konstruktionen

Prof. Dr.-Ing. H. Weißmantel

Prof. Dr.-Ing. R. Werthschützky

Aufgabenstellung

Diplomarbeit

von

Holger Müller

Aufbau und Analyse eines CAN-Bussystems in Anwendungen der Kfz- und Automatisierungstechnik

Die Ansteuerung von Sensoren und Aktoren wird durch den dezentralen Einsatz von μ-Controllern für den Benutzer fehlersicherer. Integration von Busschnittstelle und Elektronik führen zu leichter handhabbaren Modulen, da die Anpassung dem Anwender abgenommen wird.

In der vorliegenden Arbeit soll die Adaptation von Aktoren und Sensoren an ein CAN-Bussystem dargestellt werden. Schwerpunkt liegt dabei auf der Bewertung von Übertragungsgeschwindigkeit und Latenzzeit unterschiedlicher Übertragungsraten. Hierzu ist exemplarisch ein Sensorsystem für Druck Temperatur und Feuchte mit Linearisierung zum Einsatz in der Automobilindustrie an das Netzwerk anzupassen. Die Implementation soll auf Basis des Siemens C167 μ-Controllers erfolgen. Eine universelle Steuerplatine für Sensor und Aktoranwendungen ist zu erstellen. Das Modul soll sowohl an einem PC-gestütztem Entwicklungssystem als auch selbstständig arbeiten können.

Beginn:	08.06.1998
Ende:	11.09.1998
Vortragstermin:	05.11.1998
Betreuer:	Dipl. Ing. Sven Kuhn
Co.-Betreuer:	Dr. Ing. Elmar Hoppach (sitronic GmbH)

Zusammenfassung

In der vorliegenden Arbeit wird zunächst eine Einleitung in das CAN-Bussystem gegeben, ohne dabei Vergleiche zu anderen Feldbussystemen zu ziehen. An diese Einleitung schließt sich eine Übersicht der marktrelevanten CAN Application Layer Protokolle an. Die Auswahl eines Protokolls geschieht nach den Kriterien Verfügbarkeit, Brauchbarkeit für diese Aufgabe und Machbarkeit innerhalb der gesetzten zeitlichen und finanziellen Grenzen.

Die Auswahl eines für die Latenzzeitmessung geeigneten Protokolls fiel auf die Entwicklung eines eigenen, an CANopen angelehnten Protokolls. Mit Hilfe dieses Protokolls und zwei Microcontrollern des Typs C167 von Siemens wurde die Übertragungsverzögerung getrennt in Latenzzeit und Jitter für charakteristische Anwendungsfälle gemessen. Die Ergebnisse der Messungen stimmen mit den analytischen Berechnungen überein und ermöglichen den Nachweis der Echtzeitfähigkeit.

Im zweiten Teil der Arbeit wurde ein CAN-Meßknoten mit Druck-, Temperatur- und Feuchtesensors auf Basis des Microcontrollers C167 implementiert. Mit dieser „CAN-Wetterstation" wurden verschiedene Methoden zur Digitalisierung eines analogen Meßwertes entwickelt und bewertet. Sehr gute Ergebnisse zeigt die Messung der Periodendauer eines analogen Eingangssignals nach der Wandlung in eine Frequenz. Die Sensorsignale werden vom CAN-Meßknoten erfaßt, umgerechnet, linearisiert und über den CAN-Bus übertragen. Abb. 1 zeigt den entwickelten Sensorkopf.

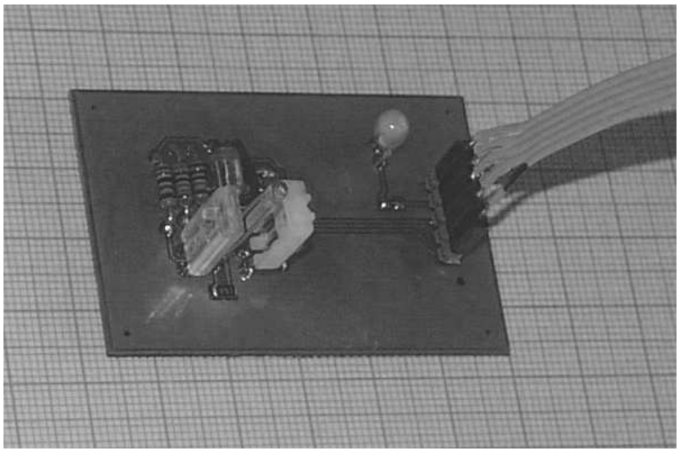


Abbildung 1: Sensorkopf mit Feuchte- und Temperatursensor

Technische Daten der CAN-Wetterstation

Microcontoller	SAB C167CR-LM mit integriertem 10 Bit A/D-Wandler, hochauflösenden Timern (0,2 μs) und CAN Schnittstelle
Feuchtesensor	0 .. 100% r.F.
Temperatursensor	-40 .. +85° C
Drucksensor	870 .. 1100 mbar
Systemgenauigkeit	0,11% analog, 0,0065% Periodendauermessung
Spannungsversorgung	+9 V, 110 mA

Inhaltsverzeichnis

Glossar **iii**

1 Einleitung **1**

2 Präzisierung der Aufgabenstellung **2**
2.1 Einführung 2
2.2 Pflichtenheft 2

3 Gliederung in Teilprobleme **3**

4 CAN - Entwicklungsumgebung **4**
4.1 CAN nach ISO 11898 4
4.2 CAN Protokolle 5
4.2.1 CAL 6
4.2.2 CANopen 7
4.3 Protokollauswahl 7
4.4 Einführung in die Entwicklungsumgebung 8
4.5 Implementierung des Protokolls 9

5 Latenzzeitmessung **10**
5.1 Analytische Berechnung 10
5.2 Auswahl des Meßprinzips 12
5.3 Messungen 13
5.4 Bewertung der Ergebnisse 15

6 Entwicklung von Sensoren und die Einbindung in CAN **15**
6.1 Sensorübersicht 15
6.2 Eingesetzte Sensoren 17
6.3 Digitalisierung 17
6.3.1 A/D-Wandlung 18
6.3.2 Periodendauermessung 18
6.3.3 Gegenüberstellung 19
6.4 Linearisierung der Sensoren 21

7 Kritik und Ausblick **22**

Literaturverzeichnis **23**

Anhang

A1 Zeitplan **A1**

A2 Kostenaufstellung **A2**

A3 Kontaktadressen **A3**

A3.1 CAN Hardware A3

A3.2 CAN Protokolle A3

A3.3 CAN Informationen A4

A3.4 C167 Compiler A4

A4 Das OSI Modell **A5**

A4.1 CAN und das OSI Modell A6

A5 Tabellen **A7**

A6 Datenblätter **A9**

A7 Schaltpläne **A17**

A8 Listings **A19**

Glossar

CAL CAN Application Layer. Ein CAN OSI Layer 7 Protokoll, welches von CiA gepflegt wird.

CAN-Bus Controller Area Network - von Bosch 1981 entwickeltes serielles Bussystem mit der Zielsetzung kostengünstig, ausfallsicher und leistungsfähig zu sein.

CAN-Knoten Gerät, welches an den CAN-Bus angeschlossen ist. Es wird auch häufig kurz mit dem englischen Wort „Node“ bezeichnet.

CANopen Ein CAN OSI Layer 7 Protokoll, welches von CiA gepflegt wird. Es ist ein deutlich vereinfachter Nachfolger des CAL Protokolls.

CiA CAN in Automation international users and manufactors group e.V.

CMS CAN Message Specification. Ein Service Element des CAL Modells.

COB Communication Object. Eine Transporteinheit in einem CAN Netzwerk. Daten werden innerhalb eines COB über das Netzwerk versendet.

COB-ID COB-Identifier. Identifiziert ein COB einheitlich in einem Netzwerk. Der Identifier bestimmt auch die Priorität eines COB auf MAC Ebene.

DBT Distributor. Ein Service Element des CAL Modells, das für die Vergabe der COB-ID's zu den COB's verantwortlich ist.

ECP Enhanced Capability Port. Von Microsoft entwickelter erweiterter bidirektionaler Druckerport, der IEEE (P1284) genormt ist. Schwerpunkt von ECP ist ein als FIFO ausgeführter 16-Byte-Puffer mit Interrupt- sowie DMA-Fähigkeit.

Feldbus Zubringerbus für die Meßwerterfassung sowie der Steuerung und Regelung. Er dient der Übertragung von Signalen von Sensoren, Aktoren und sonstigen Feldgeräten zu einem μ-Controller. Beispiele für den Feldbus sind der Profi-Bus oder der CAN-Bus.

ISO International Standardisation Organisation.

LLC Logical Link Control. Teil des OSI Data-Link Layer in Richtung der höheren Layer.

LMT Layer Management. Ein Service Element des CAL Modells. Es stellt Konfigurationsparameter für jeden Layer des CAN Modells zu Verfügung.

MAC Media Access Control. Teil des OSI Data-Link Layer in Richtung Physical Layer.

NMT Network Management. Ein Service Element des CAL Modells. Es führt die Initialisierung, Konfiguration und Fehlerbehandlung durch.

OSI Open Systems Interconnection - Modell. Beschreibt die Kommunikation zwischen einem Computer und einem beliebigen Netzwerk.

PDO Process Data Object. Teil der CAL Definition.

SDO Service Data Object. Teil der CAL Definition.

1 Einleitung

Feldbussysteme dienen als Kommunikationsmedium für verteilte Aktoren, Sensoren und Steuerungen in der Automatisierungstechnik und überall dort, wo mehr oder minder „intelligente“ elektronische Bausteine mit geringem Verkabelungsaufwand verbunden werden müssen. An einen Feldbus und die angeschlossenen Elemente werden hohe Anforderungen gestellt: sie müssen kostengünstig, stör- und ausfallsicher sowie technisch sehr leistungsfähig sein.

Im Automobilbereich treffen diese Anforderungen auf die günstige Tatsache der weiten Verbreitung und des Masseneinsatzes. Die wiederum sind Voraussetzung für geringe Preise und die Standardisierung des Feldbussystems und dessen Anschlußelemente.

Der Controller Area Network (CAN)-Bus wurde 1981 von Bosch und Intel entwickelt und findet international weite Verbreitung im Automobil- (Audi, BMW, Mercedes, Porsche) und Haushaltsgerätesektor (Bosch). Durch die hohen Stückzahlen sind preiswerte Anschaltkomponenten erhältlich. In der Automobilindustrie hat allerdings jeder Fahrzeughersteller sein eigenes Protokoll, was zu einer entsprechenden Vielfalt und zu Problemen bei der Austauschbarkeit führt. Bei der Etablierung in der Industrie wurde eine eigene Nutzerorganisation „CAN in Automation e.V.“ (CiA e.V.) geschaffen, die sich um die Standardisierung einiger weniger Protokolle bemüht. Eine derartige Standardisierung ist sehr wichtig, da eine Neuentwicklung eines CAN Protokolls mehrere Mannjahre benötigt und einen kostengünstigen Einsatz verhindern würde.

Die Übertragungssicherheit von Nachrichten in einem CAN-Bus ist durch Maßnahmen auf unterster Protokollebene gegeben. Die Echtzeitfähigkeit und definierte Übertragungsdauer hingegen ist oft Diskussionspunkt im Zusammenhang mit CAN. Mit Hilfe theoretischer Betrachtungen und Messungen der Latenzzeit bei der Nachrichtenübermittlung soll hier gezeigt werden, daß die Übertragungsdauer sehr wohl berechenbar ist.

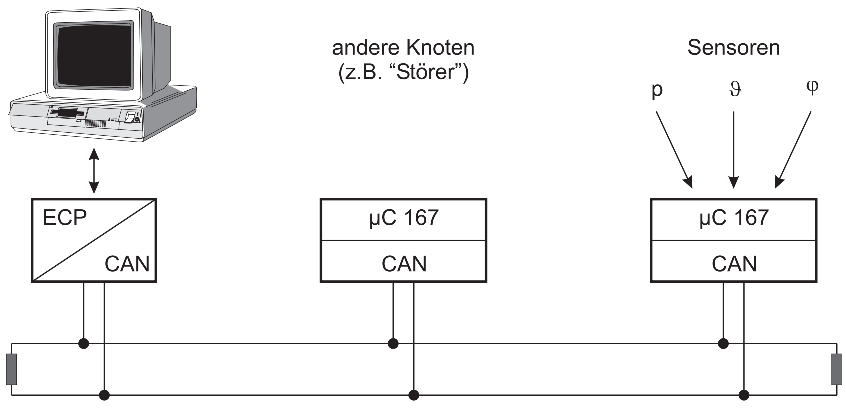


Abbildung 1.1: CAN Netzwerk mit Sensor-Knoten

Für die praktische Demonstration des CAN-Netzwerks ist die Entwicklung einer CAN-Wetterstation sehr anschaulich. Ein typisches Konfigurationsbeispiel ist in

Abb. 1.1 dargestellt. Die verwendeten sehr leistungsfähigen Microcontroller (μC) C167 eignen sich neben der einfachen Ansteuerung des CAN-Busses hervorragend zur Meßwerterfassung. Für die Wetterstation kommen Druck-, Temperatur- und Feuchtesensoren zum Einsatz. Dabei werden verschiedene Verfahren zur Meßwerterfassung vorgestellt.

Diese Arbeit ist in Zusammenarbeit mit der Firma sitronic in Gärtringen entstanden und durchgeführt worden.

2 Präzisierung der Aufgabenstellung

2.1 Einführung

In einem ersten Teil ist die Verzögerung von Nachrichten auf einem CAN basierten Bus-System zu zeigen und zu messen. Dazu sind verschiedene Microcontroller und ein PC einzusetzen. Für diese Anwendung wird bereits ein Protokoll der Anwendungsebene (OSI Layer 7, s. Kap. A4) benötigt.

Das im ersten Teil entwickelte CAN System ist zu einer Wetterstation zu erweitern. Dazu sind die Meßgrößen Druck, Temperatur und Feuchte zu erfassen und über den Bus zu übertragen. Es sind verschiedene Verfahren zur Meßwerterfassung zu entwickeln und miteinander zu vergleichen. Desweiteren ist eine Linearisierung der erfaßten Meßgrößen durchzuführen.

2.2 Pflichtenheft

Dieses tabellarische Pflichtenheft gliedert sich in **F**orderungen und **W**ünsche. Die jeweiligen Abkürzungen sind dem Glossar zu entnehmen.

Pflichtenheft A: Latenzzeitmessung

F Aufbau eines CAN-Bus Systems mit 100 KBit/s

F Integration zweier CAN-Knoten (PC und μC C167)

W Integration eines dritten CAN-Knoten auf Basis eines μC C167

W Realisierung eines CAN OSI Layer 7 Protokolls

F Messung der Latenzzeit des Systems

- Analytische Berechnung der Latenzzeit
- Messung mit und ohne Buslast
- Messung bei unterschiedlichen Prioritäten der Buslast

F Messung des Jitters bei konkurrierendem Zugriff

Pflichtenheft B: CAN Wetterstation

- **F** Verwendung des CAN-Bus Systems aus Teil A
- **F** Einbindung eines Druck-, Temperatur- und Feuchtesensors
- **F** Messung der Sensorsignale mit einem μC C167
- **F** Anbindung mehrerer A/D-Wandler Kanäle
- **W** Messung mittels frequenzanaloger Primärelektronik
- **F** Linearisierung aller Sensoren anhand bekannter Linearisierungskoeffizienten
- **F** Sensorsystem soll auch ohne PC arbeiten

3 Gliederung in Teilprobleme

Aus dem Pflichtenheft können Teilprobleme erarbeitet werden, die an die Priorität des Problems gekoppelt sind.

- **A** Messung der Latenzzeiten in einem realen CAN-Bus System

- **B** Untersuchung vorhandener CAN Protokolle
- **B** Implementierung eines CAN Protokolls auf dem C167
- **B** Integration eines Druck-, Temperatur- und Feuchtesensors
- **B** Linearisierung des Drucksensors auf Softwareebene (C167)

- **C** Aufbau eines CAN-Bus Systems

Für den zeitlichen Ablauf ist eine andere Reihenfolge nötig. Beginnen muß man mit einer Einarbeitung in die CAN Materie. Dazu gehören neben der Hardware auch eine Menge theoretischer Informationen, wie z.B. die OSI Netzwerk Definition.

Für den PC ist eine geeignete Hardware zur Kommunikation zwischen PC und CAN zu finden. Als Auswahlkriterien werden hierfür Gesichtspunkte wie Nähe der Firma, verfügbare Software sowie Beratung sein. Daraufhin muß die C167 Hardware in Betrieb genommen und die Ansteuerung der CAN Schnittstelle programmiert werden. Auf dieser Ebene läßt sich bereits eine einfache Kommunikation zweier CAN-Knoten ohne OSI Layer 7 Protokoll zeigen.

Nach diesen grundlegenden Arbeiten muß ein geeignetes CAN Layer 7 Protokoll gesucht und ausgewählt werden. Die Auswahl wird dabei in Zusammenarbeit mit der Firma sitronic geschehen. Es ist darauf Rücksicht zu nehmen, daß die Informationen über den Druck, die Temperatur und die Feuchte in das Protokoll integriert werden können. An dieser Stelle geschieht ebenfalls die Integration der Sensoren an den C167 μ-Controller.

Erst nach diesen Tätigkeiten kann zum wesentlichen Punkt der Arbeit übergegangen werden, der Messung der Latenzzeit auf dem CAN Bus. Die Latenzzeit beschreibt die Verzögerung bei der Übertragung von Nachrichten. Sie wird durch andere Nachrichten (in diesem Fall Störer) stark beeinflußt. Dazu müssen geeignete Verfahren entwickelt und die erforderlichen Messungen durchgeführt werden.

Vor der Implementation muß die Einarbeitung in die Entwicklungsumgebung erfolgen.

4 CAN - Entwicklungsumgebung

Controller Area Network (CAN) ist ein nach ISO 11898 [5] Norm definierter serieller Multimaster-Kommunikationsbus, der 1981 von Bosch und Intel entwickelt wurde. Beschrieben wird in der ISO 11898 [5] der Physical und der Data-Link Layer des OSI Modells (s. Kap. A4). Der Bus kann Datenraten bis 1 MBit/s verarbeiten (s. auch Tabelle A5.2), bietet hohe Sicherheit gegen elektrische Störungen und die Möglichkeit, auftretende Fehler zu erkennen.

