Die Kommunikation zwischen Mikrocontrollern

Elektronische Systeme mit modularem Aufbau werden meist als Multicontrollersysteme entworfen. Dies resultiert aus verschiedenen Anforderungen; einerseits aus der Modularität selbst, dann aus der Übersteigung der Leistungsfähigkeit eines einzelnen Controllers und der eventuellen räumlichen Trennung verschiedener Systemgruppen. Dadurch ist man gezwungen Kommunikation zwischen verschiedenen Controllern zu betreiben. Auf Grund des einfachen Hardwareaufwands sowie der Störsicherheit wird die Kommunikation auf einem seriellen Bus bevorzugt. Als Basisnorm für serielle Bussysteme dient die Norm ISO/OSI 7498 (International Standardization Organisation, Open Systems Interconnection), in der ein sieben Schichtenmodell für serielle Schnittstellen definiert wird.

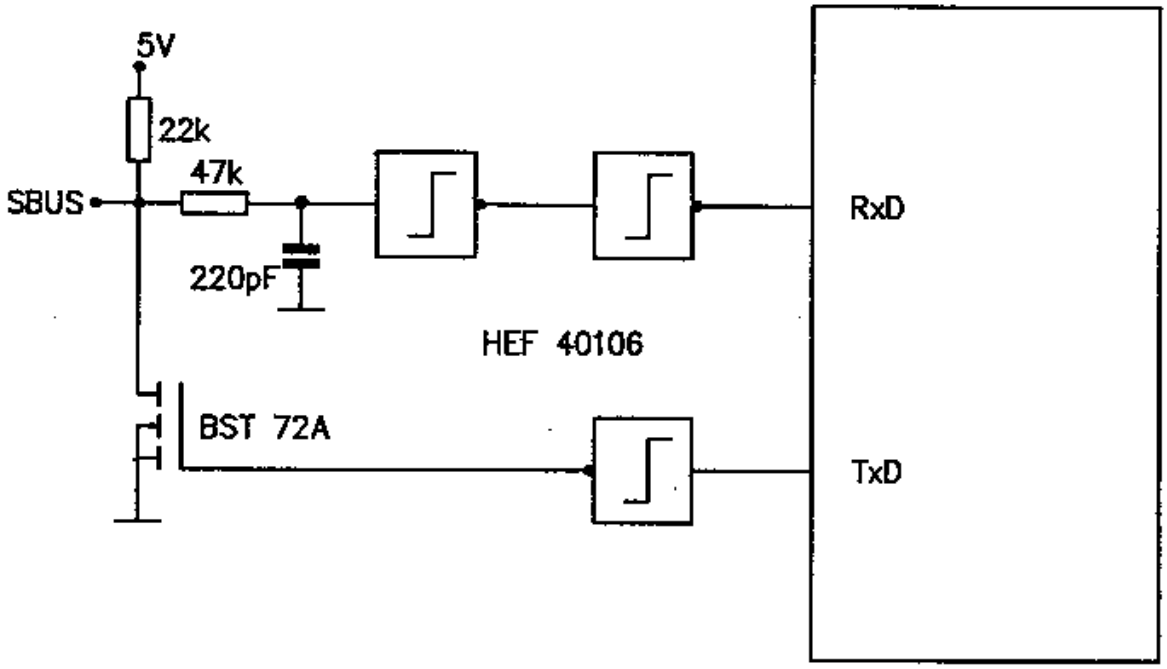
1. Physical Layer — Physikalische Schicht
   In ihr werden Signalverlauf, duplex, halbduplex, synchron, asynchron, und die Hardware definiert
2. Data Link Layer — Sicherungsschicht
   In ihr werden der Übertragungsrahmen, Parity, Quittierung und die Zugangskontrolle zum Bus vereinbart.
   Bei der Zugangskontrolle unterscheidet man z. B.:
   a.) CSMA/CD (Carrier Sense Multiple Access with Collision Detection)
   b.) CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance)
   c.) Token Ring
   d.) Token Bus
3. Network Layer — Vermittlungsschicht
4. Transport Layer — Transportschicht
5. Session Layer — Sitzungsschicht
6. Presentation Layer — Präsentationsschicht
7. Application Layer — Anwendungsschicht

Für den Datenverkehr zwischen Controllern sind die Schichten 1, 2 und 7 zu berücksichtigen.