4.1 CAN nach ISO 11898

Jeder Knoten (Node) hört auf beliebig viele Kennungen (ID's). Hierbei sind verschiedene Varianten ausgeprägt: Variante 2.0A spezifiziert ein 11 Bit und Variante 2.0B eine 29 Bit lange Nachrichtenkennung (ID). Damit sind 2032 oder über 536 Millionen Kennungen möglich. Die Einschränkung der Variante 2.0A auf 2032 Kennung hängt mit der Entwicklung des ersten CAN Controllers von Intel zusammen, der eine derartige Einschränkung aufwies.

Zur Vermeidung von Kollisionen wird bei CAN das Verfahren der bitweisen Arbitrierung verwendet. Es gibt dominante und rezessive Signale. Nachrichten werden immer in Paketen (sogenannten „Frames“) übertragen, die dem Aufbau von Abb. 4.1 entsprechen. Jeder Knoten im Netzwerk darf senden, sobald der Bus frei (idle) ist. Nach dem „Start of Frame“ Bit wird sofort das „Arbitration field“ (11 Bit) gesendet, um auszuhandeln welcher Knoten den Bus letztendlich bekommt. Der Knoten, der zuerst ein rezessives Bit gegen ein dominantes sendet verliert den „Wettstreit“ und darf nicht weiter mitsenden. Die Felder haben die folgende Bedeutung und Bitlängen:

- „Start of Frame“ (1 Bit) markiert den Beginn einer Übertragung.
- „Arbitration field“ (12 Bit) beinhaltet die Nachrichtenkennung (ID) und das RTR (Remote Transmission Request) Bit der Nachricht. Durch die bitweise Arbitrierung wird gleichzeitig die Priorität der Nachricht bestimmt.
- „Control field“ (6 Bit) bestimmt die Länge des nachfolgenden „Data Field“ in Bytes.

- „Data field“ (0 - 64 Bit) trägt die eigentlich zu übertragende Nachricht.
- „CRC field“ setzt sich aus 15 Bit CRC Summe und einem 1 Bit Delimiter zusammen.
- „ACK field“ (2 Bit) teilt sich in „ACK slot“ und „ACK Delimiter“ auf. Alle empfangenden Knoten des Netzwerks überschreiben das rezessive „ACK slot“ Bit des Senders mit einem dominanten Bit und teilen damit die fehlerfreie Übertragung dem Sender mit. Im Falle eines Fehlers wird die Nachricht erneut übertragen.
- „End of Frame“ (1 Bit) markiert das Ende der Übertragung.

Start of frame	Arbitration field	Control field	Data field	CRC field	ACK field	End of frame

Abbildung 4.1: MAC Data Frame

Insgesamt können Nachrichten zwischen 44 und 108 Bit lang sein. Zusätzlich ist aber zu berücksichtigen, daß zwischen den einzelnen Frames ein „Interframe space“ vorgesehen ist, der 3 rezessive Bits beträgt.

Da jeder CAN-Knoten eine Nachricht immer an alle Busteilnehmer sendet, entsteht eine entsprechende Belastung für den jeweiligen CAN-Host, die für ihn adressierten Nachrichten herauszufiltern. Dies kann im ungünstigen Fall soweit führen, daß der Hostrechner ausschließlich damit beschäftigt ist, Nachrichten auf ihre Relevanz hin zu prüfen. Derartige CAN-Implementierungen werden als Basic-CAN bezeichnet. Mit der Full-CAN-Implementierung wird die Nachrichtenpufferung und -filterung vollständig vom CAN-Protokollbaustein übernommen. Hierbei werden in Hardware feste Nachrichtenpuffer realisiert, und der CAN-Baustein blendet über „Akzeptanz Filter“ alle für ihn nicht relevanten Nachrichten selbstständig aus. Dies ermöglicht eine drastische Reduktion der CPU Belastung.

4.2 CAN Protokolle

An dieser Stelle wird die Bezeichung „CAN Protokolle“ ausschließlich für Protokolle des OSI Layer 7 (Application Layer, s. Kap. A4) verwendet. Das Protokoll des CAN Data-Link Layer der ISO 11898 unterstützt nur die Übertragung und Anfrage von bis zu 8 Datenbytes. Praktische Anwendungen benötigen aber meist mehr als 8 Datenbytes. Auch ist häufig eine Initialisierung und Verwaltung der CAN-Knoten notwendig. Ein Application Layer Protokoll sollte also die folgenden Aufgaben erfüllen:

- Transparente Übertragung von Daten
- Vergabe der Identifier (ID's)
- Network Management (z.B. Hochfahren des Netzwerks)
- Synchronisation

- Übermitteln einer Zeitbasis

Im Automobilbereich wird von jedem Hersteller ein eigener Application Layer verwaltet, der i.d.R. nicht an dritte freigegeben wird. Beispiele hierfür sind:

- der Komfort-Bus bei Audi
- der I-, K-, M-Bus bei BMW
- der Comfort-Bus bei Mercedes

In der industriellen Automatisierungtechnik ist ein anderer Weg gegangen worden. Hier bemüht sich eine eigene Nutzerorganisation „CAN in Automation e.V.“ (CiA e.V.) um die Standardisierung einiger weniger Protokolle. So wurde 1992 von der CiA Benutzergruppe die Spezifikation eines allgemein akzeptierten Application-Layer-Protokolls besonders für industrielle Systeme initiiert. Dazu wurde ein bereits existierendes und getestetes Protokoll von Philips Medical Systems übernommen und erweitert. 1993 wurde dies von der CiA als „CAN Application Layer (CAL)“ *DS 201 ... 207* [2] herausgegeben.

Neben CAL unterstützt CiA die folgenden Protokolle, die in verschiedenen Ländern ihren Ursprung und Einsatz haben:

- CANopen (Europa)
- CAN Kingdom (Skandinavien)
- DeviceNet (USA von Rockwell/Allen Bradley)
- Smart Distributed System (SDS) (von Honeywell)

CANopen ist aus CAL heraus entwickelt worden und stellt eine Vereinfachung dar. Diese beiden Protokolle sollen im folgenden kurz betrachtet werden.

4.2.1 CAL

Neben den beiden Basisdiensten des CAN Data-Link Layer bietet CAL eine offene Umgebung für offene Standard Netzwerk Lösungen. CAL bietet vier Application Layer Service Elemente:

- CAN based Message Specification (CMS)
- Network Management (NMT)
- Identifier Distributor (DBT)
- Layer Management (LMT)

Der in jedem Netzwerk vorhandene CAN-Master stellt diese Dienste zur Verfügung und verwaltet damit alle im Netzwerk vorhandenen Clients. Weitere Informationen können der CiA Dokumentation *DS 201 ... 207* [2] entnommen werden. CAL ist ein älteres Protokoll, aus dem CANopen entstanden ist. Es gibt viel Parallelen, so daß hier nicht näher auf CAL eingegangen werden soll.

4.2.2 CANopen

Die Spezifikationen von CANopen in *DS 301* [3] beschreiben einen minimalen Funktionsumfang, den jeder Knoten erfüllen muß. So muß z.B. jeder Knoten ein Objektverzeichnis kennen, in dem alle für diesen Knoten relevanten CANopen-Objekte eingetragen sind. Lesende und schreibende Zugriffe erfolgen mit einem Service Data Object (SDO). Sie werden für Änderungen im Objektverzeichnis und für Statusabfragen verwendet. Jeder CANopen Knoten hat mindestens ein SDO, dem zwei CAN-Identifier zugeordnet sind. Mit dem „Multiplexed Domain Transfer Protocol“ der CAL Definition lassen sich Daten beliebiger Länge übertragen. Die Daten werden ggf. auf mehrere CAN-Nachrichten aufgeteilt (segmentiert). In der ersten CAN-Nachricht des SDO sind vier Byte mit Protokollinformationen belegt. Bei Nachrichten bis 4 Byte Länge genügt folglich eine einzige CAN-Nachricht. Für Daten mit mehr als 4 Byte erfolgt eine Aufteilung auf mehrere Nachrichten, bei der ab der 2. CAN-Nachricht jeweils 7 Byte für Nutzdaten zur Verfügung stehen. Jedes SDO wird vom Empfänger durch eine entsprechende CAN-Nachricht bestätigt. Insgesamt entsteht ein nicht unerheblicher Overhead durch das Protokoll.

Für die Übertragung von Prozeßdaten kommen die Process Data Objects (PDO) zur Anwendung. Es wird das „Stored Event Protocol“ von CAL verwendet. Dem Knoten stehen dabei alle 8 Datenbytes der CAN-Nachricht zur Verfügung. PDO’s werden im Gegensatz zu den SDO’s unbestätigt übertragen, da der CAN Data-Link Layer (OSI Layer 2) eine fehlerfreie Übertragung sicherstellt. Dadurch steht den PDO’s die maximale Busbandbreite ohne den Protokolloverhead der SDO’s zur Verfügung. Dies ist besonders bei zeitkritischen Anwendungen wichtig.

Die Zuordnung der CAN-Identifier erfolgt automatisch mit Hilfe der eingestellten Knotennummer und der definierten Objekte (COB’s, s. Tabelle 4.1). PDO’s haben demnach eine höhere Priorität als die SDO’s. Die höchste Priorität ist für das Netzwerk-Management (NMT) vorgesehen, mit dem das Netzwerk gestartet und gestoppt wird. Die COB-ID’s werden in Klassen (Priority Level) aufgeteilt (s. Tabelle A5.1). Die anderen definierbaren Objekte, wie EMERGENCY, SYNC usw., sind optional. Ein „Minimum Capability Device“, also ein Knoten mit minimalem Funktionsumfang, muß zumindest die Dienste bzw. COB’s NMT, SDO und PDO unterstützen.

Neben dem Protokoll sind in CANopen auch die empfohlenen Datenraten und die daraus folgenden maximalen Buslängen und Bitzeiten definiert (s. Tabelle A5.2). Jeder CANopen Knoten muß 20 KBit/s unterstützen, da mit dieser Datenrate das Netzwerk initialisiert wird.

4.3 Protokollauswahl

Durch die Anbindung der Arbeit an die Automobilindustrie stand der Wunsch nahe, ein Protokoll eines Fahrzeugherstellers zu verwenden. Die Realisierung scheiterte an der Verfügbarkeit eines entsprechenden Protokolls, da alle Fahrzeughersteller dies nur an Zulieferer mit entsprechenden Geheimhaltungsverträgen weiterreichen. Andererseits ermöglichen diese Protokolle ohne weitgehende Veränderung keine Latenzzeitmessung. Die weitere Prüfung der verfügbaren Protokolle in Hinsicht auf die

Objekt	function code (binär)	COB-IDs	Priority Level
NMT	0000	0	0
SYNC	0001	128	0
TIME STAMP	0010	256	1
EMERGENCY	0001	129 .. 255	0, 1
PDO1 (tx)	0011	385 .. 511	1, 2
PDO1 (rx)	0100	513 .. 639	2
PDO2 (tx)	0101	641 .. 767	2, 3
PDO2 (rx)	0110	769 .. 895	3, 4
SDO (tx)	1011	1409 .. 1535	6
SDO (rx)	1100	1537 .. 1663	6, 7
Nodeguard	1110	1793 .. 1919	-

Tabelle 4.1: Unterstützte Objekte und ihre zugeordneten COB-ID's nach CANopen

Brauchbarkeit für diese Arbeit, führte schnell zu CANopen. Die offene Struktur läßt Freiraum für nahezu alle möglichen Anwendungen. Andererseits setzt CANopen eine aufwendige Programmierung von ca. 3 - 4 Mannjahren voraus. Fertige CANopen Master und Clients sind auf dem freien Markt (auch im Source-Code) verfügbar. Die Kosten für den Source-Code liegen allerdings im Bereich von 10 - 20 TDM. Eine kostengünstige Alternative versprach der CAN-Master „CANsetter“ von STA Reutlingen (s. Kap. A3.2) sowie der kostenlose „Minimum Capability Device“ Client Beispielcode für den Siemens C515. Der C515 hat den gleichen CAN Controller wie der C167 und läßt sich deshalb einfach portieren. Dennoch ließ sich diese Variante aus zeitlichen Gründen nicht realisieren, so daß die Entscheidung auf ein eigenes, an CANopen angelehntes Protokoll fiel.

4.4 Einführung in die Entwicklungsumgebung

Aufgrund der Vorgaben werden mindestens zwei - besser drei - CAN Knoten benötigt. Grundlage bilden die beiden C167 Entwicklungsboards, die bereits vorhanden waren und durch ihre Architektur ideal zur Meßwerterfassung und Latenzzeitmessung geeignet sind. Die Ankopplung an den PC sollte so flexibel wie möglich erfolgen, so daß ein Adapter für die Druckerschnittstelle (CAN-Dongle) am besten geeignet ist.

Im einzelnen wurden folgende Geräte für den CAN Bus eingesetzt:

- CAN-Dongle der Firma PEAK-Service am Druckerport eines Pentium 133 mit dem Betriebssystem NT 4.0
- C167 μC-Board der Firma Phytech mit CAN Transceiver
- C167 μC-Board der Firma Keil ohne CAN Transceiver

Das Board der Firma Keil ist mit dem CAN Transceiver 82C250 erweitert worden, um die CAN Funktionalität bereitzustellen.

4.5 Implementierung des Protokolls

Wie in Kap. 4.3 beschrieben, ist der Entwicklung eines eigenen, einfachen Protokolls Vorzug zu geben. Das hier beschriebene Protokoll entstand zwar in Anlehnung an CANopen, kann jedoch nicht annähernd als CANopen kompatibel bezeichnet werden. Nur für die Aufgabe wichtige Teile wurden übernommen, etliche stark vereinfacht oder durch völlig andere Mechanismen ersetzt. In Abb. 4.2 ist als Übersicht das vereinfachte Flußdiagramm eines Clients dargestellt.

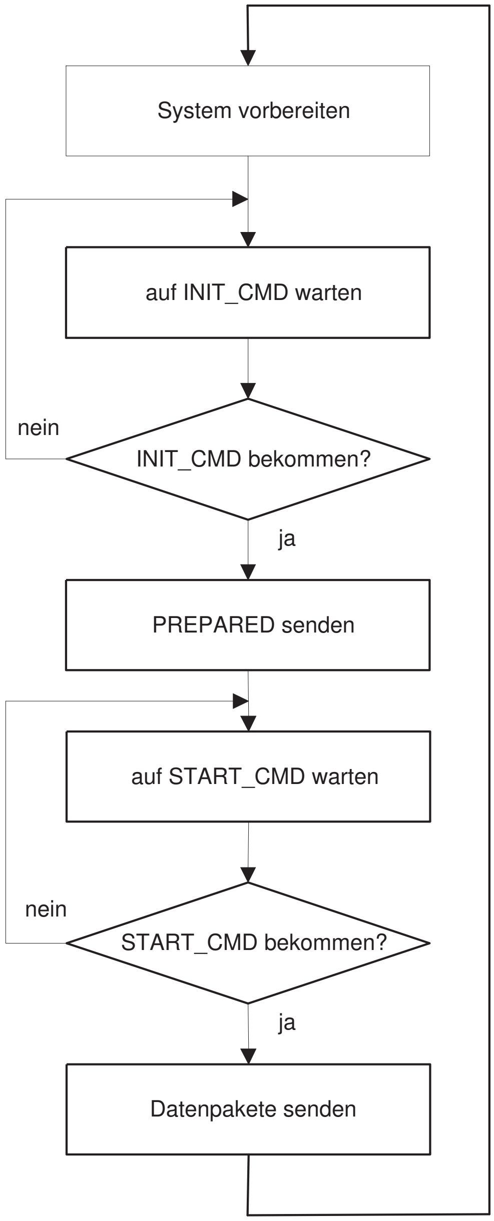


Abbildung 4.2: Flußdiagramm der CAN Kommunikation eines Clients

Im Netzwerk sind 3 Knoten (Nodes) vorhanden:

CANmaster: PC mit einem CAN-Dongle und der Hardwareadresse 0x01. Aufgaben:

- CAN-Master zum Initialisieren des Netzwerks.
- Verantwortlich für die kontrollierte Buslast.
- Schreiben der vom Client „CANlog“ protokollierten Daten in eine Datei.

CANsend: Phytech C167 Board mit der Hardwareadresse 0x02. Aufgaben:

- Senden von Daten mit der ID 400 (SYNC Priorität) im zeitlichen Abstand von 10 ms.

CANlog: Keil C167 Board mit der Hardwareadresse 0x03. Aufgaben:

- Empfangen der Daten von „CANsend“.
- Messen der Zeit zwischen den Nachrichten.
- Übertragung der Meßdaten nach 256 Messungen an den „CANmaster“.

Der „CANmaster“ sendet fortlaufend das `INIT_CMD` Kommando und wartet auf das `PREPARED` Signal der Clients. Nachdem sich alle Clients `PREPARED` gemeldet haben, sendet der Master das `START_CMD` Kommando um die Messung zu starten. Der „CANmaster“ beginnt sofort mit der Generierung der gewünschten Buslast. Der Knoten „CANsend“ sendet im 10 ms Takt die zu messenden Nachrichten. „CANlog“ wird die Zeit zwischen den Nachrichten messen und nach 256 Messungen mit höchster Priorität an den „CANmaster“ senden. Nach Empfang aller Meßdaten sendet der „CANmaster“ das Kommando `STOP_CMD`, wodurch die Clients wieder in den Zustand gehen auf das `INIT_CMD` Kommando des „CANmaster“ zu warten.

5 Latenzzeitmessung

Die Datensicherheit ist beim CAN-Bus durch umfangreiche Sicherungsmaßnahmen (CRC-Prüfsumme, Empfangsbestätigung, automatische Sendewiederholung) gewährleistet. Kritische Betrachtungen müssen allerdings bei der Echtzeitfähigkeit durchgeführt werden, da nicht immer der sofortige Zugriff auf den Bus garantiert werden kann. Für den Anwender ist sowohl die mittlere Antwortzeit als auch die maximale Verzögerung wichtig. Aus einer hinreichend großen Zahl von Messungen ist deshalb die statistische Verteilung zu bestimmen.

5.1 Analytische Berechnung

Für jeden Zugriff auf den CAN Bus entsteht eine Verzögerung, die sich aus

- der nicht erlaubten Unterbrechung einer laufenden Nachricht und

- dem Abweisen einer Nachricht durch eine andere Nachricht mit höherer Priorität

zusammensetzt. Im Detail ist das bereits in Kapitel 4.1 beschrieben.