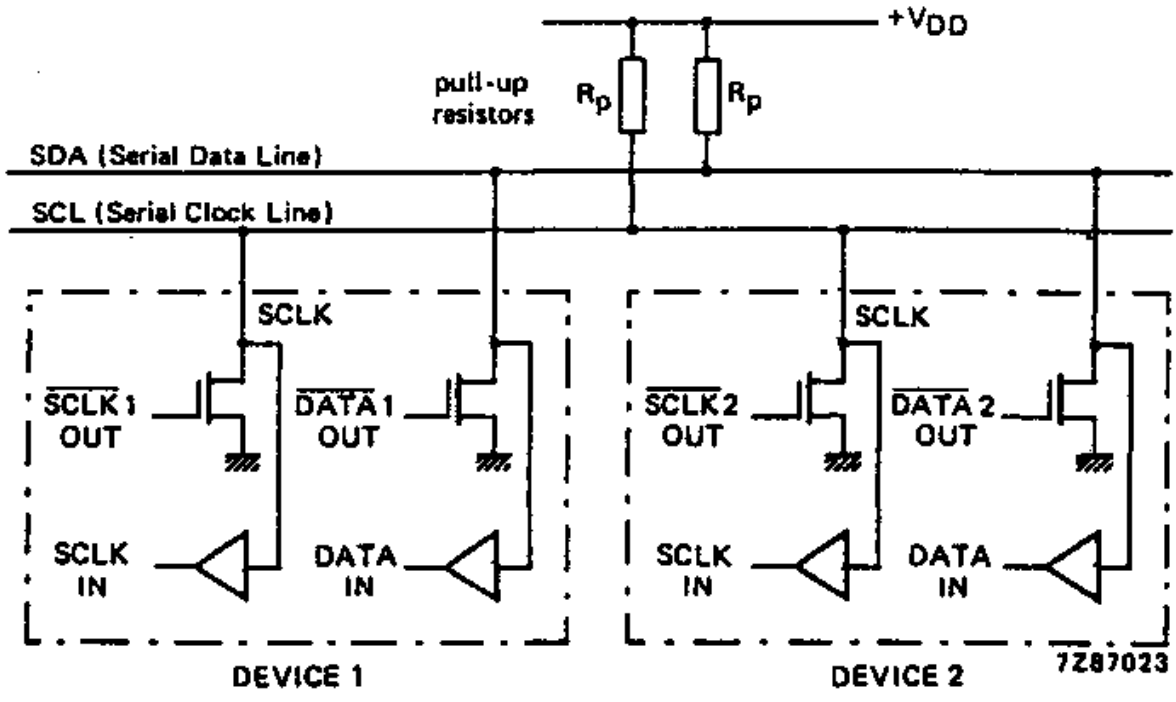
Physical Layer

Als einfachste Hardwareinterfaces bewähren sich Open Drain Verbindungen zwischen mehreren Controllern. Dabei wird der serielle Ausgang des Controllers auf das Gate eines MOS-FETs geschaltet, diese Open Drain Anschlüsse werden mit einer Leitung zwischen den Controllern verbunden und mit einem Pull-up Widerstand versehen. Über einen Treiber wird diese Leitung auch auf den seriellen Eingang der Controller geschaltet. Jeder Controller kann diese Leitung auf LOW ziehen, er selbst und die anderen lesen diesen Pegelsprung mit. Die beiden folgenden Schaltungen bedienen sich dieses Prinzips, wobei der asynchrone SBUS mit einer Leitung auskommt, der synchrone I2C Bus zwei Leitungen für Takt und Daten benötigt.

SBUS Hardware Interface

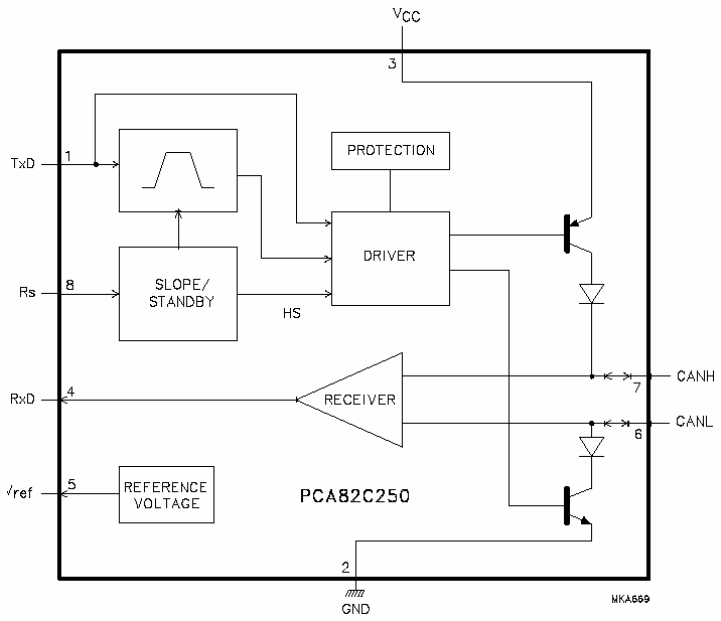


I2C BUS Hardware Interface



CAN BUS Hardware Interface

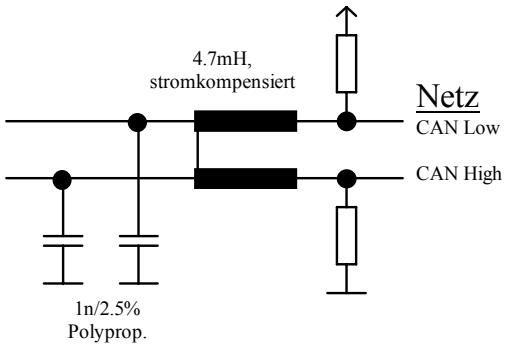
Die physikalische Ebene des CAN-Busses besteht aus zwei Leitungen (z.B. twisted pair). Um eine möglichst hohe Störsicherheit zu erreichen, werden zwei symmetrische, physikalische Signale verwendet. Ein Komparator wertet das Differenzsignal aus. Bei Ausfall einer Leitung durch Kurzschluß oder Unterbrechung ist noch ein Eindrahtbetrieb möglich. Die maximale Übertragungsrate des CAN-Busses liegt bei 1Mbit/s für Leitungslängen bis zu 40m. Bei einer maximalen Datenlänge von acht Byte ergibt sich dabei im Mittel eine theoretische Nutzdatenrate von rund 575 kbit/s.