Für die Berechnung sind einige Größen notwendig, die im folgenden definiert werden:

CAN Bustakt: 100 kHz, d.h. 10 μs Bitzeit, bzw. Bustaktzyklen
CAN Meßnachricht: 47 Bit $\leadsto$ $\tau_{Mess} = 0,47$ms Übertragungsdauer
CAN Störnachricht: 111 Bit $\leadsto$ $\tau_{Stör} = 1,11$ms Übertragungsdauer
Sendezeitpunkt der Meßnachricht: t_{Mess}
Sendezeitpunkt der Störnachricht: $t_{Stör}$

Man kann im wesentlichen 3 Fälle unterscheiden:

1. Der Bus ist frei (idle), so daß die Nachricht sofort übertragen werden kann. Dies entspricht dem günstigsten Fall.

$$t_{Lat} = 0 \quad \pm 3 \text{ Bustaktzyklen} \tag{5.1}$$

2. Zum Zeitpunkt der Sendeanforderung wird bereits eine Nachricht übertragen. Es muß bis zum Ende der Übertragung gewartet werden, was bis zu 111 Bitzeiten dauern kann (vgl. Kap. 4.1).

$$t_{Lat} = \tau_{Stör} - (t_{Mess} - t_{Stör}) \quad \forall \; t_{Stör} < t_{Mess} \leq (t_{Stör} + \tau_{Stör}) \tag{5.2}$$

3. Eine Nachricht mit niedriger Priorität verliert n-Mal die Arbitrierung und muß die Übertragung der Nachrichten mit höherer Priorität abwarten. Im ungünstigsten Fall bekommt die Nachricht den Bus niemals, wartet also ewig auf die Übertragung der anderen Nachrichten. Bei CANopen wurden für diesen Fall besondere Maßnahmen getroffen, damit Nachrichten mit niedriger Priorität nicht dauerhaft vom Bus verdrängt werden können.

$$\begin{aligned} t_{Lat} &= \tau_{Stör} - (t_{Mess} - t_{Stör}) + \sum_{i=1}^{n} \tau_{Stör} \\ &= t_{Stör} - t_{Mess} + \sum_{i=1}^{n+1} \tau_{Stör} \quad \forall \; t_{Stör} < t_{Mess} \leq (t_{Stör} + \tau_{Stör}) \end{aligned} \tag{5.3}$$

Allgemein (unabhängig von dem vorliegenden Fall) betrachtet ist die Verzögerung eine statistische Größe, deren Funktion sich aus der Buslast, den verwendeten Prioritäten und der eigenen Priorität zusammensetzt.

5.2 Auswahl des Meßprinzips

Möglichkeiten/Varianten der Latenzzeitmessung:

A) Elektrische Messung auf dem Bus mit Ankopplung an den Sendezeitpunkt. Probleme bereiten die exakte Auskopplung des Sendezeitpunkts sowie die Protokollierung der gemessenen Latenzzeit.

B) Übertragung des Sendezeitpunkts in der Nachricht mit Auswertung beim Empfänger bzgl. dessen Zeit.
Hier besteht die Schwierigkeit der Synchronisierung von Sender und Empfänger.

C) Knoten A sendet ein Start Kommando an Knoten B und C. Daraufhin sendet Knoten B sofort eine Nachricht an Knoten C. Knoten C mißt die Verzögerung zwischen dem Start Kommando und dem Empfang der Nachricht von B.

D) Senden einer Nachricht aufgrund der Anforderung mit einem „Remote Frame“. Der anfordernde Knoten mißt die Zeit zwischen Anforderung und Empfang der Nachricht.

E) Senden einer Nachricht in einem festen Zeitraster. Messung des zeitlichen Abstands zwischen den Nachrichten.

Da an dieser Stelle noch nicht eindeutig ersichtlich ist, welche Variante am günstigsten ist, wird eine Punktewertung[1] durchgeführt. Die Auswahlkriterien sollen dabei „Implementierungsaufwand“, „Grundgenauigkeit“ (wie genau kann systembedingt gemessen werden), „Reproduzierbarkeit“ und „Datengenerierung beim Senden“ (können Daten für die Übertragung generiert werden) sein.

Variante	A	B	C	D	E
Implementierungsaufwand	1	3	3	4	3
Grundgenauigkeit	1	1	2	3	4
Reproduzierbarkeit	2	3	2	3	4
Datengenerierung beim Senden	2	3	4	0	4
Summe	6	10	11	10	**15**

Aufgrund dieser Bewertung eignet sich die Variante E am besten.

[1] 4 Punkte ≡ ideal, 3 Punkte ≡ gut, 2 Punkte ≡ ausreichend, 1 Punkt ≡ gerade noch tragbar, 0 Punkte ≡ unbefriedigend

5.3 Messungen

Zur Realisierung der in Kapitel 5.2 beschriebenen Variante wurde das in Kapitel 4.5 beschriebene Protokoll implementiert. Nach dem `START_CMD` des CANmaster sendet der Client CANsend im 10 ms Takt Datenpakete mit 1 Byte Länge an den Client CANsend und wird dabei ggf. vom CANmaster gestört. CANsend bestimmt den zeitlichen Abstand zwischen den einzelnen Paketen mit einer Auflösung von 0,4 μs.

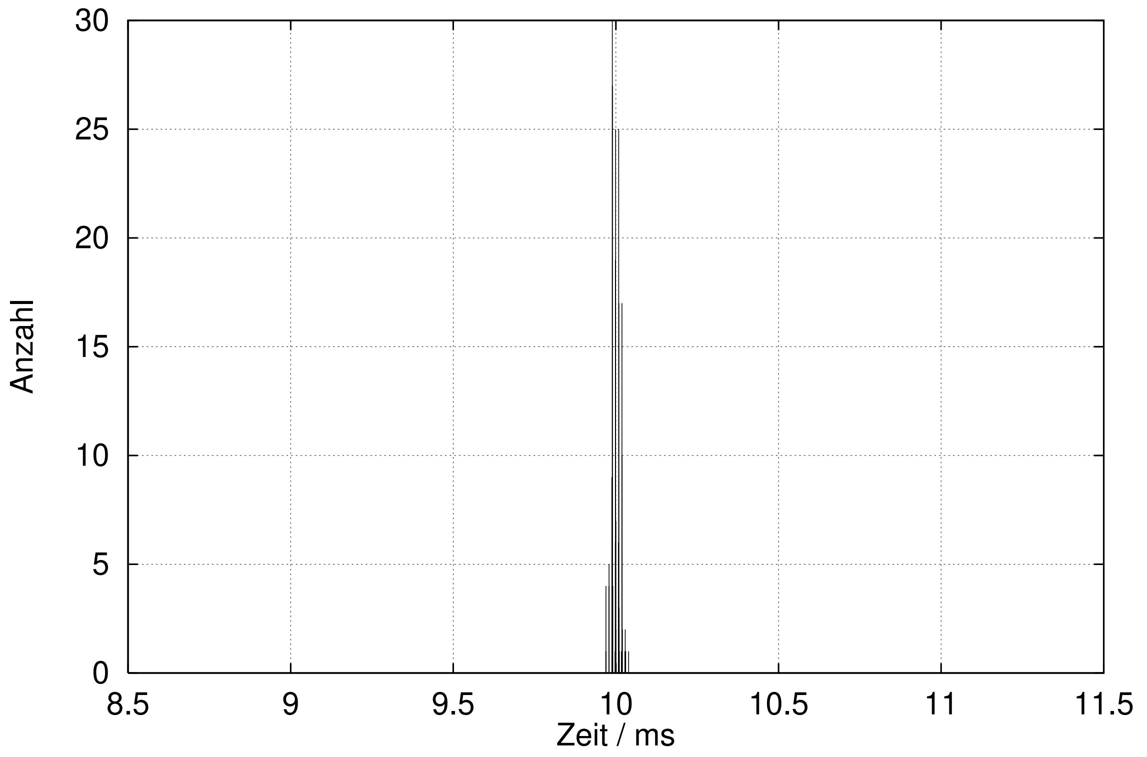


Abbildung 5.1: Jitter einer Nachricht (ID 400) im 10 ms Raster ohne äußere Buslast

Die folgenden 3 Messungen wurden dazu mehrmals mit sehr ähnlichen Ergebnissen durchgeführt:

Messung 1: Signaldrift (Jitter) ohne Buslast, d.h. ohne Störung durch andere Knoten im Netzwerk. Der mittlere Nachrichtenabstand beträgt 10 ms mit einer Standardabweichung von 0,0127 ms (s. Abb. 5.1).

Messung 2: Jitters bei ca. 10% zusätzlicher Buslast mit sehr niedriger Priorität (ID 1792). Der mittlere Nachrichtenabstand beträgt 10 ms mit einer Standardabweichung von 0,14766 ms (s. Abb. 5.2).

Messung 3: Jitters bei ca. 10% zusätzlicher Buslast mit sehr hoher Priorität (ID 128). Der mittlere Nachrichtenabstand beträgt 10 ms mit einer Standardabweichung von 0,28848 ms (s. Abb. 5.3).

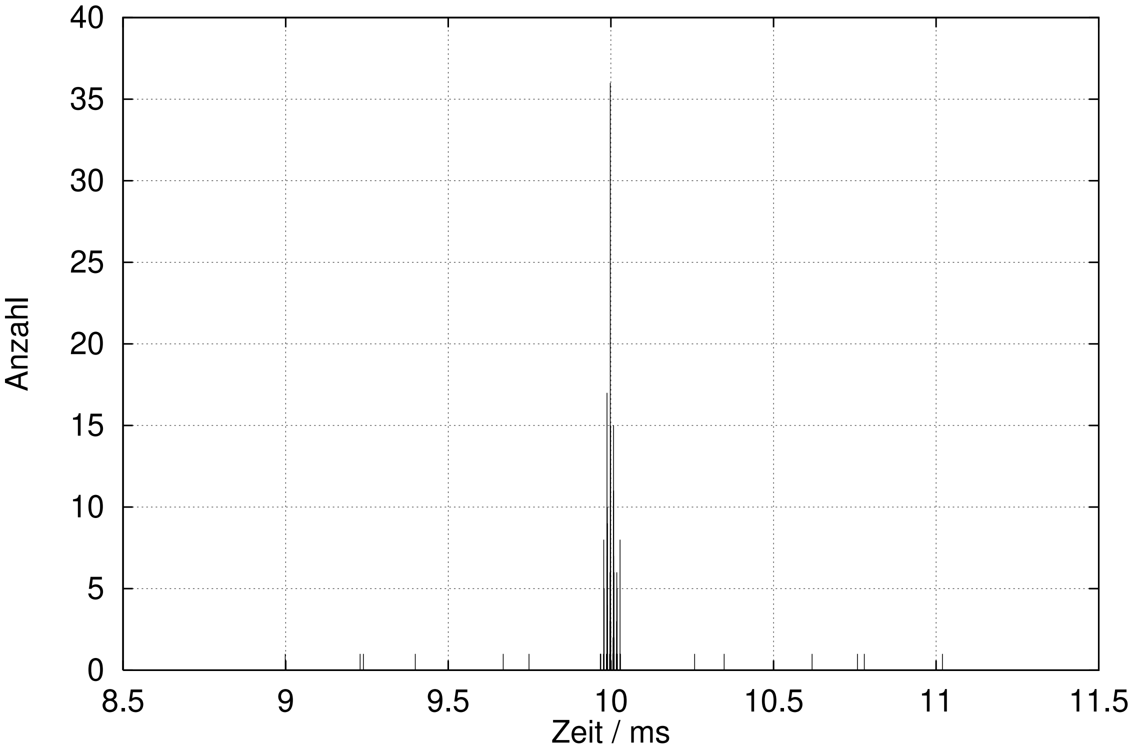


Abbildung 5.2: Jitter einer Nachricht (ID 400) im 10 ms Raster mit zusätzlicher Buslast (ID 1792)

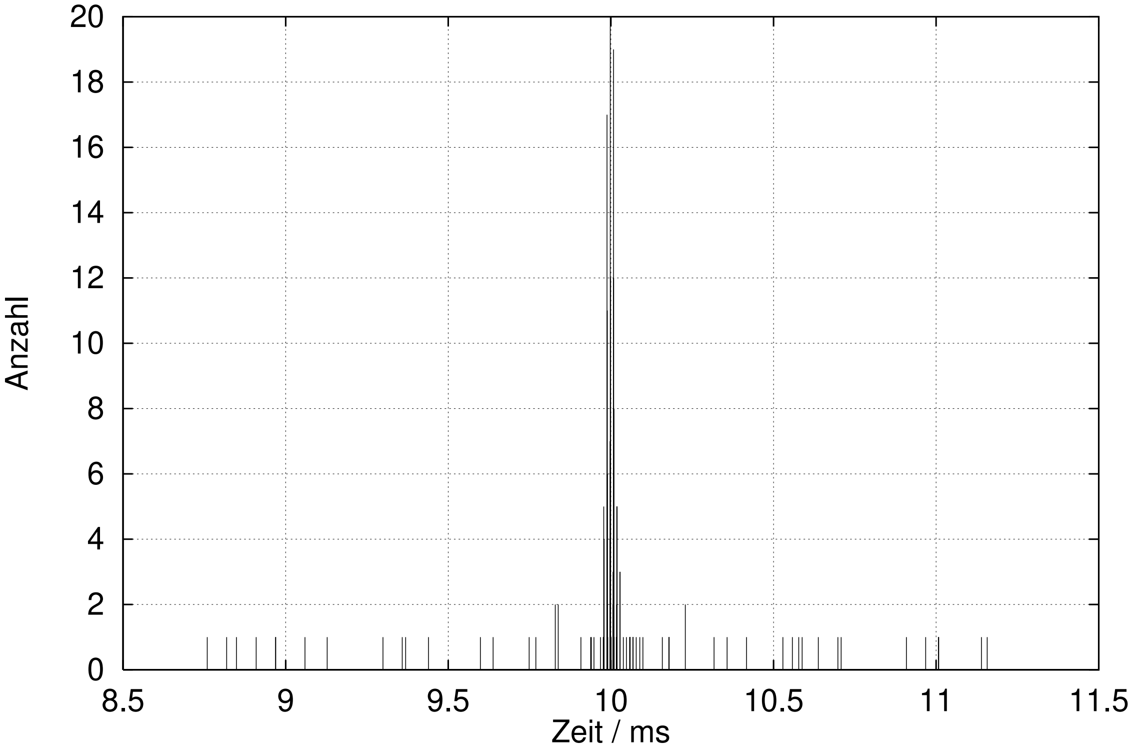


Abbildung 5.3: Jitter einer Nachricht (ID 400) im 10 ms Raster mit zusätzlicher Buslast (ID 128)

5.4 Bewertung der Ergebnisse

Das Ergebnis der Messungen bestätigt die analytischen Berechnungen aus Kap. 5.1. Der Jitter innerhalb der einzelnen Blöcke in der ersten Messung entsteht durch das Auflösungsvermögen des Timers. Der Jitter der einzelnen Blöcke zueinander entsteht durch leichte Verzögerungen bei dem Zugriff auf den Bus, der direkt die Taktrate des Busses von 100 kHz wiederspiegelt. Die 2. Messung zeigt deutlich, daß die „Störung“ vom Bus vertrieben werden kann. Es treten maximale Verzögerungen von 1 ms auf, die der Übertragungsdauer einer Störnachricht entsprechen. In der 3. Messung treten auch Verzögerungen > 1 ms auf. Hier konnte sich die Störnachricht gegenüber der Meßnachricht durchsetzen. Bei einer höheren Buslast wäre dieser Effekt noch deutlicher sichtbar. Leider war es mit dem PC nicht möglich, eine höhere Buslast zu generieren.

Deutlich sichtbar ist auch, daß sich bei den gestörten Nachrichten keine Gaußverteilung ergibt. Dies liegt daran, daß der Sendewunsch der Meßnachricht zu jedem Zeitpunkt der Störnachricht gleichwahrscheinlich ist. Die Rechnung (5.2) setzte voraus, daß die Zeitpunkte des Eintreffens der Störnachricht gleichverteilt sind. Die Schlußfolgerung aus der Rechnung, daß auch der Jitter gleichverteilt ist, wird durch die Messung bestätigt.

6 Entwicklung von Sensoren und die Einbindung in CAN

Abb. 6.1 gibt einen Überblick über die für die CAN-Wetterstation benötigten Meßgrößen, deren Erfassung, Signalverarbeitung und Datenübertragung.

Im folgenden wird zunächst eine Einführung über die für diese Arbeit benötigten Sensorprinzipien gegeben. Dabei erhebt diese Übersicht keinen Anspruch auf Vollständigkeit. Vertiefende Informationen können der Literatur aus *Schaumburg* [7] und *Werthschützky* [11] entnommen werden.

6.1 Sensorübersicht

Für die CAN-Wetterstation werden die drei Meßgrößen Druck, Temperatur und Feuchte benötigt. Folgende Prinzipien sind für die jeweilige Meßgröße bekannt:

Druck

Drucksensoren können im wesentlichen in vier verschiedene Wirkprinzipien aufgeteilt werden:

- Resistive Drucksensoren
- Piezoresistive Drucksensoren

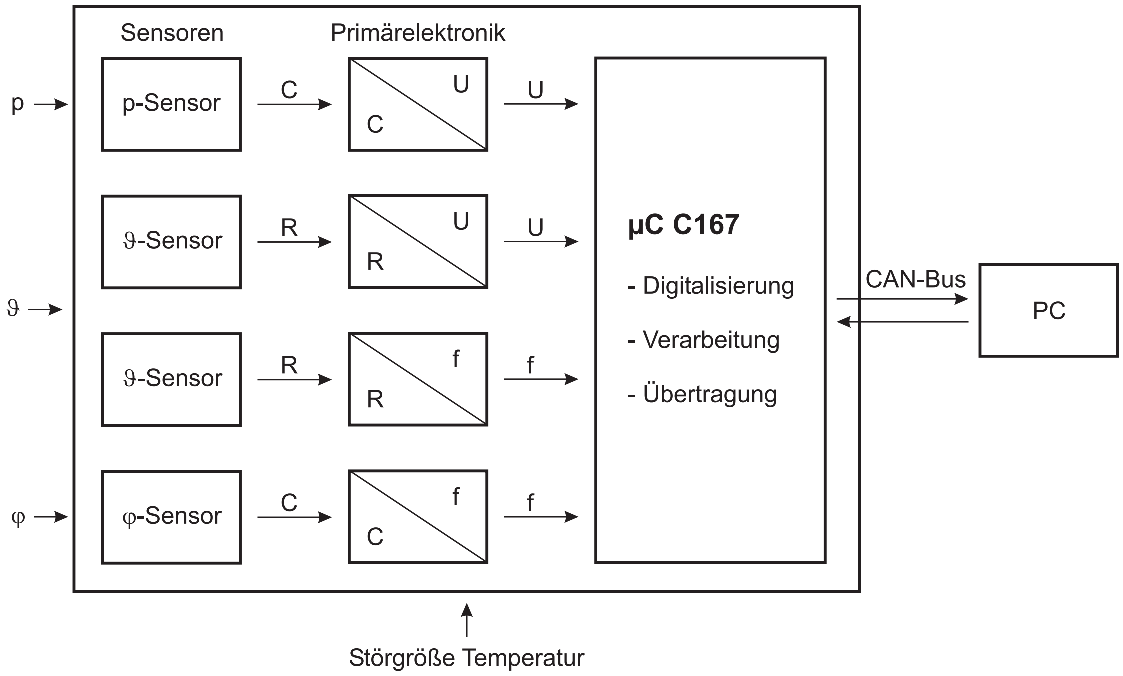


Abbildung 6.1: Überblick über die Ankopplung der Sensoren

- Piezoelektrische Drucksensoren
- Induktive Drucksensoren
- Kapazitive Drucksensoren

Der für die Messung des Umgebungsdrucks interessante Meßbereich liegt zwischen 870 und 1100 mbar. Hierfür eignen sich die kapazitiven Drucksensoren sehr gut, die es bereits mit Auswerte- und Kompensationselektronik gibt. Mit dieser Elektronik liegt der Meßfehler unterhalb 0,4 mbar. Mit einer zusätzlichen Linearisierung können die Meßfehler unterhalb 0,04 mbar gebracht werden.