CAN-Bus Treiber PCA82C250 (Philips):

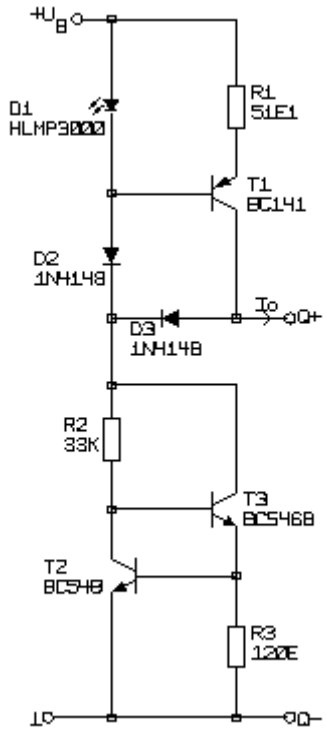
Neben zusätzlichen Extras wie zum Beispiel die Reduzierung der Flankenanstiegszeit haben die meisten CAN-Bus Treiber integrierte Schutzstrukturen.

Eingangsfilter:

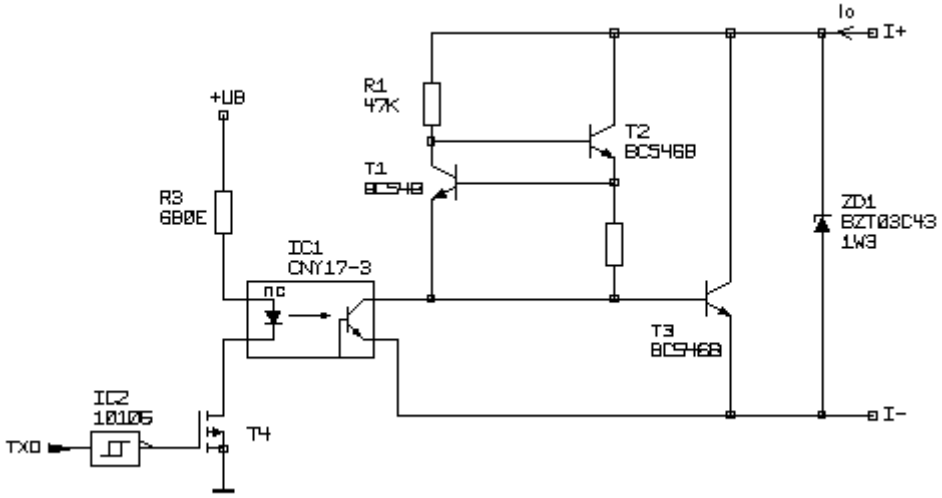


Mit den folgenden diskreten Schaltungen kann eine 0 bis 20 mA Stromschleife aufgebaut werden. Zu beachten ist, dass in der Schleife pro Teilnehmer je eine Empfänger und Sendeschaltung eingebaut werden muß, die jeweils einen Spannungsabfall von ca. 1...2 Volt ergeben, d. h. die Stromquelle muß aus einer relativ hohen Spannung gespeist werden. Häufig verwendet man auch 4 bis 20 mA, um eine Leitungsunterbrechung detektieren zu können

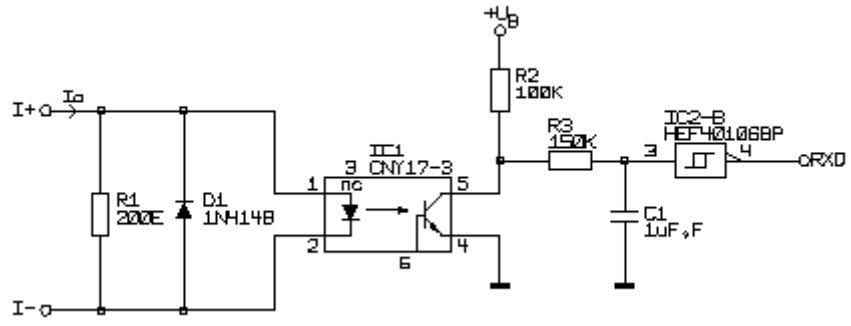
20 mA Stromquelle



20 mA Stromschleifen Sender



20 mA Stromschleifen Empfänger



Data Link Layer

SBUS

Der SBUS ist ein am Institut entwickelter serieller Bus, der ursprünglich für das modulare Prozessdatensystem mPDS 4000 konzipiert wurde. Er wird mit 4800 Baud betrieben. Der Rahmen besteht aus einem Startbit, 7 Datenbits, gerader Parität und einem Stoppbit, die Information wird im ASCII Code dargestellt. Der Bus ist für einen Single Master Betrieb konzipiert, d. h. sämtliche Module werden vom Master hintereinander abgefragt.

a.) Übertragungssequenz vom Prozessrechner zu einem Modul

? Adresse Kommando / Datum 1 /...../ Datum n / CR

Der Block beginnt mit dem Zeichen ' ? ', gefolgt von einem zweistelligen Adressfeld, das die Adresse des anzusprechenden Moduls enthält, und einem vierstelligen Kommandofeld. Daran schließen sich beliebig viele, in ihrer Länge variable Datenfelder an. Diese Datenfelder sind sowohl vom Kommandofeld als auch untereinander durch das Zeichen ' / 'getrennt. Der Sendeblock wird mit dem Zeichen 'Wagenrücklauf' beendet und der Prozessrechner schaltet auf Empfang.

b.) Übertragungssequenz von einem Modul zum Prozessrechner

! Adresse Status / Datum 1 /...../ Datum n / CR

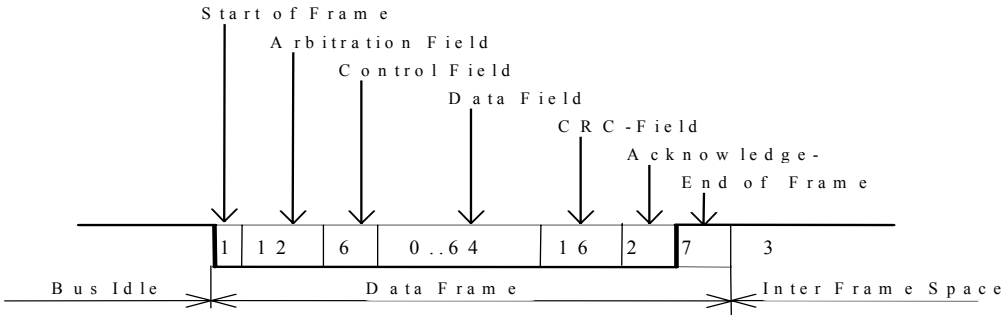
Der Block beginnt mit dem Zeichen ' ! ', gefolgt von einem zweistelligen Adressfeld, dem vierstelligen Statusfeld und einer beliebigen Anzahl von Datenfeldern mit variabler Länge. Auch hier werden das Statusfeld und die einzelnen Datenfelder durch das Zeichen ' / 'voneinander getrennt. Als Abschluss dient wieder das Zeichen 'Wagenrücklauf '.

Werden bei einer Übertragung das Kommandofeld, das Statusfeld oder Datenfelder nicht benötigt, werden sie unterdrückt bzw. durch das Trennzeichen ersetzt. Wird beispielsweise ein Eingabemodul aufgefordert seine Messdaten zu senden, besteht die Übertragung vom Prozessrechner zum Modul lediglich aus dem Startzeichen ' ? ', dem Adressfeld und dem Zeichen 'Wagenrücklauf '.

CAN BUS

Der CAN BUS kommt aus dem automotiven Bereich und ist ein sehr schneller Bus, der ideal für kurze Datensätze und gute Priorisierungsmöglichkeiten ist. Im Gegensatz zu üblichen Protokollen basiert das CAN-Protokoll nicht auf einer Adressierung des Teilnehmers, sondern gibt jedem Kommunikationsobjekt (Botschaft) eine Nummer (Identifier). Das heißt, in einem Datentelegramm gibt es weder Quell- noch Zieladresse. Die Botschaft wird gewissermaßen etikettiert und jeder beteiligte Knoten muss selbst entscheiden, ob die Information für ihn bestimmt ist. Somit ist natürlich auch Broadcasting möglich. Zur Reduzierung der ankommenden Informationsflut sind die CAN-Receiver mit Eingangsfiltern versehen.

Datentelegramm des CAN-Busses, Version 2.0 A



Start of Frame:

Der Beginn eines Telegramms wird durch ein dominantes Bit signalisiert. Ein CAN-Busteilnehmer darf mit einer Arbitrierung nur dann beginnen, wenn sich der Bus im Ruhezustand befindet, d.h. seit mindestens 11 Bit besitzt er einen rezessiven Pegel. Über die fallende Flanke des Startbits werden alle Teilnehmer auf den Sender synchronisiert.

Arbitration Field:

Das Arbitrierungsfeld besteht aus dem Identifier und dem Remote-Bit. Über den Identifier[1] wird eine Nachricht gekennzeichnet und deren Priorität festgelegt. Das Anforderungsbit (Remote Transmission Request Bit, RTR) unterscheidet zwischen Daten- und Anforderungstelegramm. Da es sinnvoll ist, einem Datentelegramm gegenüber einer gleichzeitigen Anforderung derselben Nachricht den Vorzug zu geben, wird das RTR-Bit einer Anforderung rezessiv gesendet.

Control Field:

Mit den vier niederwertigen Bits des Steuerfeldes wird die Datenlänge (Data Length Code, DLC) des nachfolgenden Datenfeldes in Byte angegeben. Ein Wert von 0 entspricht 0 Datenbytes, ein Wert von 8 entsprechend 8 Datenbytes. Es ist eine maximale Sendung von 8 Datenbytes pro Übertragung möglich.

[1] Nach CAN-Spezifikation Version 2.0 Part A mit elf Identifier-Bit lassen sich 2032 verschiedene Botschaften[1] bearbeiten, nach Part B mit 18 zusätzlichen Identifier-Bit sind es über 500 Millionen Botschaften.

Data Field:

Dieses Feld enthält die eigentliche Nutzinformation einer Nachricht und kann zwischen 0 und 8 Datenbytes enthalten. Die Übertragung eines Bytes beginnt mit dem höchstwertigen Bit.