Temperatur

Temperatursensoren sind in vielen Varianten am Markt eingeführt und behaupten sich in den unterschiedlichsten Einsatzgebieten. Dabei kommen im wesentlichen zum Einsatz:

- Thermoelektrische Temperatursensoren
- Resistive Temperatursensoren
- Andere Temperatursensoren, wie Quarze, Transistoren, etc.

Für die Messung der Umgebungstemperatur sind die resistiven Temperatursensoren interessant, da sie im Bereich von -40° C bis +85° C eine gute Empfindlichkeit besitzen. Man unterscheidet weiter zwischen Positive Temperature Coefficient (PTC) und Negative Temperature Coefficient (NTC) Sensoren. Problematisch ist besonders bei den NTC die starke Nichtlinearität. Diese muß z.B. durch Widerstandsnetzwerke und Rechnung kompensiert werden.

Relative Feuchte

- Resistive Feuchtesensoren
- Kapazitive Feuchtesensoren

Hier ist den kapazitiven Feuchtesensoren der Vorzug zu geben. Bei resistiven Feuchtesensoren kommen poröse Keramiken zum Einsatz, die ohne Reinigung eine schlechte Langzeitstabilität haben. Die meßbare relative Feuchte liegt im Bereich von 0 bis 100%. Ohne Kompensation bleibt der Fehler bei kapazitiven Sensoren unterhalb 1,5% relativer Feuchte. Allerdings muß auf jeden Fall eine Kompensation der Temperaturabhängigkeit vorgesehen werden.

6.2 Eingesetzte Sensoren

Die hier dokumentierten Sensoren wurden nicht explizit ausgewählt, sondern aufgrund von Vorgaben bzw. dem Vorhandensein im Institut eingesetzt.

Feuchtesensoren (kapazitiv)				Primär-elek-tronik
Sensor	Grundkapazität	Kapazitäts-änderung	Kapazitäts-bereich	
Mela FE09/4	115±15 pF bei 10% r.F.	0,27 pF/% r.F.	112 .. 140 pF	C → f
E+E HC 101	200±20 pF bei 76% r.F.	0,56 pF/% r.F.	157 .. 213 pF	C → f
Drucksensor (kapazitiv)				
Sensor	Spannungsbereich			
Ceracore C1	0,5 .. 4,5 V bei 800 .. 1200 mbar			C → U
Temperatursensor (resistiv)				
Sensor	Widerstandsbereich			
KTY10-6	1,1 .. 3 kΩ bei -40 .. +85° C			R → U
NTC 68k	2500 .. 6,5 kΩ bei -40 .. +85° C			R → f

Tabelle 6.1: Übersicht der verwendeten Sensoren

6.3 Digitalisierung

Zur weiteren Signalverarbeitung müssen die analogen Meßgrößen i.d.R. digitalisiert werden. Dazu bietet der μC C167 von Siemens (s. Kap. A6) folgende Möglichkeiten:

- A/D-Wandlung durch bis zu 16 Eingänge mit jeweils 10-Bit
- Frequenzanaloge Messung (Periodendauermessung) unter Einsatz der hochauflösenden Timer und der bis zu 8 Fast External Interrupts mit Flankenerkennung

Nachfolgend wird untersucht, wofür die jeweilige Methode geeignet ist.

6.3.1 A/D-Wandlung

Die A/D-Wandlung ist die klassische Methode der Meßwerterfassung. Die Eingänge des C167 haben bei einem Spannunghub von 5 V eine Auflösung von 4,88 mV ±2,44 mV. Das entspricht einem Fehler von 0,1 % aufgrund der Quantisierung des A/D-Wandlers. Neben dem Quantisierungsfehler gibt es beim A/D-Wandler noch Offset- und Verstärkungsfehler, die allerdings leicht zu kompensieren sind. Viel problematischer und nicht kompensierbar sind Fehler durch totale und differentiale Nichtlinearitäten. Weiterhin entstehen Fehler durch Temperaturdrift und dynamische Fehler, verursacht durch das „Sample and Hold“ Glied. Letzter Fehler wird auch Apertur-Jitter genannt. Für eine tiefgreifendere Behandlung der A/D-Wandler Fehler empfiehlt sich das Buch Halbleiter-Schaltungstechnik von *Tietze und Schenk* [10].

Für die Messung ist i.d.R. der Pegel der Meßgröße an den Pegel des A/D-Wandlers anzupassen. Dies ist notwendig, um die volle Auflösung des A/D-Wandlers nutzen zu können. Die Pegelanpassung an den Drucksensor ist in Abb. A7.1 und die an den Temperatursensor in Abb. A7.2 dargestellt.

6.3.2 Periodendauermessung

Einen völlig anderen Weg kann man über die Periodendauer gehen. Schwingkreise sind sehr einfach z.B. über Schmitt-Trigger (s. Abb. A7.3) oder den C555 realisierbar. Es lassen sich sowohl kapazitive wie auch resistive Elemente als Sensor einsetzen. Die Messung der Pulsbreite kann im C167 mittels eines hochgenauen Timers und der External Interrupts mit Flankenerkennung implementiert werden. Neben dem Quantisierungsfehler, der der Timerauflösung entspricht, sind bei der Messung keine nennenswerten Fehler vorhanden. Sehr viel Aufwand muß allerdings in den Aufbau des Schwingkreises investiert werden, da der Meßfehler damit direkt beeinflußt wird. Hier treten auch entsprechende Abweichungen durch Temperaturänderungen auf, die nur schwer beherrschbar sind.

In der entwickelten Meßsoftware wird eine Timerauflösung von 1,6 μs verwendet. Außerdem wird, um einzelne Ausreißer in der Messung zu eliminieren, über 100 Flanken gemessen und gemittelt. Damit lassen sich Periodendauern zwischen 4,54 μs und 1,05 ms messen. Die untere Periodendauer ist begrenzt durch die Timerauflösung und die Laufzeit der entsprechenden Programmteile. Die obere Periodendauer ist durch den verwendeten Variablentyp limitiert und läßt sich problemlos weiter ausbauen. Die Frequenzmessung wurde mit einem HP Frequenzgenerator bei einer Frequenz von 4 kHz verifiziert. Die gleichzeitige Messung mit einem HP Gigaherz-Zähler lieferte eine stabile Frequenz von 4 kHz +0,01 Hz. Die Ausgaben der Keil C167 Messung pendelten zwischen den beiden diskreten Werten 3,99974 kHz und 4 kHz. Der Meßfehler beträgt also bei einer Periodendauer von 0,25 ms 0,0065 %. Dies entspricht exakt 1/100 der Timerauflösung.

Messungen mit den entwickelten Schmitt-Trigger Schwingkreisen weisen einen deutlich höheren Fehler auf, der sich in der Standardabweichung s wiederspiegelt. Während der Messung von ca. 10 s diente eine Blechschachtel über der Schaltung und den Sensoren als Abschirmung und sorgte für konstante Verhältnisse. Da dieser

Fehler auch bei der Messung mit dem Styroflex Kondensator und dem NTC auftritt, ist ein Rauschen der Feuchtesensoren auszuschließen. Stattdessen sind kleine Unterschiede in der Schaltschwelle der Schmitt-Trigger Ursache für diese Streuung.

Sensor	$\overline{T}$ / μs	s / μs	ΔT_{max} / μs (berechnet)	s_{rel} / %	Abbildung
Mela	49,085	0.03762	10,9	0,345	6.2
E+E	47,3596	0,06989	14,3	0,489	6.3
NTC	40,3924	0,0307	30,3	0,101	6.4

Tabelle 6.2: Sensormeßwerte (Periodendauern) und Standardabweichungen bei Umgebungsbedingungen

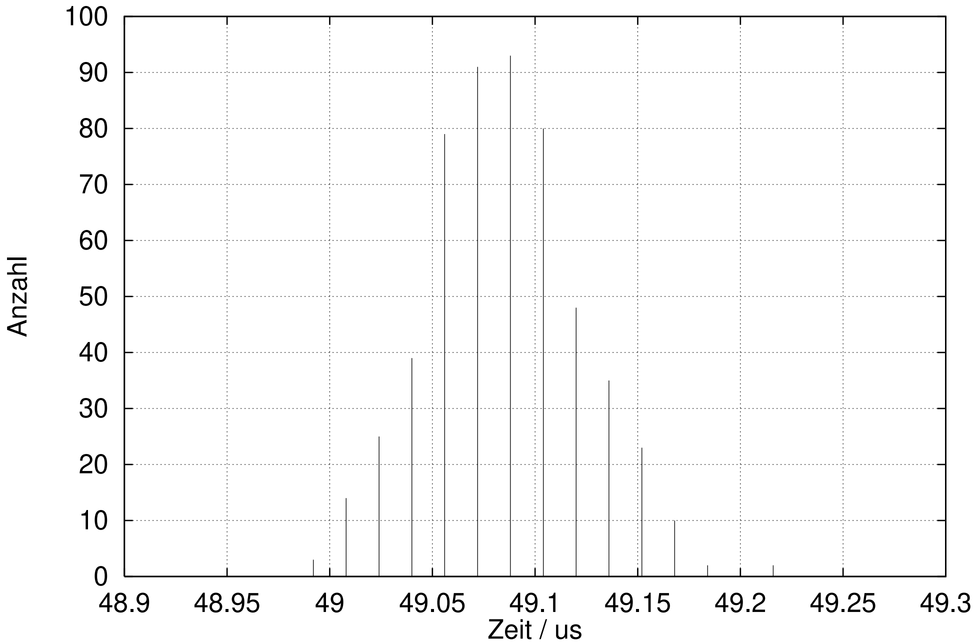


Abbildung 6.2: Sensor 1

6.3.3 Gegenüberstellung

Bei beiden Meßprinzipen gibt es Vor- und Nachteile. So gibt es vom Anwendungsfall abhängige Einsatzgründe. Ein kapazitives Element ist nur über einen Frequenzwandler oder in einem Dual-Slope-Wandler auswertbar. Eine Spannung mittels U-f Umsetzer zu messen, macht auch in den wenigsten Fällen Sinn. In der minimalen Auflösung unterscheiden sich beide Systeme kaum. Wohl hingegen bei der maximalen Auflösung. Diese ist bei der Periodendauermessung nahezu unbegrenzt (eine Schranke setzt nur die Speichergröße für den Zähler).

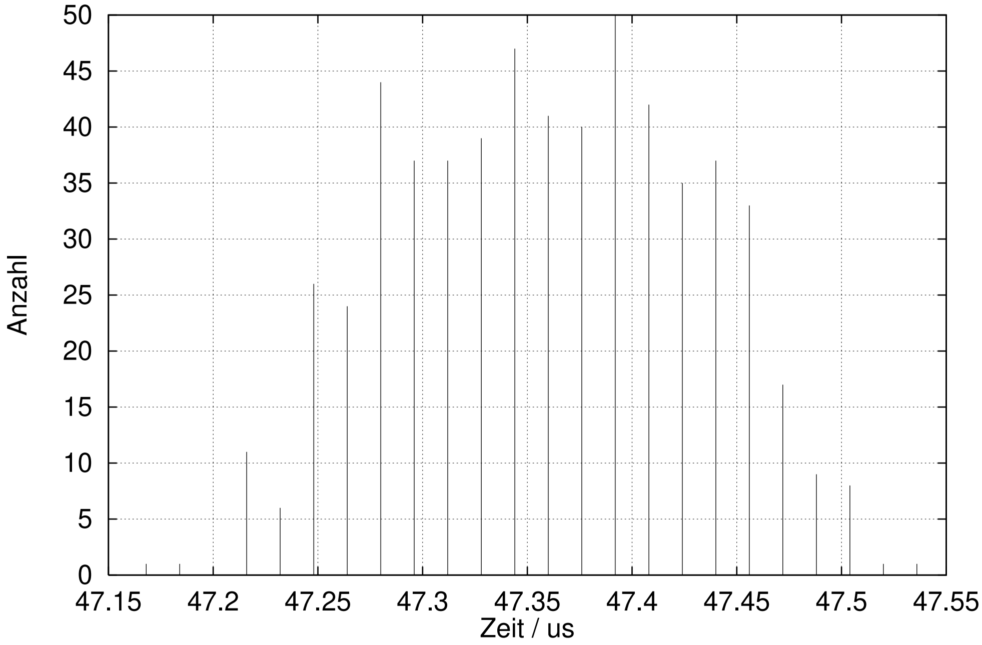


Abbildung 6.3: Sensor 2

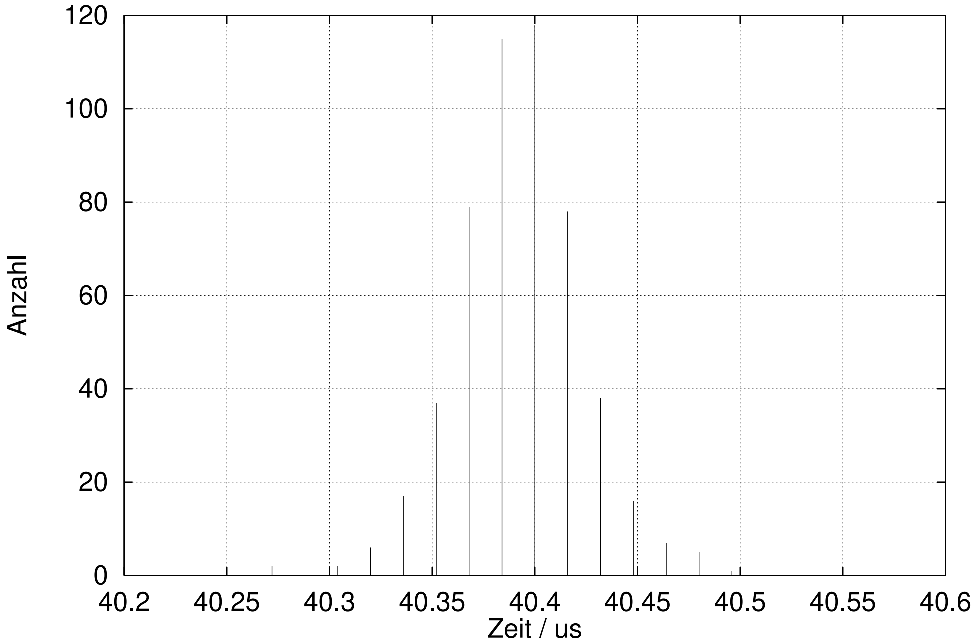


Abbildung 6.4: Sensor 3

6.4 Linearisierung der Sensoren

Druck

Die Pegelanpassung ist so eingestellt, daß eine Verstärkung von 9,929 und Offset 4,25 V errricht wird. U_p ist die Spannung des Drucksensors, U_{OP} ist die gemessene Spannung am Ausgang des Operationsverstärkers. Die Umrechnung der Verstärkerparameter liefert:

$$\begin{aligned} U_{OP} &= \left[U_p + \frac{R_3}{R_3 + R_5}(5\ \mathrm{V} - U_p) - 4,25\ \mathrm{V}\right] \cdot (1 + \frac{50\ \mathrm{k\Omega}}{R_1}) \\ &= (0,213U_p - 0,313\ \mathrm{V}) \cdot 9,929 \end{aligned} \tag{6.1}$$

$$\rightsquigarrow U_p = 0,473U_{OP} + 1,469 \tag{6.2}$$

Das Ausgleichspolynom aus Abb. 6.5 bestimmt sich zu

$$p = -0,0376U_p^3 + 0,2835U_p^2 + 9,4427U_p + 75,299 \tag{6.3}$$

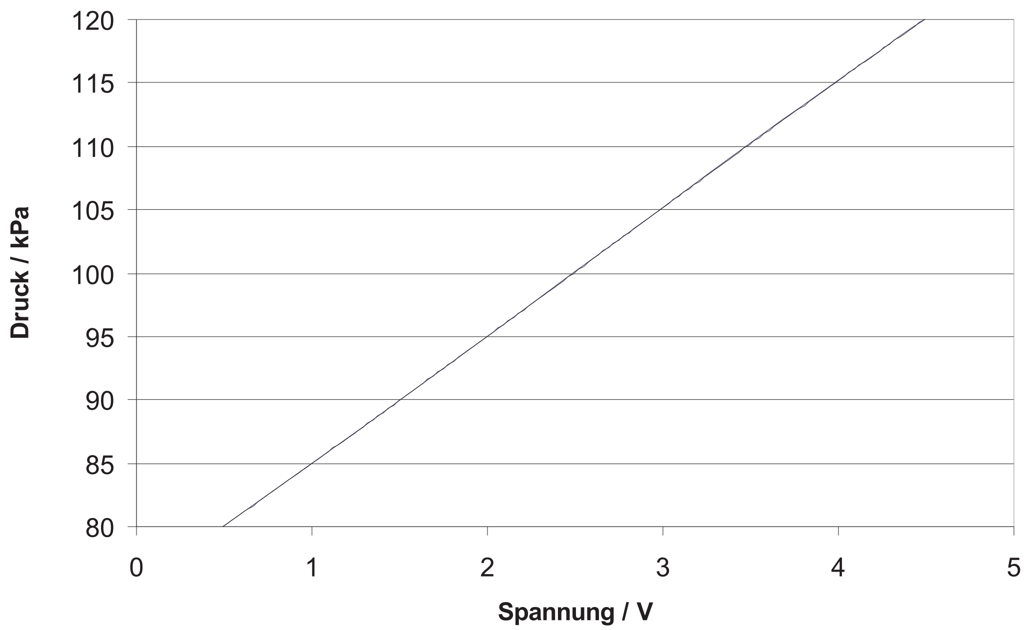


Abbildung 6.5: Linearisierung des Drucksensors und Umwandlung Spannung in Druck

Temperatur

Der KTY10-6 Sensor ist nahezu linear. Eine Messung zur Bestimmung des Ausgleichspolynoms ist nicht durchgeführt worden.