CRC-Field (Cycle Redundancy Check):

Das Datensicherungsfeld besteht aus einer 15 Bit langen Prüfsequenz, sowie einem rezessiv übertragenen Begrenzungsbit. Mit der, in der Prüfsequenz enthaltenen Information kann ein Empfänger nachprüfen, ob die empfangenen Nachricht durch Störeinwirkung verfälscht wurde. Durch diese Blockkodierung erhält man einen Maximalwert der Restfehler-wahrscheinlichkeit von $< 10^{-3}$ * Nachrichten-Fehlerrate.

Acknowledge Field:

Aknowledge-Slot: Hier muß mindestens eine empfangsbereite Station ein dominantes Signal ausgeben um dem Sender eine korrekte Übertragung zu bestätigen.

Aknowledge-Delimiter: Wird im Rahmen der CRC-Prüfung ein Fehler festgestellt, so kann der Empfänger dies durch ein dominantes Bit signalisieren.

Arbitrierung:

CAN ist multimasterfähig, deshalb kann jeder Teilnehmer mit jedem anderen sich im Netz befindlichen Teilnehmer direkt kommunizieren. Versuchen nun zwei oder mehr Teilnehmer gleichzeitig auf den Bus zu schreiben, muss eine kontrollierte Verteilung der Zugriffsrechte erfolgen. CAN verwendet hierfür die "Bitweise Arbitrierung".

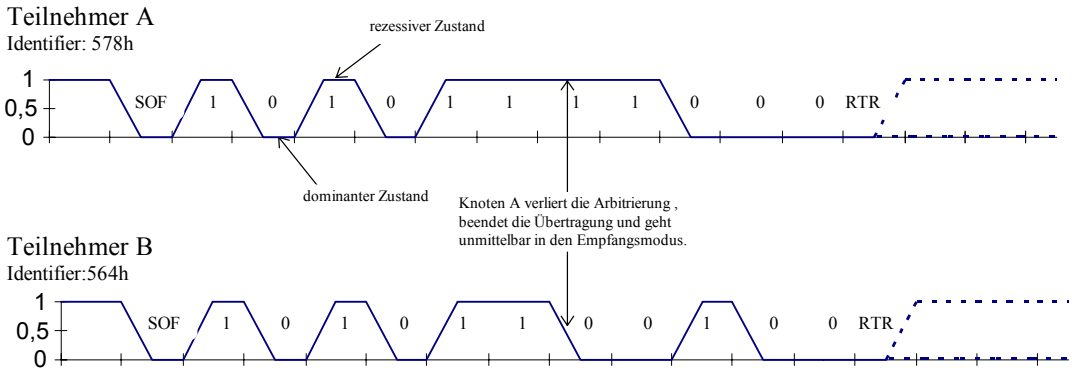
> Jeder Knoten darf eine Aussendung starten wenn sich der Bus im rezessiven Zustand befindet. Abgesehen vom Startrahmen beginnt die Übertragung des Datentelegramms mit dem höherwertigen Bit des Identifiers. Wenn nun ein Knoten eine Kollision[2] entdeckt, stoppt er die Sendung und geht unmittelbar in den Empfängermodus über.

Leicht zu erkennen ist, daß bei gleichzeitigem Zugriff jene Bewerber mit höherwertigem Identifier (und damit niedrigerer Priorität) die Arbitrierung verlieren. Das Telegramm mit dem Identifier 0 und aktivem Remotebit hätte somit die höchste Priorität im Netzwerk. Dieses Buszugriffsverfahren wird als CSMA/CA (Carrier Sense Multiple Access Collision Avoidance) bezeichnet, denn im Gegensatz zum Ethernet (CSMA/CD, Carrier Sense Multiple Access Collision Detect) wird nach einer Kollision die Übertragung nicht verworfen.

---

[2] Die physikalische Ankopplung an den Bus entspricht einer "wired AND-Verknüpfung". Nur wenn alle Teilnehmer einen rezessiven Pegel ausgeben befindet sich das Netzwerk im rezessiven Zustand. Zur Erkennung von Kollisionen ist eine Überwachung der Busaktivität auch während der Aussendung notwendig.

Beispiel für die Arbitrierung



Signalkodierung & Oszillatorgenauigkeit:

CAN verwendet Non-Return-to-Zero (NRZ) als Bitkodierungsverfahren. Diese Signaldarstellung beinhaltet prinzipiell keine Taktinformation, würde man bei gleichbleibendem Bitstrom nicht zusätzliche Bit einfügen (Bitstuffing). Der CAN-Bus erlaubt eine Resynchronisation entweder auf die steigende oder auf beide Flanken. Der Profit dieses Aufwandes ist eine Verkürzung der maximalen Flankenabstände und die damit verbundene Verringerung der Oszillatorgenauigkeit. Sobald ein Sender eine Sequenz von fünf gleichwertigen Bits erkennt fügt er ein komplementäres Bit in den tatsächlichen Bitstrom ein.

Die verwendete Non-Return-Zero (NRZ) Kodierung würde bei einer längeren Folge gleichwertiger Bits keine Flanken zur Nachsynchronisation der Empfänger aufweisen. Dieser Nachteil wird durch Einfügen von Stuff-Bits behoben.

Nach einer Folge von fünf Bits gleicher Polarität wird ein Stuffbit mit umgekehrter Polarität eingefügt. Dies geschieht auch dann, wenn der originale Datenstrom den benötigten Pegelwechsel bereits selbst aufweist.

Aus der Sichtweise des Empfängers erscheint die Vorschrift noch klarer: Das auf fünf gleichwertige Bits folgende Bit wird entfernt. Erst danach gelangt im Empfänger der so wieder hergestellte Originaldatenstrom zur weiteren Auswertelogik.

Latenzzeit:

Die Latenzzeit ergibt sich aus der maximalen Bitanzahl einer Information[3]:

- ♦ welcher Übertragungsrahmen wurde gewählt ?
- ♦ wie viele Datenbytes werden übertragen ?
- ♦ wie viele Stuffbits sind erforderlich ?

Version 2.0 Part A (Standardrahmen)	
1	Startbit
11	Identifier
1	RTR
6	Control
64	Daten
15	CRC Bits
1	CRC Delimiter
19	Stuffbits (maximale Anzahl von)
1	ACK Slot
1	ACK Delimiter
7	EOF
3	IFS (interframe space)
130	

Hieraus ergibt sich im Standardrahmen bei einer Bitrate von 125 Khz eine Latenzzeit von rund einer Millisekunde.

Fehlerbehandlung:

Eines der Entwurfsziele des CAN-Protokolls war eine sehr hohe Datensicherheit. Das erfordert neben einer störsicheren, physikalischen Schicht eine umfangreiche Fehlerbehandlung. Wird nun ein Fehler erkannt, so erfolgt ein Abbruch der gerade ablaufenden Übertragung durch das Aussenden eines Error-Frames. Anschließend startet der Controller eine selbständige Wiederholung der Sendung. Da auch alle anderen CAN-Teilnehmer diese Botschaft empfangen, ignorieren sie die gestörte Nachricht. Somit ist eine netzweite Konsistenz gewährleistet. Die aufgetretenen Fehler werden von jedem Teilnehmer getrennt für Senden und Empfangen bewertet. Ein fehlerhafter Transfer inkrementiert den Fehlerstand und ein erfolgreicher dekrementiert ihn. Je nach Zählerstand nimmt der CAN-Teilnehmer einen unterschiedlichen Fehlerstatus ein, bis hin zum selbständigen Abschalten vom Bus.

[3] Eine gerade ablaufende Übertragung darf auch durch eine höher priorisierte Nachricht nicht unterbrochen werden.

Basic-CAN, Full-CAN:

Prinzipiell können beide Mechanismen in einem Netzwerk verwendet werden. Unterschiede ergeben sich in der Implementierung des CAN-Bausteins und in der Bandbreite der zu verarbeitenden Information.

Beim BasicCAN stehen zwei Empfangsregister zur Verfügung. Sobald ein Datentelegramm das Acceptance-Filter passiert, wird die Information in ein freies Register geschrieben. Von dort sollten die Daten schnellstmöglich ausgelesen werden, um den ohnehin begrenzten Speicher wieder frei zu geben. Werden mehr als zwei Telegramme ohne Weiterverarbeitung empfangen, kommt es zu einem Datenverlust. Der Baustein signalisiert dies durch einen Overflow und sperrt die Empfangsregister. Die Aussendung erfolgt über einen Sendespeicher. Zu diesem Zweck wird das zu sendende Telegramm in den CAN-Baustein transferiert und anschließend freigegeben.

Mit dem Full-CAN erfolgt die Kommunikation mit dem Controller über ein Dual-Port RAM. Dieser Speicher bietet Platz für mehrere Datentelegramme. Zum Unterschied vom BasicCAN werden hier die ankommenden Daten direkt in den Empfangsspeicher geschrieben. Eine Aussendung erfolgt in ähnlicher Weise. Die Daten werden direkt im Sendespeicher vorbereitet und können anschließend sofort frei gegeben werden.

Im C167 sind Basic CAN und Full CAN realisiert. Dazu besitzt er 15 Full CAN Objekte, von denen das fünfzehnte als BASIC CAN mit doppeltem Eingangspuffer konfiguriert werden kann.

Einsatzgebiete:

BasicCAN

- Monitoring
- Kommunikation in einem Netzwerk mit vielen Teilnehmern, aber mit geringem Datentransfer
- wenn ausreichend Rechenleistung zur Verfügung steht und man auf die Flexibilität des BasicCAN nicht verzichten will (unter Umständen kann hier das Filter während des Betriebs verändert werden).

FullCAN

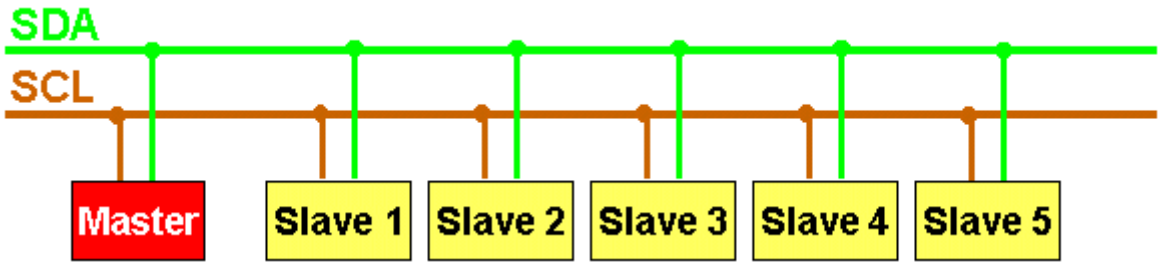
- ein hoher Datentransfer am Netz beschränkt sich auf wenige Identifier
- in Minimalsystemen (wenig Speicher, geringe Rechenleistung)