Der NTC 68k kann durch die Parallelschaltung eines 100 kΩ Widerstands und einer Reihenschaltung mit einem 82 kΩ Widerstand das logarithmische

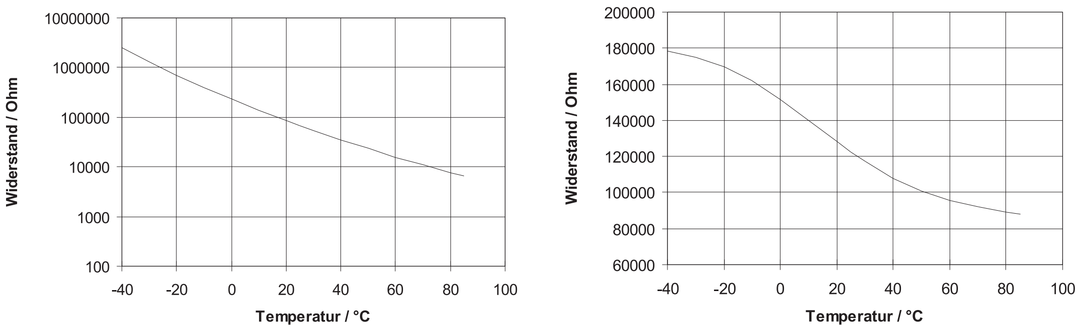


Abbildung 6.6: NTC-Kennlinie und Linearisierung durch Widerstandsnetzwerk

Verhalten genommen werden (s. Abb. 6.6). Eine weitere rechnerische Linearisierung ist dennoch notwendig.

Relative Feuchte

Die Feuchtesensoren verhalten sich bei konstanter Temperatur nahezu linear. Um die Linearität zu gewährleisten, ist eine Temperaturkompensation vorzusehen. Ein Kalibrierlauf konnte, da die Referenzfeuchten nicht mehr rechtzeitig zur Verfügung standen, nicht durchgeführt werden.

7 Kritik und Ausblick

Die Arbeit gliederte sich in zwei Aufgabenteile: Die Latenzzeitmessung und der Aufbau einer CAN-Wetterstation.

Im Bereich der Latenzzeitmessung wurde ein Application Layer Protokoll entwickelt, welches sich optimal für die Messung eignet. Weiterhin sind die Besonderheiten des μC C167 ausgenutzt worden, um eine hochpräzise Messung mit einer Auflösung von 0,4 μs zu gewährleisten. Die damit erzielten Messungen decken sich mit den analytischen Berechnungen. Es ist somit eine gute Basis für die Vorhersage der Verzögerung einer Nachricht auf dem CAN-Bus geschaffen. Dieses Wissen kann für praktische Anwendungen vorteilhaft eingesetzt werden. Besonders im Bereich der Echtzeitverarbeitung sind die Ergebnisse dieser Arbeit von höchster Bedeutung.

Der zweite Teil der Arbeit bezieht sich auf die CAN-Wetterstation. Hierzu werden die physikalischen Größen Druck, Temperatur und relative Feuchte erfaßt, verarbeitet, linearisiert und über den CAN-Bus an beliebige Knoten weitergeleitet. Die Erfassung der Sensorsignale erfolgt mit einem μC C167 und dessen spezifischer Möglichkeiten. Die Messung der Periodendauer stellt dabei eine unübliche aber sehr einfache und präzise Variante der Meßwerterfassung dar. Eine Kalibrierung der Sensoren konnte aus zeitlichen Engpässen mit dem Meßlabor nicht mehr durchgeführt werden. In anschließenden Arbeiten sollten entsprechende Kalibrierläufe durchgeführt werden. Aus den Ergebnissen sind geeignete Kompensationsfunktionen zu entwickeln, um die Sensoren abzugleichen und die Linearitäts- sowie Hysteresefehler zu eliminieren.

Literaturverzeichnis

[1] Bronstein, I.N.; Semendjajew, K.A.: Taschenbuch der Mathematik, 24. Auflage, BSB B.G. Teubner Verlagsgesellschaft, Leipzig und Verlag Nauka, Moskau 1979, Lizensausgabe für den Verlag Harri Deutsch, Thun 1989.

[2] CiA/DS 201 ... 207: CAN Application Layer (CAL) for Industrial Applications, Version 1.1, Februar 1996.

[3] CiA/DS 301: CANopen CAL-based Communication Profile for Industrial Systems, Version 3.0, Oktober 1996.

[4] Drexler, Th.: Einplatinenrechner zur Motorsteuerung mit Busschnittstelle, Diplomarbeit Nr. DA 1399, Technische Universität Darmstadt, Institut für EMK, 1998.

[5] ISO 11898, DIN 11898: Austausch digitaler Informationen; Steuergerätenetz (CAN) für schnellen Datenaustausch, 1994.

[6] Färber, G.: Feldbussysteme, Oberseminar Prozeßrechentechnik, Technische Universität München, Lehrstuhl für Prozeßrechner, `http://www.lpr.e-technik.tu-muenchen.de/courses/seminar/fieldbus/fieldbus.html`, 1994.

[7] Schaumburg, H.: Sensoren, 1. Auflage, Teubner Verlag, Stuttgart, 1992.

[8] Siemens AG: C167 Derivatives, User's Manual 03.96, Version 2.0, 1996.

[9] Tanenbaum, A.: Computer networks, 3. Auflage, Prentice Hall, 1996.

[10] Tietze, U.; Schenk, Ch.: Halbleiter-Schaltungstechnik, 7. Auflage, Springer Verlag, Berlin Heidelberg New York Tokyo, 1985.

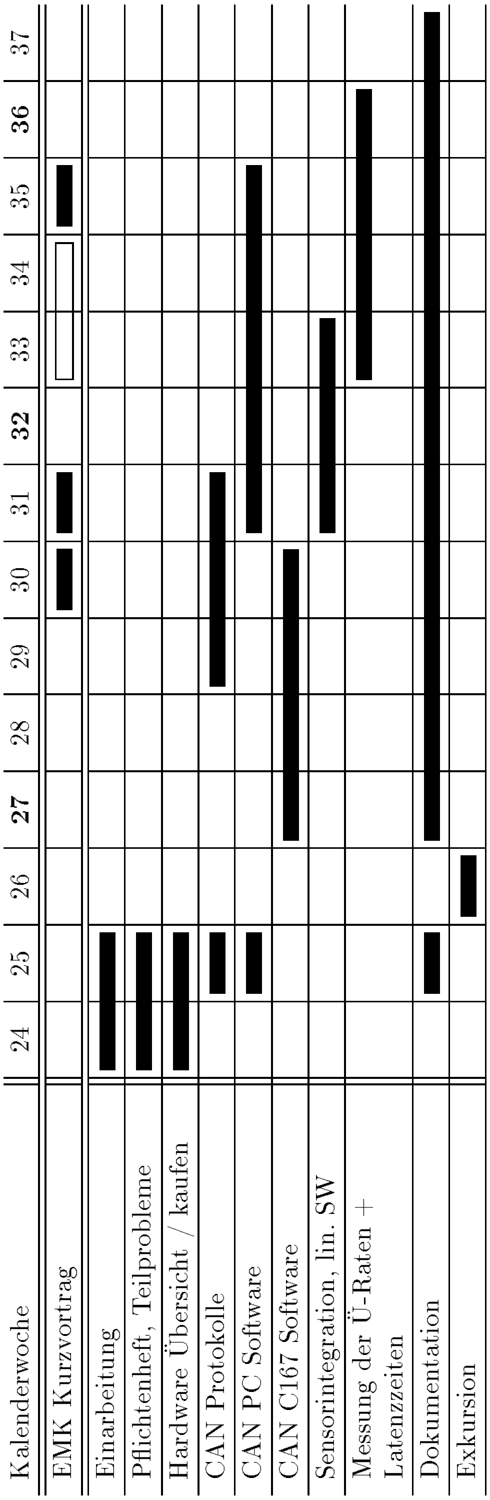
[11] Werthschützky, R.: Meßtechnik - elektrisches Messen mechanischer Größen, Band 1: Grundlagen der elektromechanischen Meßtechnik, Vorlesungsbegleitmaterial, Technische Universität Darmstadt, Institut für EMK, 1995.

[12] Wünsche, Th.: EMS Dr. Thomas Wünsche, Contoller Area Network: A Serial Bus System - Not Just For Vehicles, `http://www.ems-wuensche.com/techinfo/english/introduc/introduc.htm`, 1998.

In dieser Arbeit ist sehr viel mit dem Informationsmedium Internet gearbeitet worden. Deshalb sei an dieser Stelle auf das Kapitel A3 und besonders auf Kapitel A3.3 verwiesen. Dort befinden sich Links zu Web-Seiten, die zum Teil auch als Quelle für diese Arbeit verwendet wurden.

Anhang

A1 Zeitplan


Kalenderwoche
24
25
26
27
28
29
30
31
32
33
34
35
36
37
EMK Kurzvortrag
Einarbeitung
Pflichtenheft, Teilprobleme
Hardware Übersicht / kaufen
CAN Protokolle
CAN PC Software
CAN C167 Software
Sensorintegration, lin. SW
Messung der Ü-Raten + Latenzzeiten
Dokumentation
Exkursion

A2 Kostenaufstellung

Eigene Zeit	14 Wochen x 8 Stunden x 7 Tage à 20,-	15680,- DM
Betreuer	13 Wochen x 1 Stunde à 50,- DM	600,- DM
Co-Betreuer	7 Wochen x 5 Stunden à 50,- DM	1750,- DM
CAN Dongle	Fa. PEAK-Service Darmstadt	280,- DM
CAN PC SW	Fa. PEAK-Service Darmstadt 980,- -30%	686,- DM
CAL Profile	CiA Dokumentation	128,- DM
CANopen Profile	CiA Dokumentation	96,- DM
Bauteile		100,- DM
Summe		19320,- DM

A3 Kontaktadressen

A3.1 CAN Hardware

EMS Dr. Thomas Wuensche
Sonnenhang 3
D-85304 Ilmmuenster (Germany)
Tel: +49-8441/490260
FAX: +49-8441/81860
Email: marketing@ems-wuensche.com
http://www.ems-wuensche.com

PEAK-Service GmbH
Benzweg 4
D-64293 Darmstadt (Germany)
Tel: +49-6151/8173-0
FAX: +49-6151/8173-59
Email: Info@PEAK-Service.com
http://www.peak-service.com

PHYTEC Meßtechnik GmbH
Robert-Koch-Str. 39
55129 Mainz (Germany)
Tel: +49-6131/9221-0
FAX: +49-6131/9221-33
support@phytec.de
http://www.phytec.de

Vector Informatik GmbH
Friolzheimer Straße 6
D-70499 Stuttgart (Germany)
Tel: +49-711/139996-0
FAX: +49-711/139996-30
can@vector-informatik.de
http://www.vector-informatik.de

A3.2 CAN Protokolle

CAN in Automation (CiA) e.V.
International Headquarters
Am Weichselgarten 26
D-91058 Erlangen (Germany)
Tel: +49-9131/69086-0
FAX: +49-9131/69086-79
Email: headquarters@can-cia.de
http://www.can-cia.de

esd GmbH
Vahrenwalder Str. 205
30165 Hannover (Germany)
Tel: +49-511/372980
FAX: +49-511/633650
Email: sales@esd.h.eunet.de
http://www.esd-electronics.com

Janz Computer AG
Im Dörener Feld 8
D-33100 Paderborn (Germany)
Tel: +49-5251/1550-0
FAX: +49-5251/1550-90
Email: mail@janzag.de
http://www.janzag.de

SOFTING GmbH
Richard-Reitzner-Allee 6
D-85540 Haar (Germany)
Tel: +49-89/45656-0
FAX: +49-89/45656-399
Email: support.can@softing.com
www.softing.com

STA Reutlingen
Alteburgstr. 150
72762 Reutlingen (Germany)
Tel: +49-7121/271-327
FAX: +49-7121/25713
Email: sta@fh-reutlingen.de
http://www.fh-reutlingen.de/~www-sta

A3.3 CAN Informationen

Discussion lists
http://www.kvaser.se/can/info/discuss.htm

2nd International CAN Conference '95
http://www.docs.uu.se/~ken/CiA95_programme.html

Seminar „Feldbussysteme“ an der TU-München
http://www.lpr.e-technik.tu-muenchen.de/
courses/seminar/fieldbus/fieldbus.html

CAN: Controller Area Network
http://www.nrtt.demon.co.uk/can.html

CAN higher layer protocols
http://www.stzp.de/papers/icc97/icc97_e.html

The Robotics and Automation Group
http://www.ncl.ac.uk/~nrauto/canopen.htm

Worldwide Status of CAN - Present and Future
http://www.fh-wolfenbuettel.de/fb/i/institute/ifvs/CAN/canfund.htm

A3.4 C167 Compiler

HighTec EDV-Systeme GmbH
Feldmannstr. 98
D-66119 Saarbrücken (Germany)
Tel: +49-681/92613-0
FAX. +49-681/92613-26
htc@hightec-rt.com
http://www.hightec-rt.com

Keil Elektronik GmbH
Bretonischer Ring 15
85630 Grasbrunn (Germany)
Tel: +49-89/456040-0
FAX: +49-89/468162
support.intl@keil.com
http://www.keil.com

Siemens AG
Halbleiter Distribution
Richard-Strauss-Straße 76
D-81679 München (Germany)
Tel: +49-89/9221-4086
FAX: +49-89/9221-2071
http://www.siemens.de/semiconductor

TASKING GmbH
Brennerstr. 5
D-71229 Leonberg (Germany)
Tel: +49-7152/97991-0
FAX: +49-7152/97991-20
support_de@tasking.com
http://www.tasking.com/german.htm

A4 Das OSI Modell

In der ISO 7498 Norm ist der Kommunikationsstandard für das Open Systems Interconnection (OSI) Modell definiert. Dieses Modell beschreibt, wie ein Computer mit einem beliebigen Netzwerk kommunizieren sollte. Das OSI Modell setzt sich aus einem Protokoll-Stack mit 7 unabhängigen Layern (Ebenen), wie er in Abb. A4.1 dargestellt ist, zusammen.

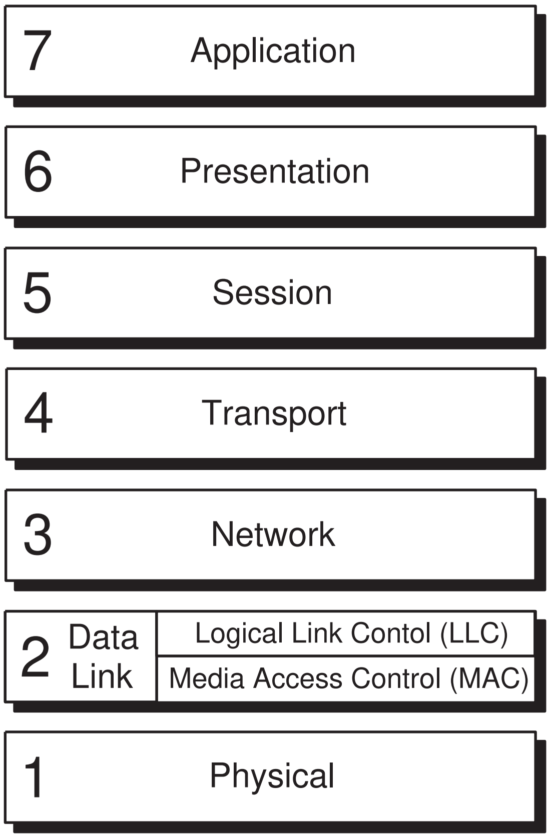


Abbildung A4.1: Die 7 Layer des OSI Modells

Die Layer haben dabei die folgende Bedeutung:

- Der „Application Layer“ (7) ist die höchste Ebene und stellt das User-Interface zu allen anderen niederen Layern zur Verfügung.
- Der „Presentation Layer“ (6) übernimmt die Daten-Formatierung und Code-Transformation, um die Kommunikation zwischen verschiedenen Daten-Formaten zu ermöglichen. Dies ist bei einem CAN in der Regel nicht notwendig, da die Daten-Formate innerhalb eines Netzwerks konsistent sind.
- Der „Session Layer“ (5) leitet die Koordination während einer Sitzung (session).
- Der „Transport Layer“ (4) kontrolliert die Reihenfolge von Message-Komponenten.

- Der „Network Layer“ (3) setzt die Adressen und liefert Message-Pakete.
- Der „Data-Link Layer“ (2) arbeitet mit der Message auf Bit und Byte Ebene und stellt die Verbindung zwischen dem Node und dem Netzwerk her.
- Der „Physical Layer“ (1) bezieht sich auf die Hardware-Komponenten des Busses. Er sendet und empfängt Bit-Streams des Netzwerks.

Für tiefgreifendere Literatur zum Thema „OSI Modell“ empfiehlt sich das Buch „Computer networks“ von *Tanenbaum* [9].

Als ein Beispiel, wie das OSI Modell in einem CAN-System verwendet wird, kann man von einem ganz einfachen Netzwerk mit 2 Nodes ausgehen. An dem einen Node befindet sich ein Drucksensor und an dem anderen Node ein μ-Controller der den Druck überwacht und ggf. nachregelt. Der Drucksensor erzeugt auf dem „Application Layer“ eine Nachricht der Form „Druck am Sensor X mit dem Wert Y“. Diese Information wandert die OSI Layer nach unten, so daß die notwendigen Informationen in einen CAN Message Frame kodiert werden. Dabei wird dem Sensor „X“ ein entsprechender Identifier und Daten entsprechend dem Wert „Y“ zugeordnet. Damit ist die Nachricht kodiert und kann über den Bus in Form von elektrischen Signalen übertragen werden. An dem μ-Controller wird der umgekehrte Weg gegangen bis die Nachricht auch dort den „Application Layer“ erreicht hat und entsprechende Aktionen ausgeführt werden können.

A4.1 CAN und das OSI Modell

In der ISO 11989 (CAN Spezifikation [5]) Norm sind nur die Layer 1 und 2 („Physical Layer“ und „Data-Link Layer“) definiert.

Der „Physical Layer“ spezifiziert die elektrischen Eigenschaften des Busses. Weiterhin wird die Hardware beschrieben, welche Zeichen einer Nachricht in elektrische Signale wandelt und umgekehrt.