I2C BUS

Der I2C-Bus wurde 1982 von Philips Semiconductors spezifiziert, um in Geräten der Unterhaltungselektronik auf eine einfache Art Steuerungsaufgaben durchzuführen. I2C-Bus steht für Inter-Integrated-Circuit-Bus. Diese Bezeichnung erklärt die Absicht der Entwicklung: Der I2C-Bus dient der Kommunikation zwischen integrierten Schaltkreisen. Seit der Einführung des I2C-Busses durch Philips sind zahlreiche Integrierte Bausteine mit I2C-Spezifikation entwickelt worden. Viele andere Halbleiterhersteller (Siemens, Texas Instruments, SGS-Thomson, Maxim usw.) fertigen ebenfalls Chips, die den I2C-Bus nutzen. Einige Chip-Typen sind auf die Bedürfnisse der Unterhaltungselektronik zugeschnitten. Wegen der weiten Verbreitung und der Störunanfälligkeit des Busses wurden auch Schaltkreise für die allgemeine Elektronik, die Mess- und Datentechnik entwickelt, z.B.:

Analog-Digital-Umsetzer
Digital-Analog-Umsetzer
Speicher-Bausteine
Echtzeit-Uhren und Timer
Display-Treiber
Bausteine für Fernseh- und HIFI Geräte
Parallele I/O-Ports

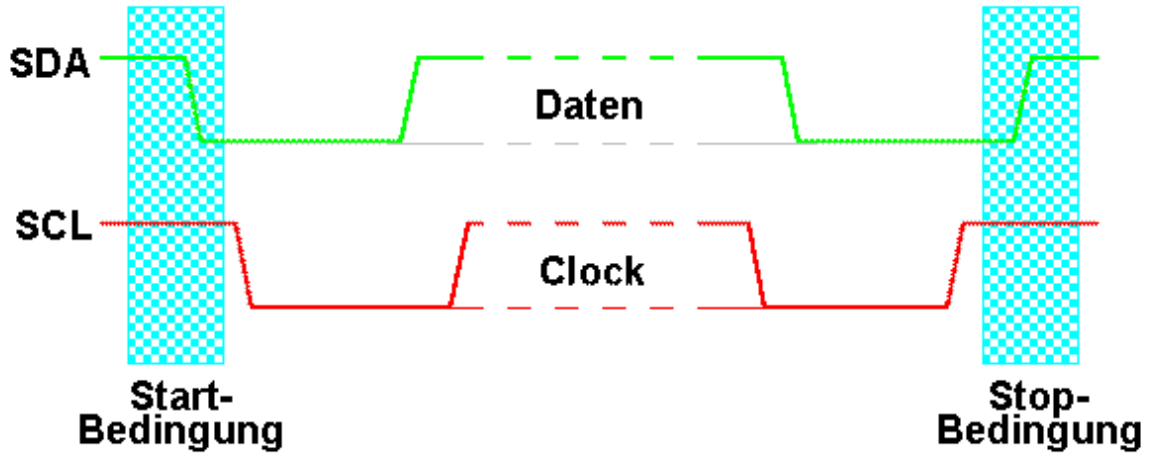
Viele der auf dem Markt befindlichen Mikrocontroller sind mit einer I2C-Bus-Schnittstelle versehen oder die Schnittstelle kann leicht durch Programmierung implementiert werden. Die Vielfalt der erhältlichen I2C-Bausteine und die Tatsache, daß dieser Bus mit nur zwei Leitungen auskommt, macht in für die computergestützte Mess- und Regeltechnik interessant.



Physikalisch handelt es sich um einen bidirektionalen seriellen Zweidraht-Bus, bei der eine der beiden Leitungen (Shift DAta - Leitung: SDA)zum Datentransport, die andere (Shift CLock-Leitung: SCL) zur Synchronisation des Datenverkehrs dient. Genau genommen benötigt man noch eine dritte Leitung, die Masseleitung (GND), die als Bezugspotential mitgeführt werden muß. An den Bus angeschlossen sind mindestens ein Master-Baustein, der den Datenverkehr auf dem Bus regelt und eine beliebige Anzahl Slave-Bausteine, die Daten vom Bus empfangen oder auf den Bus legen. Grundsätzlich ist der Master für die Generierung des Shift-Clock-Signals zuständig, mit dem die Daten auf der Shift-Data-Leitung getaktet werden. Die Slaves können entweder Empfänger (Receiver) von Daten, oder Sender (Transmitter) von Daten oder beides (Transceiver) sein. Ein typisches Receiver-Bausteil, also ein Baustein, der nur Daten empfängt, wäre etwa ein Anzeigentreiber. Ein EEPROM-Baustein wäre ein Beispiel für einen Transceiver, den es sollen ja sowohl Daten in den Speicherbaustein eingelesen als auch wieder ausgelesen werden.

Das I2C-Busprotokoll

Im I2C-Bus erfolgt die Datenübertragung bitseriell und synchron, d.h. jedes Datenbit auf der SDA-Leitung wird mit dem Takt der SCL-Leitung synchronisiert. Vor Beginn der Kommunikation haben beide Busleitungen SCL und SDA zunächst High-Pegel. Als nächstes erfolgt die Startbedingung. Die SDA-Leitung wird auf LOW-Pegel gezogen während die SCL-Leitung auf High-Pegel bleibt. Alle Bausteine empfangen nun die auf dem Bus anliegenden Daten.