Der „Data-Link Layer“ versteht bereits das Format einer Message. Dieser Layer kodiert und dekodiert zum oder vom „Physical Layer“. In CAN Controllern ist dieser Teil üblicherweise in Hardware realisiert. Aufgrund der Komplexität wird dieser Layer unterteilt in

- einen „Logical Link Control“ (LLC) Layer der die Kommunikation mit den höheren Layern des OSI Modells übernimmt und
- einen „Media Access Control“ (MAC) Layer der die Messages de-/kodiert. Der MAC Layer führt auch die Priorisierung bei der Busvergabe (zerstörungsfreie bitweise Arbitrierung), die Fehlererkennung und Behandlung sowie den Zugriff auf den „Physical Layer“ durch.

A5 Tabellen

Priority Level	ID	Message / Object
0	1 .. 220	Synchronisierung Notfälle (Fehler)
1	221 .. 440	Notfälle (Fehler) Netzwerk Timing (SYNC, time stamp) andere synch. Messages mit hoher Priorität
2	441 .. 660	PDO (synchron)
3	661 .. 880	PDO (asynchron)
4	881 .. 1100	PDO (asynchron)
5	1101 .. 1320	PDO (asynchron)
6	1321 .. 1540	SDO
7	1541 .. 1760	reserviert für SDO
-	1761 .. 2031	reserviert

Tabelle A5.1: CANopen CMS Prioritäts-Level

Bitrate	Buslänge	Bit-Zeit t_b
1 MBits/s	25 m	1 μs
800 KBit/s	50 m	1,25 μs
500 KBit/s	100 m	2 μs
250 KBit/s	250 m	4 μs
125 KBit/s	500 m	8 μs
50 KBit/s	1000 m	20 μs
20 KBit/s	2500 m	50 μs
10 KBit/s	5000 m	100 μs

Tabelle A5.2: Empfohlene Baudraten und max. Buslängen für CANopen Geräte

Kommando	ID	Länge	Inhalt	Beschreibung
`NMT`	0	1	`<Cmd>`	Netzwerkmanagement (s.u.)
`PREPARED`	1	1	`<HwrAdr>`	Client mit der Hardwareadresse `HwrAdr` bereit
`LOGDATA`	2	3	`<CntD> <tHi> <tLo>`	Übertragung der Meßdaten an den „CANmaster“
`TRFCHI`	128	1	`<CntT++>`	Buslast mit hoher Priorität
`SENDDATA`	400	1	`<CntD++>`	Meßdaten
`TRFCLO`	1792	1	`<CntT++>`	Buslast mit niedriger Priorität

Tabelle A5.3: Beschreibung der ID Vergabe des entwickelten Protokolls

`<Cmd>`	`<Data>`	Kommando	Beschreibung
`0x01`	-	`InitCmd`	Anforderung an die Clients sich `PREPARED` zu melden
`0x02`	-	`StartCmd`	Beginnen der Meßkommunikation
`0x03`	-	`StopCmd`	Beenden der Meßkommunikation

Tabelle A5.4: Kommandosequenz der entwickelten NMT Nachricht des CANmaster

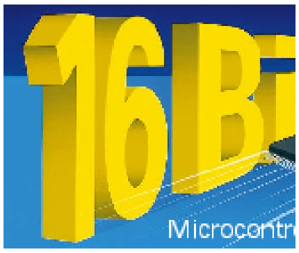
A6 Datenblätter

C167

Alle Informationen über den μC C167 von Siemens sind im Internet unter `http://www.siemens.de/semiconductor/products/ics/34/c167.htm` erhältlich.

SIEMENS

C167CR
Highly Integrated Microcontroller with On-chip CAN-Module



The C167CR* is a new derivative of the Siemens C166 family of full featured single-chip CMOS micro-controllers. It combines high CPU performance with high peripheral functionality and enhanced IO-capabilities.

The C167CR features a CAN module which meets version 2.0 B active of the Controller Area Network specification and was designed to fulfill the requirements of coming generations of automotive and industrial control applications.

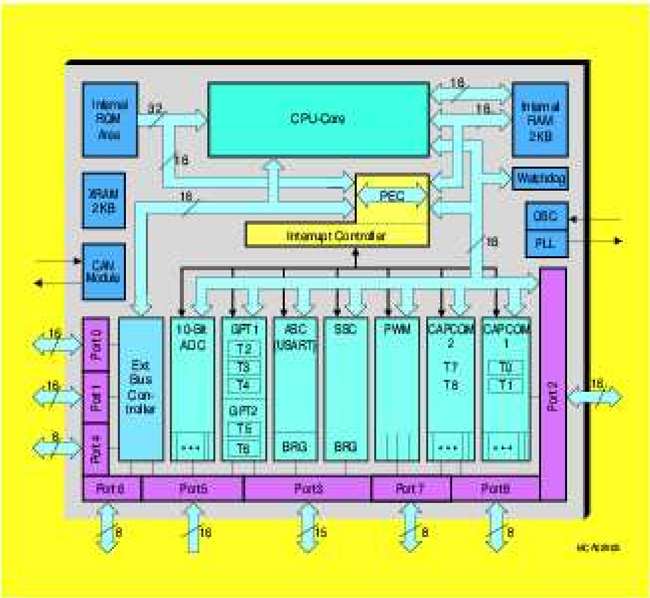
Device	ROM
C167CR-LM	-
C167CR-4RM	32 KB
C167CR-16RM	128 KB

- High Performance 16-bit CPU with 4-Stage Pipeline
- Versions with 25 MHz available
- 80 ns Instruction Cycle Time at 25 MHz CPU Clock
- 400 ns Multiplication (16 x 16 bit), 800 ns Division (32/16 bit) at 25 MHz CPU Clock
- Enhanced Boolean Bit Manipulation Facilities
- Additional Instructions to Support HLL and Operating Systems
- Register-Based Design with Multiple Variable Register Banks
- Single-Cycle Context Switching Support
- Clock Generation via on-chip PLL or via direct clock-input
- Up to 16 MBytes Linear Address Space for Code and Data
- 4 KBytes On-Chip SRAM (2 KB Internal RAM, 2 KBytes Extension RAM)
- Programmable External Bus Characteristics for Different Address Ranges
- 8-bit or 16-bit External Data Bus
- Multiplexed or Demultiplexed External Address/Data Buses
- Five Programmable Chip-Select Signals
- Hold- and Hold-Acknowledge Bus Arbitration Support
- 1024 Bytes On-Chip Special Function Register Area
- Idle and Power Down Modes
- 8-Channel Interrupt-Driven Single-Cycle Data Transfer Facilities via Peripheral Event Controller (PEC)
- 16-Priority-Level Interrupt System with 56 Sources, Sample-Rate down to 40 ns
- 16-Channel 10-bit A/D Converter with 9.7 µs Conversion Time
- Two 16-Channel Capture/Compare Units
- 4-Channel PWM Unit
- Two Multi-Functional General Purpose Timer Units with five 16-bit Timers
- Two Serial Channels (Synchronous/Asynchronous and High-Speed-Synchronous, 6 MBit/s at 25 MHz)
- On-Chip CAN Interface 2.0 B active with 15 Message Objects (Full-CAN/Basic-CAN)
- Programmable Watchdog Timer
- Up to 111 General Purpose IO Lines, partly with Selectable Input Thresholds and Hysteresis
- Supported by a Wealth of Development Tools like C-Compilers, Macro-Assembler Packages, Realtime Operating Systems, Emulators, Evaluation Boards, HLL-Debuggers, Simulators, Logic Analyzer Disassemblers, Programming Boards
- On-Chip Bootstrap Loader
- 144-Pin MQFP Package
- Full Automotive Temperature Range −40 °C to 125 °C

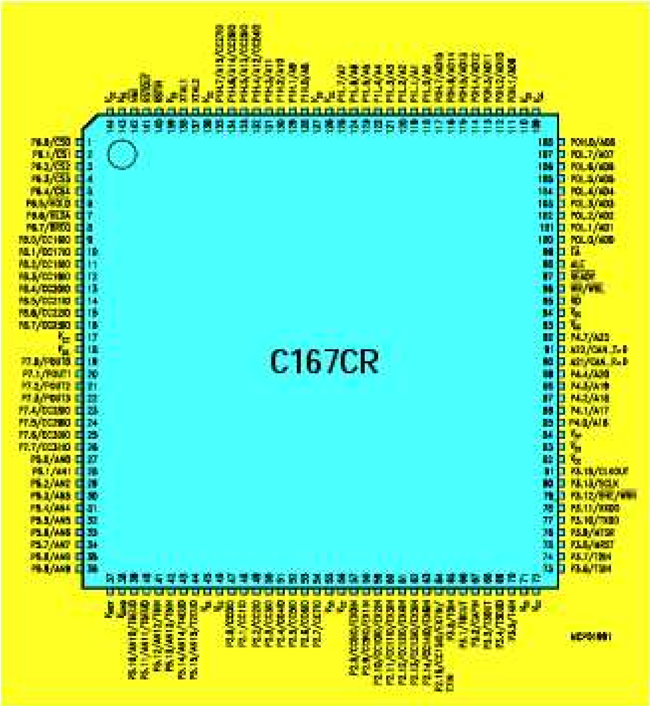
Controller Area Network (CAN): License of Robert Bosch GmbH

* For complete device designations corresponding to PRO ELECTRO please refer to the data sheet..

C167CR
Block Diagram



C167CR
Pin Configuration



Published by Semiconductor Group

Global PartnerChip for Systems on Silicon

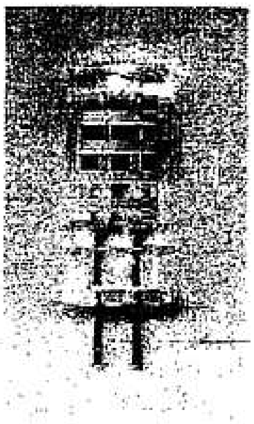
Siemens Aktiengesellschaft

Feuchtesensor E+E HC 101

HC 101

FEUCHTE SENSOR ELEMENT

Langjährige Produktionserfahrung im Bereich Feuchtesensorik ermöglichte die Entwicklung des Feuchtesensors HC 101.
Mit diesem Element werden neue Maßstäbe bezüglich Linearität, Hysterese, Aufsättigungsverhalten und Langzeitstabilität gesetzt.
Die geringe Nichtlinearität der Sensorcharakteristik von < 1,5 % r. F. im Bereich 0 – 98 % r. F. bringt dem Anwender ein wesentliches Einsparungspotential im Bereich Elektronik und Kalibration.
Für die einfache Handhabung und Montage ist das Element in ein Polycarbonat-Gehäuse eingebaut, welches auch die Möglichkeit einer direkten Leiterplattenmontage bietet.
Mit dem HC 101 offeriert E+E Elektronik einen hochqualitativen Feuchtesensor für den Bereich Meßtechnik, Automobil-, und Klimatechnik.



TECHNISCHE DATEN

Einsatzbereich	Feuchte 0 ... 100 % r. F. Temperatur -40 ... +120°C
Nennkapazität C_{76} bei 23°C	200 ± 20 pF
Empfindlichkeit	0,56 pF / % r.F.
Linearitätsfehler (0 ... 98 % r. F.)	< ± 1,5 % r. F.
Hysterese	< 1,5 % r. F.
Ansprechzeit t_{90}	< 6 sec
Temperaturquerempfindlichkeit	T>0°C d_c=-0,001661*r.F.*(T-30°C) T≤0°C d_c=-0,001661*r.F.*(0-30°C)
zulässige Spannung	5 V max (U_{SS}) max. DC Anteil: < 20 mV
Meßfrequenz	10 ... 150 kHz empfohlen 20 kHz

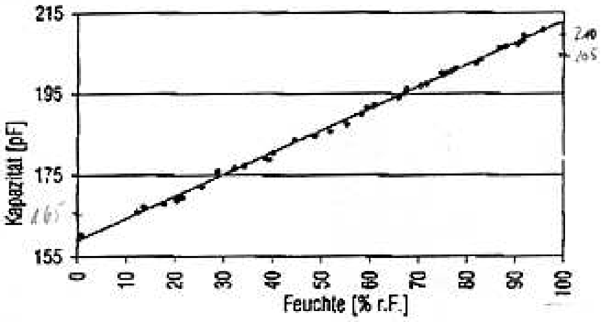
EIGENSCHAFTEN

- Einsatzbereich 0 – 100 % r. F.
- Linearitätsfehler < 1,5 % r. F.
- Betaubar
- Langzeitstabil
- Sehr gute Beständigkeit gegen Schadstoffe

SENSORCHARAKTERISTIK

Die Sensorkapazität steigt linear mit einem Kapazitätshub von etwa 55 pF über den gesamten Meßbereich an.
Im Feuchtebereich zwischen 0 – 98 % r. F. kann das Verhalten des Sensors mit einer minimalen Abweichung von ± 1,5 % r. F. durch folgende lineare Gleichung dargestellt werden:

$C(r.F.) = C_{76} * [1 + FK * (r.F. - 76)]$
mit $FK = 2800 \pm 120$ ppm / % r. F.

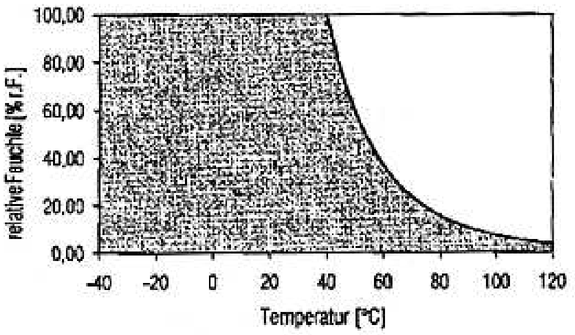


Im Falle einer softwaremäßigen Erfassung der Sensor-charakterisiik, kann für hochgenaue Anwendungen die Kennlinie bei 30 °C durch folgendes Polynom beschrieben werden:

$C(r.F.) = C_{76} * [1 + FK * (r.F. - 76) + K(r.F.)]$
mit $K(r.F.) = A1*r.F. + A2*r.F.^{1.5} + A3*r.F.^2 + A4*r.F.^{2.5}$ / A1= 2,1159E-3 / A2= -7,6305E-4 / A3= +8,9470E-5 / A4= -3,4130E-6

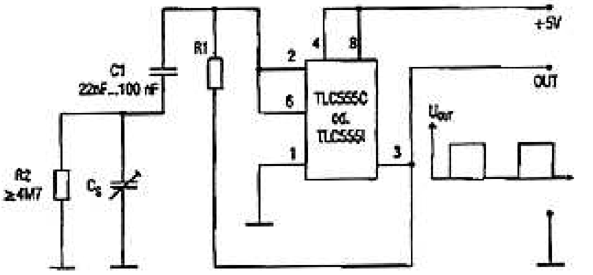
ARBEITSBEREICH

Die schraffierte Fläche zeigt den erlaubten Meßbereich für den Feuchtesensor HC 101.
Arbeitspunkte, die außerhalb dieses Bereiches liegen, müssen
zwar nicht zur Zerstörung des Sensorelementes führen, die spezifizierte Meßgenauigkeit kann aber nicht garantiert werden.
Weiters muß bei gleichzeitigem Auftreten von hoher Feuchte und hoher Temperatur eine zeitliche Komponente berücksichtigt werden.



TESTSCHALTUNG

Die vorgeschlagene Testschaltung liefert ein Taktsignal mit feuchteabhängiger Frequenz. Die Meßfrequenz ist über den Widerstand R1 einstellbar. Bei einem Wert von 62 kOhm ergibt sich eine Taktfrequenz von etwa 50 kHz bei 76% r. F.



KALIBRIERUNG

Werksseitig werden alle gelieferten Sensorelemente einer Prüfung bei Feuchte-Referenzwerten unterzogen. Wir empfehlen die Referenzpunkte 0 % r. F. und 76 % r. F. für die Kalibration. Hohe Feuchtewerte sind nur bedingt zu empfehlen, da Betauungsvorgänge die präzise Kalibrierung beeinträchtigen können.

E + E ELEKTRONIK Ges.m.b.H.
Langwiesen 7, A-4210 Engerwitzdorf, AUSTRIA
Tel: ++ 43/7235/605 - 0 Fax : ++ 43/7235/605-8
e-mail: vertrieb@epluse.co.at

Feuchtesensor Mela

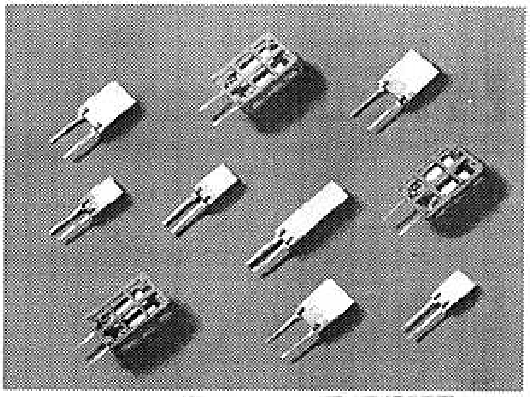
Feuchtesensorelemente, Module

MELA Sensortechnik GmbH Fritz-Ebert-Straße 16 D-07973 Greiz Tel: 03661/627040 Fax: 03661/6270413

MELA SENSORTECHNIK

Beschreibung

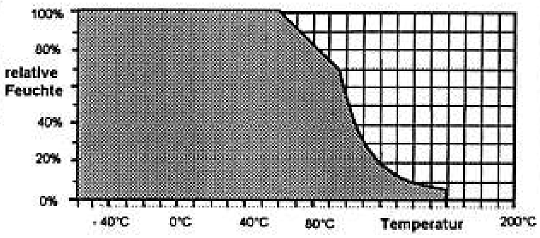
Die ***MELA-Feuchtesensorelemente*** dienen zur Messung der relativen Feuchte in Luft und anderen nichtaggressiven Gasen und arbeiten nach dem kapazitiven Meßprinzip.
Auf einem glasierten Keramiksubstrat befindet sich ein Elektrodensystem und eine feuchteempfindliche Polymerschicht. Dieses Schichtsystem stellt einen feuchteabhängigen Kondensator dar, dessen Kapazität ein Maß für die umgebende relative Feuchte ist.
Mit unterschiedlichem Schichtaufbau und verschiedenen Abmessungen werden mehrere Typen von Feuchtesensorelementen hergestellt, die sich vor allem in ihrer Widerstandsfähigkeit gegenüber äußeren Einflüssen und durch unterschiedliche Kapazität unterscheiden. Damit sind sie auch für viele Anwendungsfälle geeignet. Alle Sensorelemente haben sehr gute dynamische Eigenschaften, sind langzeitstabil und wasserfest, das heißt, auch Betauung schadet dem Sensorelement nicht.



Feuchtesensorelemente Typ FE 09/4 sind preiswerte Elemente mit sehr kleinen Abmessungen, geeignet für vielseitige Anwendungen, auch mit Schutzrahmen ***(Typ FE 09.R/4)*** lieferbar.
Feuchtesensorelemente Typ FE 09/1000 sind gekennzeichnet durch eine extrem hohe Grundkapazität und durch einen doppelten Schichtaufbau, der das Element widerstandsfähiger gegen äußere, mechanische Einflüsse macht. Diese hohe Grundkapazität senkt den Störkapazitätseinfluß erheblich. Dadurch ist auch ein Einsatz abgesetzt von der Elektronik, zum Beispiel in Trocknungsanlagen, problemlos möglich.
Feuchtesensorelemente Typ FE 09/2 sind universal einsetzbare Elemente, die auch mit Schutzrahmen ***(Typ FE 09.R/2)*** angeboten werden.
Feuchtesensorelemente Typ FE 09/1 unterscheiden sich von Elementen des Typs FE09/2 nur durch den doppelten Schichtaufbau, der die Elemente widerstandsfähiger gegen äußere mechanische Einflüsse macht. Sie werden auch ***als Typ FE 09.R/1*** mit Schutzrahmen angeboten.

Zum Einlöten des Sensorelementes ist ein Kleinspannungslötkolben (Löttemperatur 240 °C, max. Lötdauer 2s) zu benutzen, Flußmittelrückstände sind zu entfernen. Die Sensorelemente sind zur Säuberung abzublasen. Sie können auch in dest. Wasser abgespült werden. Die Kennlinie verändert sich dadurch nicht. Ein exakter Meßwert ist aber erst nach vollständiger Trocknung wieder zu erreichen. Das trifft auch bei Betauung zu.

Arbeitsbereich



Anwenderhinweise

Das hochempfindliche Sensorelement sollte nicht an der aktiven Oberfläche berührt werden!

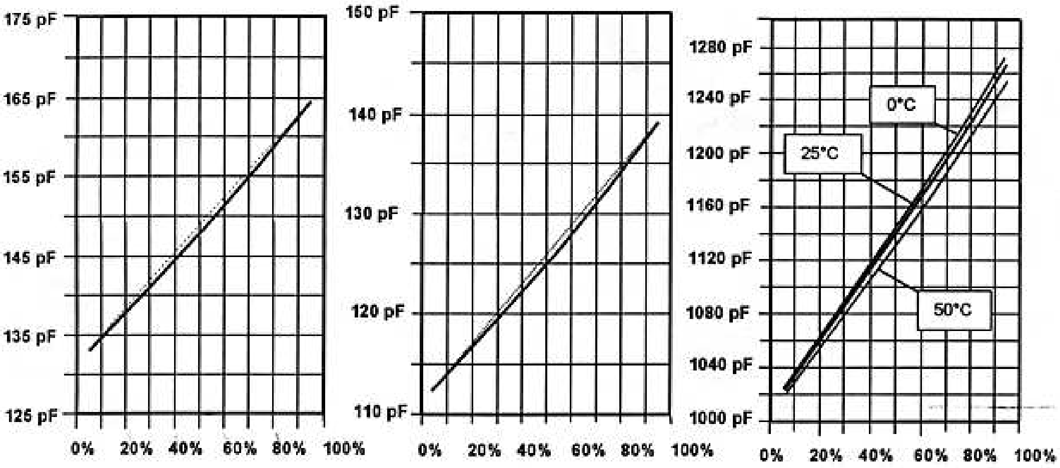
Technische Daten

Typ	FE09/1, FE09.R/1 FE09/2, FE09.R/2	FE09/4 FE09.R/4	FE09/1000
Meßbereich	0...100% r.F.	0...100% r.F.	0...100% r.F.
Temperaturfestigkeit FE09/X	-60...200°C	-60...200°C	-60...200°C
Temperaturfestigkeit FE09.R/X	-40...70°C	-40...120°C	
Grundkapazität	135±10pF	115±15pF	1050±50pF
Kapazitätsänderung	0,3±0,05pF/% r.F	0,27±0,08pF/%r.F.	2,5±0,5pF/% r.F.
Zulässige Spannung	max 3V ohne DC	max 3V ohne DC	max 3V ohne DC
Meßfrequenz	30...300kHz	30...300kHz	3...30kHz
Verlustfaktor	<0,03 (bei 50kHz)	<0,03 (bei 50kHz)	<0,1 (bei 10kHz)
Einstellzeit (T90)	10s	10s	10s
Hysterese (MB 5...95% r.F)	< 1,5% r.F.	< 1,5% r.F.	< 1,5% r.F.
Linearität (MB 5...95% r.F.)	< 1,5 % r.F.	< 1,5 % r.F.	< 1,5 % r.F.

Feuchtesensorelemente Produktinfo. Nr.: **B 1.1**

Typischer Kennlinienverlauf

Typen FE 09/1, FE 09.R/1, FE 09/2, FE 09.R/2 — Typen FE 09/4, FE 09.R/4 — Typ FE 09/1000



Schadstofftest

Keine Schädigung an den Feuchtesensorelementen nach einer Einwirkungsdauer von 3 Monaten.

Schadstoff	Konzentration	
NH_3 (nur kurzzeitig)	100 ppm	(2 x MAK)
H_2S	20 ppm	(2 x MAK)
SO_2	3,7 ppm	(10 x MIK)
No_x	1,0 ppm	(10 x MIK)
O_3	0,6 ppm	(10 x MIK)
Cl_2	1,0 ppm	(2 x MAK)

Temperaturabhängigkeit

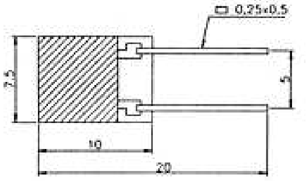
Für die Temperaturabhängigkeit der Feuchtesensorelemente kann folgender Korrekturalgorithmus angegeben werden:

$$K = [A + a(T - 25)] \cdot \sum_{i=0}^{2} b_i \cdot T^i$$

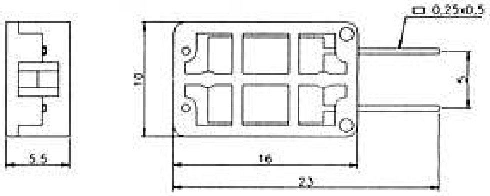
K = korrigierter Wert
A = Ausgangssignal (0...100%)
T = Temperatur (°C)
$b_1 = 6 \cdot 10^{-4}$
$a = 0{,}04$ (für $T \geq 25$°C)
$a = 0$ (für $T < 25$°C)
$b_0 = 0{,}98125$
$b_2 = 6 \cdot 10^{-6}$

Maßbilder

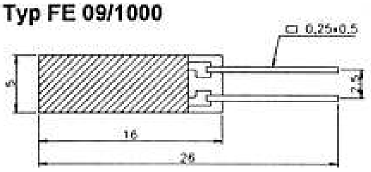
Typ FE 09/1, FE09/2



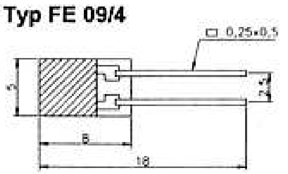
Typ FE 09.R/1, FE09.R/2



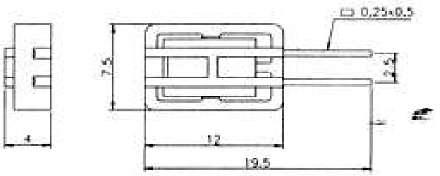
Typ FE 09/1000



Typ FE 09/4



Typ FE 09.R/4



Änderungen im Zuge der technischen Weiterentwicklung vorbehalten Ausgabe 04/98

Temperatursensor KTY10-6

SIEMENS

Silicon Spreading Resistance Temperature Sensor in Leaded Plastic Package **KT 100 KTY 10**

Features

- Temperature dependent Resistor with Positive Temperature Coefficient
- Small plastic package
- Fast response
- High reliability due to multilayer gold contacts
- n-conducting silicon crystal
- Polarity independent due to symmetrical construction
- Available selected in ± 1% tolerance groups

Modified TO-92

Electrical Characteristics
at T_A = 25 °C unless otherwise specified

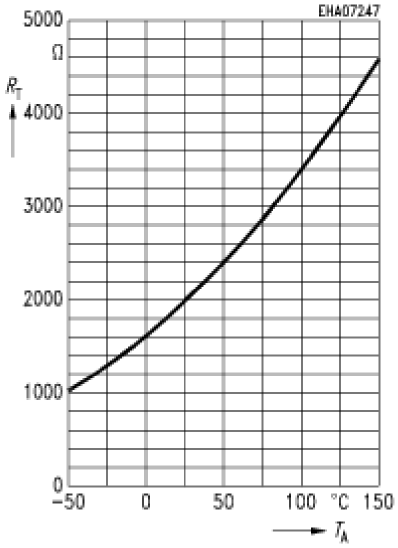
Parameter	Symbol	Limit Values			Unit
		min.	typ.	max.	
Temperature sensor resistance IB = 1 mA	R_{25}				Ω
KT 100		1940	–	2060	
KTY 10-5		1950	–	1990	
KTY 10-6		1980	–	2020	
KTY 10-62		1990	–	2010	
KT 10-7		2010	–	2050	
Thermal time constant (63% of ΔT_A)					s
in still air	τ_{air}	–	40	–	
in still oil (Freon FC40/PP7)	τ_{oil}	–	4	–	

Type	Marking	Ordering Code	Pin Configuration		Package
			1	2	
KT 100	KT 100	Q62705-K331	electrical contact	electrical contact	Modified TO-92
KTY 10-5	KTY 10-5	Q62705-K110			
KTY 10-6	KTY 10-6	Q62705-K132			
KTY 10-62	KTY 10-62	Q62705-K71			
KTY 10-7	KTY 10-7	Q62705-K111			

SIEMENS **KT- and KTY-Series Temperature Sensors**

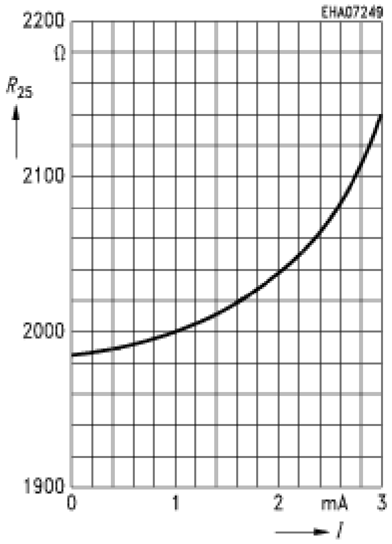
Sensor Resistance $R_T = k_T \times R_{25} = f(T_A)$

I_B = 1 mA; Example: R_{25} = 2000 Ω



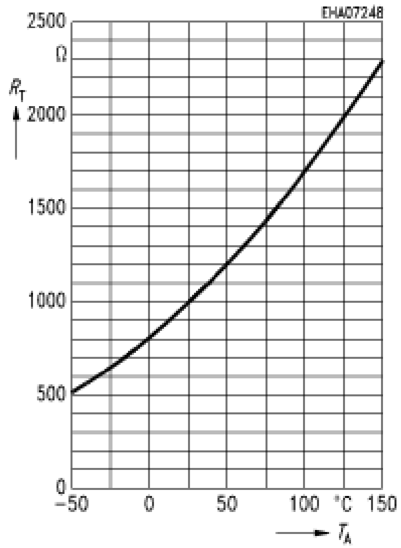
Typical Dependence of Sensor Resistance on Supply Current

Example: KTY 10-6 in oil at T_A = 25 °C



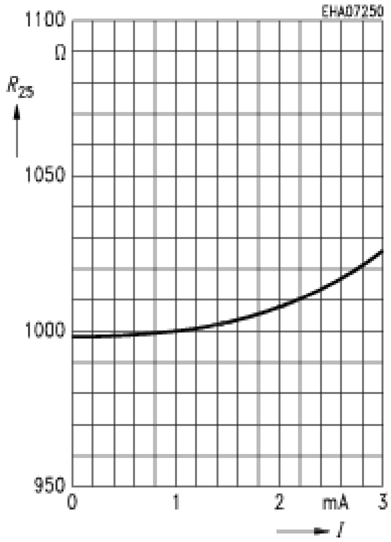
Sensor Resistance $R_T = k_T \times R_{25} = f(T_A)$

I_B = 1 mA; Example: R_{25} = 1000 Ω



Typical Dependence of Sensor Resistance on Supply Current

Example: KTY 21-6 in oil at T_A = 25 °C



A7 Schaltpläne

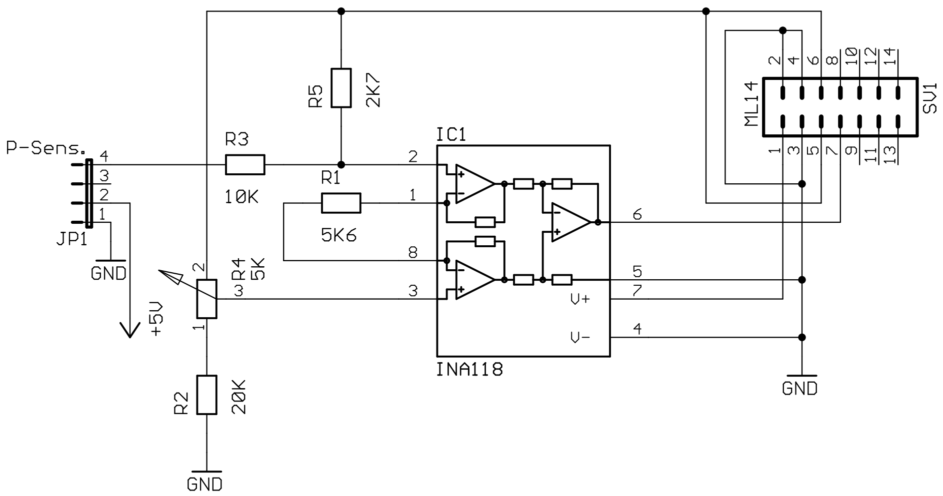


Abbildung A7.1: Pegelanpassung des Drucksensors

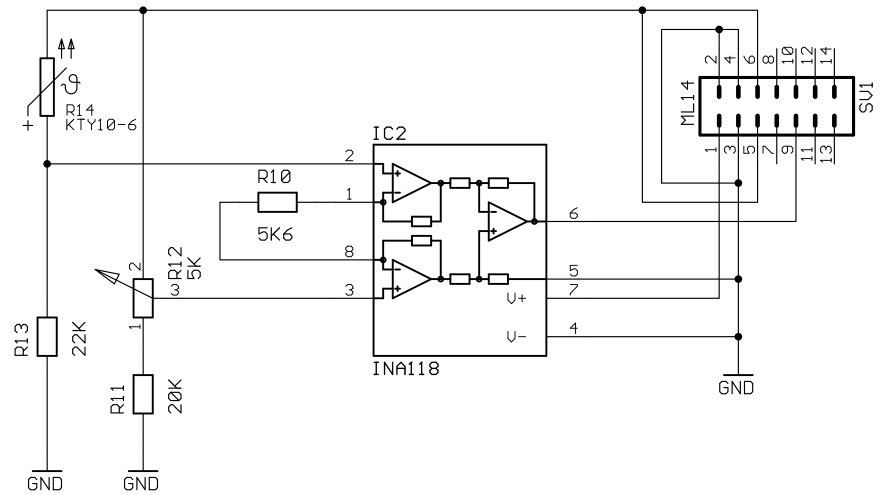


Abbildung A7.2: Pegelanpassung des Temperatursensors KTY 10-6

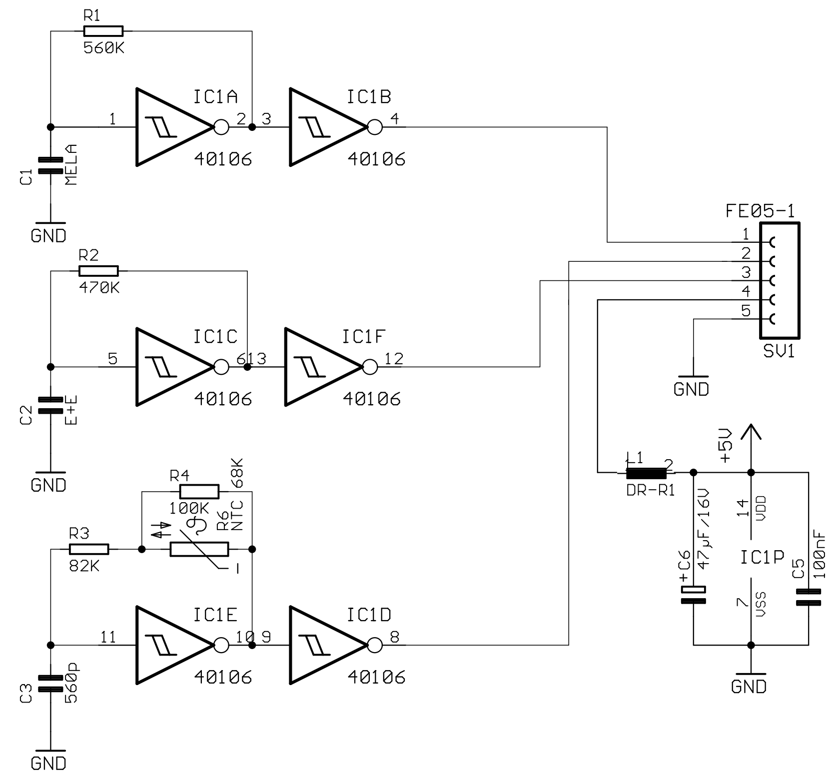


Abbildung A7.3: Schwingkreis der Feuchtesensoren und des NTC

A8 Listings

Der vollständige Source Code der erstellten Software kann im Internet von der Startseite des Online-Berichts geladen werden. Hier sollen nur wichtige und beispielhafte Teile der Software gezeigt werden.