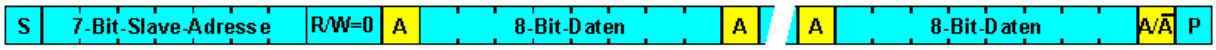
Nun können vom Master Daten auf den Bus gegeben werden. Dies erfolgt seriell auf die Leitung SDA und synchron mit dem vom Master erzeugten SCL-Impulsen. Dabei wird zunächst die Adresse des anzusprechenden Bausteins vom Master seriell auf die SDA-Leitung gelegt und mit SCL getaktet. Dabei beginnt der Master mit dem MSB und endet mit dem LSB. Der Baustein, dessen Adresse gesendet wurde, erkennt nun, dass er angesprochen wurde - der Bus "gehört" nun ausschließlich dem Master und dem angesprochenen Slave. Ferner ist durch den Wert des LSB bekannt, ob im darauffolgenden Byte der Slave Daten vom Master empfangen sollen (R/W=0), z.B. Steuersignale, oder ob Informationen vom Slave auf die SDA-Busleitung gegeben werden sollen (R/W=1). Datentransport erfolgt also 8-Bitweise (byteorientiert) seriell.

Generell quittiert der Empfänger von Daten den Empfang des Bytes mit einem ACKNOWLEDGE-Bit A=0. Dies erfolgt in der Weise, daß der Slave die SDA-Leitung auf LOW-Pegel zieht und dies vom Master im neunten SCL-Takt überprüft wird. Ist der adressierte Baustein z.B. nicht vorhanden, so wird die SDA-Leitung sich auf High-Pegel (Pull-Up-Widerstände !) befinden (Not Acknowledge: ACKNOWLEDGE-Bit A=1, SDA=high) und der Master kann eine Fehlerbehandlung einleiten. Wurde die Übertragung vom Slave ordnungsgemäß quittiert, kann der Master weiteren Daten byteweise an den Slave senden, wobei dieser den Empfang jedes Bytes mit ACKNOWLEDGE-Bit A=0 im neunten Shift-CLock-Takt bestätigt. Das Ende der Kommunikation wird durch die Herstellung der STOP-Bedingung erreicht. Sie ist dann gegeben, wenn die SDA-Leitung von LOW- auf HIGH-Pegel wechselt, während sich die SCL-Leitung auf High-Pegel befindet. Aus diesem Grunde darf bei der normalen Datenübertragung nur dann Pegelwechsel auf SDA erfolgen, wenn SCL Low-Pegel führt:

Die Datenübertragung zwischen den Busteilnehmern

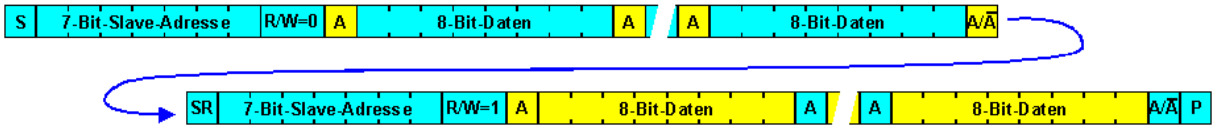
Hat der Master mit Startbedingung, Adressierung und R/W-Bit erst einmal einen Slave angesprochen, so kann in diesem Modus prinzipiell eine beliebige Zahl von Bytes vom Master zum Slave (R/W=0) oder vom Slave zum Master (R/W=1) übertragen werden. Dies erfolgt Byte-Weise, wobei der Empfänger (Slave oder Master) den Empfang mit einem LOW-Pegel auf der SDA-Leitung im neunten SCL-Puls bestätigt. Es sind folgende Fälle der Kommunikation zu unterscheiden;

Datenübertragung vom Master zum Slave



Datenübertragung vom Slave zum Master

Kombinierte Datenübertragung zwischen Master und Slave
Wechselt der R/W-Modus so sind vom Master erneut die Startbedingungen wiederherzustellen und der entsprechende Baustein neu zu adressieren. Erst dann kann wieder die Datenkommunikation auf SDA aufgenommen werden.



Der I2C Bus ist auch multimasterfähig, wobei die Arbitrierung gleich wie beim CAN Bus ohne Datenverlust erfolgt.

SPI BUS

Ein weiteres serielles Bussystem ist der so genannte SPI (Serial Peripheral Interface)-Bus. Ursprünglich von Motorola konzipiert gibt es keinen eigentlichen Standard dafür.

Es werden drei Leitungen
- SERIAL DATA OUT (MOSI)
- SERIAL DATA IN (MISO)
- SERIAL CLOCK (SCK)

und pro SLAVE-IC eine CS (CHIP SELECT) Leitung benötigt.
Jeder Teilnehmer verfügt über ein Schieberegister, in das Daten ein und ausgetaktet werden.

Zusätzlich unterscheidet man einen Master(-IC), der die Kommunikation steuert und den Bustakt auf die SCK Leitung legt, und den SLAVE(-IC), der die gesendeten Daten empfängt oder auf Anforderung des Masters Daten zurücksendet.
Bei dem SPI BUS ist einstellbar ob die Daten auf die steigende oder fallende Flanke übernommen werden sollen.
Außerdem kann auch noch der Ruhezustand des Taktes eingestellt werden.

Das MSB zuerst zu senden ist üblich.