CAN Profil-Definition für die Latenzzeitmessung

```
/*
****************************************************************
$Source: f:/work/DplAr/src/include/lprofile.h,v $
$Author: hmueller $
$Date: 1998/09/11 05:43:34 $
$Revision: 1.1 $
****************************************************************
Compiler: Microsoft Visual C++ 5.0 / Keil C166 3.0
Module  : lprofile.h
Purpose : CAN profile for latency nodes

Modification:
 * $Log: lprofile.h,v $
 * Revision 1.1  1998/09/11 05:43:34  hmueller
 * Initial revision
 *
****************************************************************
*/
#ifndef _LPROFILE_H_
#define _LPROFILE_H_

#define NMT           1         // Msg object number for NMT
#define NMT_ID        0         // CAN ID for NMT
#define NMT_CMD       0         // offset for NMT command entry
#define INIT_CMD      0x01      // NMT command for Init
#define START_CMD     0x02      // NMT command for Start
#define STOP_CMD      0x03      // NMT command for Stop

#define PREPARED      2         // Msg object number for Prepared
#define PREPARED_ID   1         // CAN ID for Prepared
#define PREPARED_DATA 0         // offset for Prepared data entry

#define CANmstr_HWADR 0x01      // Hardware address of CANmaster
#define CANsend_HWADR 0x02      // Hardware address of CANsend
#define CANlog_HWADR  0x03      // Hardware address of CANlog

#define LOGDATA       3         // Msg object number for LogData
#define LOGDATA_ID    2         // CAN ID for LogData
#define LOGDATA_CntD  0         // offset for CntD data
#define LOGDATA_tHi   1         // offset for Time high byte
#define LOGDATA_tLo   2         // offset for Time low byte
#define LOGDATA_MAX   256       // max. number of data to log

#define SENDDATA      4         // Msg object number for SendData
```

```
#define SENDDATA_ID   400       // CAN ID for SendData
#define SENDDATA_CntD 0         // offset for CntD data

#define TRFCHI_ID     128       // CAN ID for high bus traffic
#define TRFCLO_ID     1792      // CAN ID for low bus traffic
#define TRFC_CntT     0         // offset for traffic CntT data

#define T6_1MS        2500      // 1ms takes 2500 counts at 0.4us res.

#endif /* _LPROFILE_H_ */
```

CAN Profil-Definition für die CAN-Wetterstation

```
/*
****************************************************************
$Source: f:/work/DplAr/src/include/mprofile.h,v $
$Author: hmueller $
$Date: 1998/09/11 05:45:14 $
$Revision: 1.1 $
****************************************************************
Compiler: Microsoft Visual C++ 5.0 / Keil C166 3.0
Module  : mprofile.h
Purpose : CAN profile for measurement nodes

Modification:
 * $Log: mprofile.h,v $
 * Revision 1.1  1998/09/11 05:45:14  hmueller
 * Initial revision
 *
****************************************************************
*/
#ifndef _MPROFILE_H_
#define _MPROFILE_H_

#define SENS_D_1      1         // Msg object number for damp sensor 1
#define SENS_D_1_ID   0x0101    // CAN ID for damp sensor 1
#define SENS_D_1_DATA 0         // offset for damp sensor 1 data

#define SENS_D_2      2         // Msg object number for damp sensor 2
#define SENS_D_2_ID   0x0102    // CAN ID for damp sensor 2
#define SENS_D_2_DATA 0         // offset for damp sensor 2 data

#define SENS_D_3      3         // Msg object number for damp sensor 3
#define SENS_D_3_ID   0x0103    // CAN ID for damp sensor 3
#define SENS_D_3_DATA 0         // offset for damp sensor 3 data

#define SENS_P        5         // Msg object number for pressure sensor
#define SENS_P_ID     0x0105    // CAN ID for pressure sensor
#define SENS_P_DATA   0         // offset for pressure sensor data

#define SENS_T        6         // Msg object number for temperature sensor
#define SENS_T_ID     0x0106    // CAN ID for temperature sensor
#define SENS_T_DATA   0         // offset for temperature sensor data

#define TIME_1S       10        // Time count for 1 s
#define TIME_1MIN     TIME_1S*60 // Time count for 1 min
```

```
#define TIME_10MIN    TIME_1MIN*10 // Time count for 10 min

#define MAX_DAMP_PULSE_CNT 100  // number of pulses to summarize
#define T5_RES 1.6e-6           // timer 5 resolution

#endif /* _MPROFILE_H_ */
```

CANmaster für die Latenzzeitmessung

Der CANmaster Code kann auch als Beispielcode für die Programmierung des PEAK-Service CAN-Dongle verwendet werden.

```
/*
****************************************************************
$Source: f:/work/DplAr/src/CANmaster/CANmaster.c,v $
$Author: hmueller $
$Date: 1998/09/11 05:48:51 $
$Revision: 1.1 $
****************************************************************
Compiler: Microsoft Visual C++ 5.0
Module  : CANmaster.c
Purpose : Master of CAN network. Inits all clients,
  generates traffic, receives data from CANlog and writes them
  into a file.

Modification:
 * $Log: CANmaster.c,v $
 * Revision 1.1  1998/09/11 05:48:51  hmueller
 * Initial revision
 *
****************************************************************
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <time.h>
#include <float.h>

#include "..\include\canapi32.h"
#include "CANmaster.h"


/*
===============================================================
Function : InitCAN
---------------------------------------------------------------
Purpose  : Registers CAN client, conntects to net, registers
           messages
Call     : InitCAN(&hClient, &hNet, &active);
===============================================================
Parameter:
---------------------------------------------------------------
VAR             I/O TYPE       REMARK
---------------------------------------------------------------
hClient          O  HCANCLIENT handle of client
```

```
hNet              O  HCANNET    handle of net
active            O  ubyte      TRUE: CAN init was ok
                                FALSE: something was wrong while CAN init
============================================================
*/
void InitCAN(HCANCLIENT *hClient, HCANNET *hNet, ubyte *active)
{
  int err;                          // error status
  HCANHW hHw;                       // hardware handle

  // if init runs through, do the send/receive loop
  *active = TRUE;

  // register at driver
  if (*active) {
    err = CAN_RegisterClient(CANmstr_CLIENT, 0, hClient);
    if (err != CAN_ERR_OK) {
      printf("\nError CAN_RegisterClient\n");
      printf("Application will be terminated.\n");
      *active = FALSE;          // skip the send/receive loop!
    }
  }

  // connect client to net
  if (*active) {
    err = CAN_ConnectToNet(*hClient, CANmstr_NET, hNet);
    if (err != CAN_ERR_OK) {
      printf("\nError CAN_ConnectToNet\n");
      printf("Application will be terminated.\n");
      *active = FALSE;          // skip the send/receive loop!
    }
  }

  // register rcv messages
  if (*active) {
    err = CAN_RegisterMsg(*hClient, *hNet, PREPARED_ID);
    if (err != CAN_ERR_OK) {
      printf("\nError CAN_RegisterMsg\n");
      printf("Application will be terminated.\n");
      *active = FALSE;          // skip the send/receive loop!
    }
  }
  if (*active) {
    err = CAN_RegisterMsg(*hClient, *hNet, LOGDATA_ID);
    if (err != CAN_ERR_OK) {
      printf("\nError CAN_RegisterMsg\n");
      printf("Application will be terminated.\n");
      *active = FALSE;          // skip the send/receive loop!
    }
  }

  // reset hardware
  if (*active) {
    err = CAN_HardwareOfNet(*hNet, &hHw);
    if (err != CAN_ERR_OK) {
      printf("\nError CAN_HardwareOfNet\n");
      printf("Application will be terminated.\n");
```

```
      *active = FALSE;          // skip the send/receive loop!
    }
    err = CAN_ResetHardware(hHw);
    if (err != CAN_ERR_OK) {
      printf("\nError CAN_ResetHardware\n");
      printf("Application will be terminated.\n");
      *active = FALSE;          // skip the send/receive loop!
    }
  }

  return;
} /* InitCAN */


/*
===============================================================
Function : Wait4Prepared
---------------------------------------------------------------
Purpose  : Wait for PREPARED answer of CAN clients
Call     : Wait4Prepared(hClient, &ClientStatus);
===============================================================
Parameter:
---------------------------------------------------------------
VAR             I/O TYPE           REMARK
---------------------------------------------------------------
hClient          I  HCANCLIENT     handle of client
ClientStatus    I/O ClientStatus_t status of clients
===============================================================
*/
void Wait4Prepared(HCANCLIENT hClient, ClientStatus_t *ClientStatus)
{
  WORD err;                     // error status
  uword Wait4Prepared;          // counter waiting for PREPARED
  long rcvtime;                 // time when message was received
  TCANMsg msg;                  // message data
  HCANNET rcvNet;               // net name, of which message came from
  char text[255];               // buffer for text output

  Wait4Prepared = 100;          // number of read loops
  while (Wait4Prepared--) {
    SleepEx(1, TRUE);           // wait at least 1 ms
    err = CAN_Read(hClient, &msg, &rcvNet, &rcvtime);
    if (err == CAN_ERR_OK) {
      if (msg.ID == PREPARED_ID) {
        // We've got a message from a client ;-)
        switch (msg.DATA[PREPARED_DATA]) {
        case CANmstr_HWADR:
          // This is not possible!!!
          printf("\nERROR: Hardware address %#0.2x is used by CANmaster!\n"\
                 , CANmstr_HWADR);
          break;
        case CANsend_HWADR:
          // CANsend is up :-)
          printf("\nGot PREPARED by CANsend.\n");
          ClientStatus->CANsend = TRUE;
          break;
        case CANlog_HWADR:
```

```
          // CANlog is up :-)
          printf("\nGot PREPARED by CANlog.\n");
          ClientStatus->CANlog = TRUE;
          break;
        default:
          // Got unused hardware adress :-(
          printf("\nERROR: Hardware address %#0.2x not known to me!\n"\
                 , msg.DATA[PREPARED_DATA]);
          break;
        }
      } else {
        printf("\nERROR: Got unknown message ID %#0.3x!\n", msg.ID);
      }
    } else if ((err & ~CAN_ERR_QRCVEMPTY) != 0) {
      // We've found an error :-(
      CAN_GetErrText(err, text);
      printf("\nERROR: %s\n", text);
    }
    if (ClientStatus->CANsend && ClientStatus->CANlog) {
      // all Clients are PREPARED
      Wait4Prepared = 0;
    }
  } /* while(Wait4Prepared--) */
} /* Wait4Prepared */


/*
===============================================================
Function : InitClients
---------------------------------------------------------------
Purpose  : Init clients of CAN network (until PREPARED)
Call     : err = InitClients(hClient, hNet, &ClientStatus);
===============================================================
Parameter:
---------------------------------------------------------------
VAR             I/O TYPE           REMARK
---------------------------------------------------------------
hClient          I  HCANCLIENT     handle of client
hNet             O  HCANNET        handle of net
ClientStatus    I/O ClientStatus_t status of clients
err              O  WORD           error status (-1 = abort,
                                   else CAN status s. canapi32.h)
===============================================================
*/
WORD InitClients(HCANCLIENT hClient, HCANNET hNet, \
                 ClientStatus_t *ClientStatus)
{
  WORD err;                     // error status
  long sndtime;                 // time for sending message
  ubyte Wait4Clients;           // TRUE, if waiting for clients
  TCANMsg msg;                  // message data

  CAN_GetSystemTime(&sndtime);  // get current system time

  err = CAN_ERR_OK;
  ClientStatus->CANsend = FALSE;
  ClientStatus->CANlog = FALSE;
```

```
  Wait4Clients = TRUE;
  while (Wait4Clients) {
    printf(".");
    /*
     * Send INIT_CMD to all clients
     */
    msg.ID = NMT_ID;             // set NMT message ID
    msg.RTR = FALSE;
    msg.LEN = 1;
    msg.DATA[NMT_CMD] = INIT_CMD; // set init command
    err = CAN_Write(hClient, hNet, &msg, &sndtime);
    if (err != CAN_ERR_OK) {
      printf("\nError CAN_Write: %#x\n", err);
    }

    /*
     * Waiting for PREPARED answer
     */
    Wait4Prepared(hClient, ClientStatus);
    if (ClientStatus->CANsend && ClientStatus->CANlog) {
      // all Clients are PREPARED
      Wait4Clients = FALSE;
    }

    if (kbhit()) {
      getch();                   // remove pressed key
      Wait4Clients = FALSE;
      err = -1;
    }
  } /* while (Wait4Clients) */

  printf("\n");
  return(err);
} /* InitClients */


int main(int argc, char *argv[], char *envp[])
{
  WORD err;                      // error status
  ubyte active;                  // TRUE - loop, FALSE - exit
  long sndtime;                  // time for sending message
  long rcvtime;                  // time when message was received
  HCANCLIENT hClient = 0;        // my client
  HCANNET hNet = 0;              // my net
  HCANNET rcvNet;                // net name, of which message came from
  TCANMsg msg;                   // message data
  TClientInfo ClientInfo;        // info buffer about client
  FILE *f_log;                   // log file handle
  char f_name[128];              // log file name
  char text[255];                // buffer for text output
  ClientStatus_t ClientStatus;   // status of clients
  uword time;                    // measurement time
  ubyte trafficCnt = 0;          // counter for traffic generation
  ubyte trafficLevel;            // traffic level (0=none, 1=low, 2=high)

  printf("  *** Starting CANmaster by Holger Mueller 08/98 ***\n");
  printf("Client: \"%s\", Network: \"%s\"\n", \
```

```
       CANmstr_CLIENT, CANmstr_NET);

  if (argc < 2) {
    printf("\nError: Missing arguments.\n\n");
    printf("Usage: %s <trafficLevel> [logFileName]\n", argv[0]);
    printf("\ttrafficLevel\t0=none, 1=low, 2=high\n");
    printf("\tlogFileName\tlog filename (default=%s)\n", LOG_FNAME);
    return(1);
  } else {
    trafficLevel = atoi(argv[1]);
    if (trafficLevel > 2) {
      printf("\nError: Illegal traffic level!\n");
      printf("Application will be terminated.\n");
      return(1);
    }
  }

  // open log file
  if (argc > 2)
    strcpy(f_name, argv[2]);
  else
    strcpy(f_name, LOG_FNAME);
  f_log = fopen(f_name, "a");

  // init CAN
  CAN_GetSystemTime(&sndtime);  // get current system time
  InitCAN(&hClient, &hNet, &active);

  if (active) {
    err = InitClients(hClient, hNet, &ClientStatus);
    if (err != CAN_ERR_OK) {
      printf("Error InitClients: %#x\n");
      printf("Application will be terminated.\n", err);
      active = FALSE;
    }
  }

  if (active) {
    /*
     * All clients ready, send NMT START_CMD
     */
    printf("Sending START_CMD command ...\n");
    msg.ID = NMT_ID;             // set NMT message ID
    msg.RTR = FALSE;
    msg.LEN = 1;
    msg.DATA[NMT_CMD] = START_CMD; // set start command
    err = CAN_Write(hClient, hNet, &msg, &sndtime);
    if (err != CAN_ERR_OK) {
      printf("Error CAN_Write(START_CMD): %#x\n", err);
      printf("Application will be terminated.\n");
      active = FALSE;
    }
  }

  msg.DATA[LOGDATA_CntD] = 0;
  if (active) {
    do {
```

```
      /*
       * generate bus traffic
       */
      if (trafficLevel > 0) {
        err = CAN_GetClientDiag(hClient, &ClientInfo);
        if (err != CAN_ERR_OK) {
          printf("Error CAN_GetClientDiag: %#x\n", err);
        }
        //printf("XmtQueueFill: %d\n", ClientInfo.XmtQueueFill);
        SleepEx(1, TRUE);       // wait at least 1 ms
        if (ClientInfo.XmtQueueFill < 20) {
          if (trafficLevel == 1)
            msg.ID = TRFCLO_ID; // set low traffic ID
          else
            msg.ID = TRFCHI_ID; // set high traffic ID
          msg.RTR = FALSE;
          msg.LEN = 8;
          msg.DATA[TRFC_CntT] = trafficCnt++; // set start command
          err = CAN_Write(hClient, hNet, &msg, &sndtime);
          if (err != CAN_ERR_OK) {
            printf("Error CAN_Write(TRFCLO_ID): ");
            printf("%#x - trafficCnt: %d - XmtQueueFill: %d\n", \
                   err, trafficCnt, ClientInfo.XmtQueueFill);
          }
          msg.DATA[LOGDATA_CntD] = 0;
        }
      }

      /*
       * receive log data from CANlog
       */
      do {
        err = CAN_Read(hClient, &msg, &rcvNet, &rcvtime);
        if ((err == CAN_ERR_OK) && (msg.ID == LOGDATA_ID)) {
          // we've got LOGDATA !!!
          trafficLevel = 0;     // turn off traffic
          sprintf(text, "%lu\t%#0.3x\t%3u\t", rcvtime, msg.ID, \
                  msg.DATA[LOGDATA_CntD]);
          printf(text);
          fprintf(f_log, text);
          time = ((uword)msg.DATA[LOGDATA_tHi] << 8) | \
            (uword)msg.DATA[LOGDATA_tLo];
          sprintf(text, "%u\t%f\n", time, (float)time/T6_1MS);
          printf(text);
          fprintf(f_log, text);
        }
        if (msg.DATA[LOGDATA_CntD] == LOGDATA_MAX-1) {
          active = FALSE;
        }
      } while ((err == CAN_ERR_OK) && \
               (msg.DATA[LOGDATA_CntD] != LOGDATA_MAX-1));

      if (kbhit()) {
        getch();                      // remove pressed key
        active = FALSE;
      }
    } while ((active) && (msg.DATA[LOGDATA_CntD] != LOGDATA_MAX-1));
```

```
  }

  if (active) {
    /*
     * send NMT STOP_CMD
     */
    msg.ID = NMT_ID;          // set NMT message ID
    msg.RTR = FALSE;
    msg.LEN = 1;
    msg.DATA[NMT_CMD] = STOP_CMD; // set stop command
    CAN_Write(hClient, hNet, &msg, &sndtime);
    printf("Send STOP_CMD command.\n");
  }

  if (hNet != 0) {
    /*
     * send NMT STOP_CMD
     */
    printf("\nSending STOP_CMD command");
    msg.ID = NMT_ID;            // set NMT message ID
    msg.RTR = FALSE;
    msg.LEN = 8;
    msg.DATA[NMT_CMD] = STOP_CMD; // set stop command
    err = CAN_Write(hClient, hNet, &msg, &sndtime);
    if (err != CAN_ERR_OK) {
      printf("\nError CAN_Write: %#x\n", err);
    }
    ClientInfo.XmtQueueFill = 1; // to enable the following loop
    while ((err == CAN_ERR_OK) && (ClientInfo.XmtQueueFill != 0)) {
      printf(".");
      err = CAN_GetClientDiag(hClient, &ClientInfo);
      if (err != CAN_ERR_OK) {
        printf("\nError CAN_GetClientDiag: %#x\n", err);
      }
    }
    printf("\n");
  }

  printf("Produced %d traffic frames.\n", trafficCnt);

  //  Disconnect client from net
  CAN_DisconnectFromNet(hClient, hNet);

  // Free ressources
  CAN_RemoveClient(hClient);

  // close log file
  fclose(f_log);

  printf("\n");
  return(0);
} /* main */
```